macromedia®
COLDFUSION®
MX

CFML Reference

Part number: ZCF60M700

Project Management: Stephen M. Gilson

Writing: Christina Lamkin

Editing: Linda Adler and Noreen Maher

First Edition: May 2002

Macromedia, Inc.
600 Townsend St.
San Francisco, CA 94103

CONTENTS

ABOUT THIS BOOK

CFML Reference is your primary ColdFusion Markup Language (CFML) reference. Use this book to learn about CFML tags and functions, ColdFusion expressions, and using JavaScript objects for WDDX in Macromedia ColdFusion MX. It also provides detailed references for Java and C++ CFX interfaces.

Contents

Developer resources

Macromedia, Inc. is committed to setting the standard for customer support in developer education, documentation, technical support, and professional services. The Macromedia website is designed to give you quick access to the entire range of online resources. The following table shows the locations of these resources:

Resource	Description	URL
Macromedia website	General information about Macromedia products and services	http://www.macromedia.com
Information on ColdFusion	Detailed product information on ColdFusion and related topics	http://www.macromedia.com/coldfusion
Macromedia ColdFusion Support Center	Professional support programs that Macromedia offers	http://www.macromedia.com/support/coldfusion
ColdFusion Online Forum	Access to experienced ColdFusion developers through participation in the Online Forums, where you can post messages and read replies on many subjects relating to ColdFusion	http://webforums.macromedia.com/coldfusion/
Installation Support	Support for installation-related issues for all Macromedia products	http://www.macromedia.com/support/email/isupport
Training	Information about classes, on-site training, and online courses offered by Macromedia	http://www.macromedia.com/support/training
Developer Resources	All the resources that you need to stay on the cutting edge of ColdFusion development, including online discussion groups, Knowledge Base, technical papers, and more	http://www.macromedia.com/desdev/developer/
Reference Desk	Development tips, articles, documentation, and white papers	http://www.macromedia.com/v1/developer/TechnologyReference/index.cfm
Macromedia Alliance	Connection with the growing network of solution providers, application developers, resellers, and hosting services creating solutions with ColdFusion	http://www.macromedia.com/partners/

About Macromedia ColdFusion MX documentation

The ColdFusion documentation is designed to provide support for the complete spectrum of participants. The print and online versions are organized to let you quickly locate the information that you need. The ColdFusion online documentation is provided in HTML and Adobe Acrobat formats.

Printed and online documentation set

The ColdFusion documentation set consists of the following titles:

Book	Description
Installing ColdFusion MX	Describes system installation and basic configuration for Windows NT, Windows 2000, Solaris, Linux, and HP-UX.
Administering ColdFusion MX	Describes how to use the ColdFusion Administrator to manage the ColdFusion environment, including connecting to your data sources and configuring security for your applications,
Developing ColdFusion MX Applications with CFML	Describes how to develop your dynamic web applications, including retrieving and updating your data, using structures, and forms.
Getting Started Building ColdFusion MX Applications	Contains an overview of ColdFusion features and application development procedures. Includes a tutorial that guides you through the process of developing an example ColdFusion application.
Using Server-Side ActionScript in ColdFusion MX	Describes how Macromedia Flash movies executing on a client browser can call ActionScript code running on the ColdFusion server. Includes examples of server-side ActionScript and a syntax guide for developing ActionScript pages on the server.
Migrating ColdFusion 5 Applications	Describes how to migrate a ColdFusion 5 application to ColdFusion MX. This book describes the code compatibility analyzer that evaluates your ColdFusion 5 code to determine any incompatibilities within it.
CFML Reference	Provides descriptions, syntax, usage, and code examples for all ColdFusion tags, functions, and variables.
CFML Quick Reference	A brief guide that shows the syntax of ColdFusion tags, functions, and variables.
Working with Verity Tools	Describes Verity search tools and utilities that you can use for configuring the Verity K2 Server search engine, as well as creating, managing, and troubleshooting Verity collections.
Using ClusterCATS	Describes how to use Macromedia ClusterCATS, the clustering technology that provides load-balancing and failover services to assure high availability for your web servers.

Viewing online documentation

All ColdFusion documentation is available online in HTML and Adobe Acrobat Portable Document Format (PDF) files. To view the HTML documentation, open the following URL on the web server running ColdFusion: http://*web_root*/cfdocs/dochome.htm.

ColdFusion documentation in Acrobat format is available on the ColdFusion product CD-ROM.

Getting answers

One of the best ways to solve particular programming problems is to tap into the vast expertise of the ColdFusion developer communities on the ColdFusion Forums. Other developers on the forum can help you figure out how to do just about anything with ColdFusion. The search facility can also help you search messages from the previous 12 months, allowing you to learn how others have solved a problem that you might be facing. The Forums is a great resource for learning ColdFusion, but it is also a great place to see the ColdFusion developer community in action.

Contacting Macromedia

Corporate headquarters	Macromedia, Inc. 600 Townsend Street San Francisco, CA 94103 Tel: 415.252.2000 Fax: 415.626.0554 Web: http:// www.macromedia.com
Technical support	Macromedia offers a range of telephone and web-based support options. Go to http://www.macromedia.com/support/coldfusionfor a complete description of technical support services. You can make postings to the ColdFusion Support Forum (http://webforums.macromedia.com/coldfusion) at any time.
Sales	Toll Free: 888.939.2545 Tel: 617.219.2100 Fax: 617.219.2101 E-mail: sales@macromedia.com Web: http://www.macromedia.com/store

CHAPTER 1
Variables and Reserved Words

This chapter lists ColdFusion scope variables and reserved words.

Contents

Scope-specific built-in variables

ColdFusion returns variables, such as those returned in a `cfdirectory` or `cfftp` operation. A variable is usually referenced by **scoping** it according to its type: naming it according to the code context in which it is available; for example, Session.varname, or Application.varname.

You use the `cflock` tag to limit the scope of CFML constructs that modify shared shared data structures, files, and CFXs, to ensure that modifications occur sequentially. For more information, see "cflock" on page 186, and *Developing ColdFusion MX Applications with CFML*.

Variable scope

ColdFusion supports the Variables scope. Unscoped variables created with the `cfset` tag acquire the Variables scope by default. For example, the variable created by the statement `<CFSET linguist = Chomsky>` can be referenced as `#Variables.linguist#`

Caller scope

History New in ColdFusion MX: The Caller scope is accessible as a structure. (In earlier releases, it was not.)

Client variables

The following client variables are read-only:

```
Client.CFID
Client.CFToken
Client.HitCount
Client.LastVisit
Client.TimeCreated
Client.URLToken
```

Server variables

Use the Server prefix to reference server variables, as follows:

```
Server.ColdFusion.ProductName
Server.ColdFusion.ProductVersion
Server.ColdFusion.ProductLevel
Server.ColdFusion.SerialNumber
Server.ColdFusion.SupportedLocales
Server.OS.Name
Server.OS.AdditionalInformation
Server.OS.Version
Server.OS.BuildNumber
```

Application and session variables

To enable application and session variables, use the `cfapplication` tag. Reference them as follows:

```
Application.myvariable
Session.myvariable
```

To ensure that modifications to shared data occur in the intended sequence, use the `cflock` tag. For more infomation, see "cflock" on page 186.

The predefined application and session variables are as follows:

```
Application.ApplicationName
Session.CFID
Session.CFToken
Session.URLToken
```

Custom tag variables

A ColdFusion custom tag returns the following variables:

```
ThisTag.ExecutionMode
ThisTag.HasEndTag
ThisTag.GeneratedContent
ThisTag.AssocAttribs[index]
```

A custom tag can set a Caller variable to provide information to the caller. The Caller variable is set as follows:

```
<cfset Caller.variable_name = "value">
```

The calling page can access the variable with the `cfoutput` tag, as follows:

```
<cfoutput>#Caller.variable_name#</cfoutput>
```

Request variable

Request variables store data about the processing of one page request. Request variables store data in a structure that can be passed to nested tags, such as custom tags, and processed once.

To provide information to nested tags, set a Request variable, as follows:

```
<CFSET Request.field_name1 = "value">
<CFSET Request.field_name2 = "value">
<CFSET Request.field_name3 = "value">
...
```

Each nested tag can access the variable with the `cfoutput` tag, as follows:

```
<CFOUTPUT>#Request.field_name1#</CFOUTPUT>
```

Form variable

ColdFusion supports the Form variable FieldNames. FieldNames returns the names of the fields on a form. You use it on the action page associated with a form, as follows:

```
Form.FieldNames
```

ColdFusion tag-specific variables

Some ColdFusion tags return data as variables. For example, the `cffile` tag returns file size information in the FileSize variable, referenced as CFFILE.FileSize.

The following tags return data that can be referenced in variables:

```
cfcatch
cfdirectory
cferror
cffile
cfftp
cfhttp
cfindex
cfldap
cfmail
cfpop
cfquery
cfregistry
cfsearch
cfstoredproc
```

ColdFusion query variables

A ColdFusion tag that returns a query object supports the following variables, where *queryname* is the value of the `name` attribute:

```
queryname.CurrentRow
queryname.RecordCount
queryname.ColumnList
```

CFCATCH variables

Within a `cfcatch` block, the active exception properties can be accessed as the following variables:

```
CFCATCH.Type
CFCATCH.Message
CFCATCH.Detail
CFCATCH.ErrNumber
CFCATCH.NativeErrorCode
CFCATCH.SQLState
CFCATCH.LockName
CFCATCH.LockOperation
CFCATCH.MissingFileName
CFCATCH.TagContext
CFCATCH.ErrorCode
CFCATCH.ExtendedInfo
```

CFDIRECTORY variables

The `cfdirectory` tag, with `action=list`, returns a query object as follows, where *queryname* is the `name` attribute value:

```
queryname.Name
queryname.Size
queryname.Type
queryname.DateLastModified
queryname.Attributes
queryname.Mode
```

CFERROR variables

When `cferror` generates an error page, the following error variables are available if `type="request"`, `"exception"`, or `"monitor"`.

```
Error.Diagnostics
Error.MailTo
Error.DateTime
Error.Browser
Error.GeneratedContent
Error.RemoteAddress
Error.HTTPReferer
Error.Template
Error.QueryString
```

The following error variables are available if `type="validation"`.

```
Error.ValidationHeader
Error.InvalidFields
Error.ValidationFooter
```

Any `cfcatch` variable that applies to exception type can be accessed within the Error scope, as follows:

```
Error.Type
Error.Message
Error.Detail
Error.ErrNumber
Error.NativeErrorCode
Error.SQLState
Error.LockName
Error.LockOperation
Error.MissingFileName
Error.TagContext
Error.ErrorCode
Error.ExtendedInfo
```

Note: You can substitute the prefix CFERROR for Error, if `type = "Exception"` or `"Monitor"`; for example, CFERROR.Diagnostics, CFERROR.Mailto or CFERROR.DateTime.

CFFILE ACTION=Upload variables

File variables are read-only. Use the CFFILE prefix to reference file variables; for example, CFFILE.ClientDirectory. The File prefix is deprecated in favor of the CFFILE prefix.

```
CFFILE.AttemptedServerFile
CFFILE.ClientDirectory
CFFILE.ClientFile
CFFILE.ClientFileExt
CFFILE.ClientFileName
CFFILE.ContentSubType
CFFILE.ContentType
CFFILE.DateLastAccessed
CFFILE.FileExisted
CFFILE.FileSize
CFFILE.FileWasAppended
CFFILE.FileWasOverwritten
CFFILE.FileWasRenamed
CFFILE.FileWasSaved
CFFILE.OldFileSize
CFFILE.ServerDirectory
CFFILE.ServerFile
CFFILE.ServerFileExt
CFFILE.ServerFileName
CFFILE.TimeCreated
CFFILE.TimeLastModified
```

CFFTP error variables

When you use the `cfftp stoponerror` attribute, these variables are populated:

```
CFFTP.Succeeded
CFFTP.ErrorCode
CFFTP.ErrorText
```

CFFTP ReturnValue variable

Some `cfftp` file and directory operations provide a return value, in the variable CFFTP.ReturnValue. Its value is determined by the results of the `action` attribute. When you specify any of the following actions, `cfftp` returns a value:

```
GetCurrentDir
GetCurrentURL
ExistsDir
ExistsFile
Exists
```

CFFTP query object columns

When you use the `cfftp` tag with the `listdir` action, `cfftp` returns a query object, where *queryname* is the `name` attribute value, and *row* is the row number of each file or directory entry:

queryname.Name[*row*]
queryname.Path[*row*]
queryname.URL[*row*]
queryname.Length[*row*]
queryname.LastModified[*row*]
queryname.Attributes
queryname.IsDirectory
queryname.Mode

CFHTTP variables

A `cfhttp get` operation can return text and binary files. Files are downloaded and the contents stored in a variable or file, depending on the MIME type, as follows:

CFHTTP.FileContent
CFHTTP.MimeType
CFHTTP.Header
CFHTTP.ResponseHeader*[http_hd_key]*
CFHTTP.StatusCode

CFLDAP variables

The `cfldap action=query` tag returns information about the LDAP query, as follows:

queryname.CurrentRow
queryname.RecordCount
queryname.ColumnList

CFPOP variables

The `cfpop` tag returns the following result columns, depending on the `action` attribute value and the use of other attributes, such as `attachmentpath`, where *queryname* is the `name` attribute value:

queryname.Date
queryname.From
queryname.Body
queryname.Header
queryname.MessageNumber
queryname.ReplyTo
queryname.Subject
queryname.CC
queryname.To
queryname.CurrentRow
queryname.RecordCount
queryname.ColumnList
queryname.Attachments
queryname.AttachmentFiles

CFQUERY and CFSTOREDPROC variables

The `cfquery` tag returns information about the query in this variable:

```
CFQUERY.ExecutionTime
```

The `cfquery` tag uses the query name to scope the following data about the query:

```
queryname.CurrentRow
queryname.RecordCount
queryname.ColumnList
```

The `cfstoredproc` tag returns the following variables:

```
CFSTOREDPROC.ExecutionTime
CFSTOREDPROC.StatusCode
```

CFREGISTRY variables

The `cfregistry` tag returns a query record set that you can reference after executing the `GetAll` action, as follows, where *queryname* is the `name` attribute value:

```
queryname.Entry
queryname.Type
queryname.Value
```

CFSEARCH variables

A `cfsearch` operation returns the following variables, where *searchname* is the `name` attribute value:

```
searchname.URL
searchname.Key
searchname.Title
searchname.Score
searchname.Custom1 and Custom2
searchname.Summary
searchname.RecordCount
searchname.CurrentRow
searchname.RecordsSearched
searchname.ColumnList
```

Standard CGI variables

This section lists the CGI 1.1 variables that some web servers create when a CGI script is called.

The CGI variables that are available for your use vary with the web server and configuration. Some of the following variables may not be available.

Request

```
CGI.AUTH_TYPE
CGI.CONTENT_LENGTH
CGI.CONTENT_TYPE
CGI.PATH_INFO
CGI.PATH_TRANSLATED
CGI.QUERY_STRING
CGI.REMOTE_ADDR
CGI.REMOTE_HOST
CGI.REMOTE_USER
CGI.REQUEST_METHOD
CGI.SCRIPT_NAME
```

Server

```
CGI.GATEWAY_INTERFACE
CGI.SERVER_NAME
CGI.SERVER_PORT
CGI.SERVER_PROTOCOL
CGI.SERVER_SOFTWARE
```

Client

```
CGI.CERT_ISSUER
CGI.CERT_SUBJECT
CGI.CLIENT_CERT_ENCODED
CGI.HTTP_ACCEPT
CGI.HTTP_IF_MODIFIED_SINCE
CGI.HTTP_USER_AGENT
```

The `CERT_ISSUER`, `CERT_SUBJECT`, `CLIENT_CERT_ENCODED` variables are available only when you use client certificates.

CGI environment variables

When a browser makes a request to a server, the web server and the browser create environment variables. In ColdFusion, these variables are referred to as **CGI environment variables**. They take the CGI prefix regardless of whether the server uses a server API or CGI to communicate with the ColdFusion Server.

Environment variables contain data about the transaction between the browser and the server, such as the IP Address, browser type, and authenticated username. You can reference CGI environment variables for a given page request anywhere in the page. CGI variables are read-only.

Note: The environment variables available to applications depend on the browser and server software.

Testing for CGI variables

Because some browsers do not support some CGI variables, ColdFusion always returns True when it tests for the existence of a CGI variable, regardless of whether the browser supports the variable. To determine if the CGI variable is available, test for an empty string, as shown in the following example:

```
<cfif CGI.varname IS NOT "">
  CGI variable exists
<cfelse>
  CGI variable does not exist
</cfif>
```

CGI server variables

The following table describes common CGI environment variables that the server creates (some of these are not available with some servers):

CGI server variable	Description
SERVER_SOFTWARE	Name and version of the information server software answering the request (and running the gateway). Format: name/version.
SERVER_NAME	Server's hostname, DNS alias, or IP address as it appears in self-referencing URLs.
GATEWAY_INTERFACE	CGI specification revision with which this server complies. Format: CGI/revision.
SERVER_PROTOCOL	Name and revision of the information protocol this request came in with. Format: protocol/revision.
SERVER_PORT	Port number to which the request was sent.
REQUEST_METHOD	Method with which the request was made. For HTTP, this is Get, Head, Post, and so on.
PATH_INFO	Extra path information, as given by the client. Scripts can be accessed by their virtual pathname, followed by extra information at the end of this path. The extra information is sent as PATH_INFO.

CGI server variable	Description
PATH_TRANSLATED	Translated version of PATH_INFO after any virtual-to-physical mapping.
SCRIPT_NAME	Virtual path to the script that is executing; used for self-referencing URLs.
QUERY_STRING	Query information that follows the ? in the URL that referenced this script.
REMOTE_HOST	Hostname making the request. If the server does not have this information, it sets REMOTE_ADDR and does not set REMOTE_HOST.
REMOTE_ADDR	IP address of the remote host making the request.
AUTH_TYPE	If the server supports user authentication, and the script is protected, the protocol-specific authentication method used to validate the user.
REMOTE_USER AUTH_USER	If the server supports user authentication, and the script is protected, the username the user has authenticated as. (Also available as AUTH_USER.)
REMOTE_IDENT	If the HTTP server supports RFC 931 identification, this variable is set to the remote username retrieved from the server. Use this variable for logging only.
CONTENT_TYPE	For queries that have attached information, such as HTTP POST and PUT, this is the content type of the data.
CONTENT_LENGTH	Length of the content as given by the client.

CGI client variables

The following table describes common CGI environment variables the browser creates and passes in the request header:

CGI client variable	Description
HTTP_REFERER	The referring document that linked to or submitted form data.
HTTP_USER_AGENT	The browser that the client is currently using to send the request. Format: software/version library/version.
HTTP_IF_MODIFIED_SINCE	The last time the page was modified. The browser determines whether to set this variable, usually in response to the server having sent the LAST_MODIFIED HTTP header. It can be used to take advantage of browser-side caching.

CGI client certificate variables

ColdFusion makes available the following client certificate data. These variables are available when running Microsoft IIS 4.0 or Netscape Enterprise under SSL if your web server is configured to accept client certificates.

CGI client certificate variable	Description
CERT_SUBJECT	Client-specific information provided by the web server. This data typically includes the client's name, e-mail address, etc. For example: `O = "VeriSign, Inc.", OU = VeriSign Trust Network, OU = "www.verisign.com/repository/RPA Incorp. by Ref.,LIAB.LTD(c)98", OU = Persona Not Validated, OU = Digital ID Class 1 - Microsoft, CN = Matthew Lund, E = mlund@macromedia.com`
CERT_ISSUER	Information about the authority that provided the client certificate. For example: `O = "VeriSign, Inc.", OU = VeriSign Trust Network, OU = "www.verisign.com/repository/RPA Incorp. By Ref.,LIAB.LTD(c)98", CN = VeriSign Class 1 CA Individual Subscriber-Persona Not Validated`
CLIENT_CERT_ENCODED	The entire client certificate binary, base-64 encoded. This data is typically of interest to developers,so they can integrate with other software that uses client certificates.

CHAPTER 2
ColdFusion Tag Summary

This chapter lists ColdFusion Markup Language (CFML) tags by functional category.

Contents

New tags, attributes, and values

Tag	Attribute or value	New in this ColdFusion Server release
cfargument	All	ColdFusion MX
cfcache	timespan attribute	ColdFusion MX
	cachedirectory attribute	ColdFusion MX
cfchart	All	ColdFusion MX
cfchartdata	All	ColdFusion MX
cfchartseries	All	ColdFusion MX
cfcollection	list value of action attribute	ColdFusion MX
	name attribute	ColdFusion MX
cfcomponent	All	ColdFusion MX
cffunction	All	ColdFusion MX
cfhttp	firstRowAsHeaders attribute	ColdFusion MX
	charset attribute	ColdFusion MX
cfimport	All	ColdFusion MX
cfinvoke	All	ColdFusion MX
cfinvokeargument	All	ColdFusion MX
cflogin	All	ColdFusion MX
cfloginuser	All	ColdFusion MX
cflogout	All	ColdFusion MX
cfmail	spoolEnable attribute	ColdFusion MX
cfobject	All	ColdFusion MX
cfobjectcache	All	ColdFusion MX
cfprocessingdirective	pageEncoding attribute	ColdFusion MX
cfproperty	All	ColdFusion MX
cfreturn	All	ColdFusion MX
cfsetting	requestTimeOut attribute	ColdFusion MX
cfthrow	object attribute	ColdFusion MX
cftrace	All	ColdFusion MX
cfxml	All	ColdFusion MX

Deprecated tags, attributes, and values

The following tags, attributes, and attribute values are deprecated. Do not use them in ColdFusion applications. They might not work, and might cause an error, in releases later than ColdFusion MX.

Tag	Attribute or value	Deprecated as of this ColdFusion Server release
cfcache	timeout attribute	ColdFusion MX
cffile	attributes attribute value archive	ColdFusion MX
	attributes attribute value system	ColdFusion MX
	attributes attribute value temporary	ColdFusion MX
cfgraph	All	ColdFusion MX
cfgraphdata	All	ColdFusion MX
cfgridupdate	connectString attribute	ColdFusion MX
	dbName attribute	ColdFusion MX
	dbServer attribute	ColdFusion MX
	dbType attribute	ColdFusion MX
	provider attribute	ColdFusion MX
	providerDSN attribute	ColdFusion MX
cfindex	external attribute	ColdFusion MX
cfinsert	connectString attribute	ColdFusion MX
	dbName attribute	ColdFusion MX
	dbServer attribute	ColdFusion MX
	dbType attribute	ColdFusion MX
	provider attribute	ColdFusion MX
	providerDSN attribute	ColdFusion MX
cfldap	filterConfig attribute	ColdFusion MX
	filterFile attribute	ColdFusion MX
cflog	date attribute value "No"	ColdFusion MX
	thread attribute value "No"	ColdFusion MX
	time attribute value "No"	ColdFusion MX

Tag	Attribute or value	Deprecated as of this ColdFusion Server release
cfquery	connectString attribute	ColdFusion MX
	dbName attribute	ColdFusion MX
	dbServer attribute	ColdFusion MX
	The following dbType attribute values: • dynamic • ODBC • Oracle73 • Oracle80 • Sybase11 • OLEDB • DB2	ColdFusion MX (The value query is valid.)
	provider attribute	ColdFusion MX
	providerDSN attribute	ColdFusion MX
	sql attribute	ColdFusion MX
cfregistry	All, on UNIX only	ColdFusion MX
cfsearch	external attribute	ColdFusion MX
cfservlet	All	ColdFusion MX
cfservletparam	All	ColdFusion MX
cfslider	img attribute	ColdFusion MX
	imgStyle attribute	ColdFusion MX
	grooveColor attribute	ColdFusion MX
cfstoredproc	connectString attribute	ColdFusion MX
	dbName attribute	ColdFusion MX
	dbServer attribute	ColdFusion MX
	provider attribute	ColdFusion MX
	providerDSN attribute	ColdFusion MX
cfupdate	connectString attribute	ColdFusion MX
	dbName attribute	ColdFusion MX
	dbServer attribute	ColdFusion MX
	provider attribute	ColdFusion MX
	providerDSN attribute	ColdFusion MX

Obsolete tags, attributes, and values

The following tags, attributes, and attribute values are obsolete. Do not use them in ColdFusion applications. They do not work, and might cause an error, in releases later than ColdFusion 5.

Tag	Attribute or value	Obsolete as of this ColdFusion Server release
`cfauthenticate`	All	ColdFusion MX
`cfimpersonate`	All	ColdFusion MX
`cfindex`	`action` attribute value `optimize`	ColdFusion MX
`cfinternaladminsecurity`	All	ColdFusion MX This tag did not appear in *CFML Reference*.
`cfnewinternaladminsecurity`	All	ColdFusion MX This tag did not appear in *CFML Reference*.
`cfsetting`	`catchExceptionsByPattern` attribute	ColdFusion MX

Rules for embedding comments in CFML code

New in ColdFusion MX: You can embed a CFML comment within the following ColdFusion elements:

- A start tag
- A tag
- A custom tag
- A function call
- Variable text that is within pound signs

ColdFusion ignores the comment. (In earlier releases, comments within these constructs were not allowed.)

The following examples show **valid embedded CFML comments**:

```
<cfif true eq true <!--- comment --->>
<cfoutput>myVar: #myVar<!--- comment within pound signs--->#</cfoutput>
#IncrementValue(120<!--- comment next to argument--->)#
<cf_comment_pos_05 <!--- comment within custom-tag brackets --->>
<cfset foo = evaluate("1 + 2<!--- comment --->")>
isDefined(dynamicVariable<!--- comment --->)
#isDefined<!--- comment --->("MyConstruct")#
<cfset myLVar<!--- comment ---> = "myLVar&lt;!--- comment ---&gt;">
```

The following examples show **invalid or erroneous embedded CFML comments**:

```
isDefined("myVariable<!--- no comment within name string --->")
is<!--- no comment within function name --->Defined("myVariable")
<cfout<!--- no comment within tag name --->put "#myVariable#">
```

Tag summary

ColdFusion Markup Language (CFML) includes a set of tags that you use in ColdFusion pages to interact with data sources, manipulate data, and display output. CFML tag syntax is similar to HTML element syntax.

The following table describes CFML tags:

CFML tag	Category	Description
cfabort	Flow-control tags	Stops the processing of a ColdFusion page at the tag location
cfapplet	Forms tags	Embeds Java applets in a cfform tag
cfapplication	Application framework tags	Defines an application name; activates client variables; specifies client variable storage mechanism
cfargument	Extensibility tags	Creates a parameter definition within a component definition; defines a function argument
cfassociate	Application framework tags	Enables subtag data to be saved with a base tag
cfbreak	Flow-control tags	Breaks out of a CFML looping construct
cfcache	Page processing tags	Caches ColdFusion pages
cfcase	Flow-control tags	Used with the cfswitch and cfdefaultcase tags
cfcatch	Exception handling tags, Flow-control tags	Catches exceptions in ColdFusion pages
cfchart	Data output tags	Generates and displays a chart
cfchartdata	Data output tags	Defines chart data points
cfchartseries	Data output tags	Defines style in which chart data displays
cfcol	Data output tags	Defines table column header, properties
cfcollection	Extensibility tags	Administers Verity collections
cfcomponent	Extensibility tags	Creates and defines a component object
cfcontent	Data output tags, Page processing tags	Defines content type and filename of a file to be downloaded by current page
cfcookie	Variable manipulation tags	Defines and sets cookie variables, including expiration and security options
cfdefaultcase	Flow-control tags	Receives control if there is no matching cfcase tag value

CFML tag	Category	Description
cfdirectory	File management tags	Performs typical directory-handling tasks from within a ColdFusion application
cfdump	Debugging tags, Variable manipulation tags	Outputs variables for debugging
cfelse	Flow-control tags	Creates IF-THEN-ELSE constructs
cfelseif	Flow-control tags	Creates IF-THEN-ELSE constructs
cferror	Exception handling tags, Application framework tags	Displays custom HTML error pages when errors occur
cfexecute	Flow-control tags, Extensibility tags	Executes developer-specified process on server computer
cfexit	Flow-control tags	Aborts processing of executing CFML tag
cffile	File management tags	Performs typical file-handling tasks from within ColdFusion application
cfflush	Data output tags, Page processing tags	Flushes currently available data to client
cfform	Forms tags	Builds input form; performs client-side input validation
cfftp	Forms tags, Extensibility tags, Internet Protocol tags	Permits FTP file operations
cffunction	Extensibility tags	Defines function that you build in CFML
cfgrid	Forms tags	Displays tabular grid control, in cfform tag
cfgridcolumn	Forms tags	Used in cfform; defines columns in a cfgrid
cfgridrow	Forms tags	Defines a grid row; used with cfgrid
cfgridupdate	Forms tags	Directly updates ODBC data source from edited grid data
cfheader	Data output tags, Page processing tags	Generates HTTP headers
cfhtmlhead	Forms tags, Page processing tags	Writes text and HTML to HEAD section of page
cfhttp	Forms tags, Internet Protocol tags	Performs GET and POST to upload file or post form, cookie, query, or CGI variable directly to server
cfhttpparam	Forms tags, Internet Protocol tags	Specifies parameters required for a cfhttp POST operation; used with cfhttp

CFML tag	Category	Description
cfif	Flow-control tags	Creates IF-THEN-ELSE constructs
cfimport	Application framework tags	Imports JSP tag libraries into a CFML page
cfinclude	Flow-control tags	Embeds references to ColdFusion pages
cfindex	Extensibility tags,	Creates Verity search indexes
cfinput	Forms tags	Creates an input element (radio button, checkbox, text entry box); used in cfform
cfinsert	Database manipulation tags	Inserts records in a data source
cfinvoke	Extensibility tags	invokes component methods from a ColdFusion page or component
cfinvokeargument	Extensibility tags	Passes a parameter to a component method or a web service
cfldap	Forms tags, Internet Protocol tags	Provides access to LDAP directory servers
cflocation	Flow-control tags	Controls execution of a page
cflock	Application framework tags	Ensures data integrity and synchronizes execution of CFML code
cflog	Data output tags, Other tags	Writes a message to a log file
cflogin	Extensibility tags	Defines a container for user login and authentication code
cfloginuser	Extensibility tags	Identifies an authenticated user to ColdFusion
cflogout	Extensibility tags	Logs the current user out
cfloop	Flow-control tags	Repeats a set of instructions based on conditions
cfmail	Forms tags, Internet Protocol tags	Assembles and posts an e-mail message
cfmailparam	Forms tags, Internet Protocol tags	Attaches a file or adds a header to an e-mail message
cfmodule	Application framework tags	Invokes a custom tag for use in a ColdFusion page
cfobject	Extensibility tags	Creates COM, component, CORBA, Java and web service objects
cfobjectcache	Database manipulation tags	Flushes the query cache
cfoutput	Data output tags	Displays the output of a database query or other operation

CFML tag	Category	Description
cfparam	Variable manipulation tags	Defines a parameter and its default value
cfpop	Forms tags, Internet Protocol tags	Gets and deletes messages from POP mail server
cfprocessingdirective	Data output tags	Suppresses white space and other output
cfprocparam	Database manipulation tags	Holds parameter information for stored procedure
cfprocresult	Database manipulation tags	Result set name that ColdFusion tags use to access result set of a stored procedure
cfproperty	Extensibility tags	Defines components
cfquery	Database manipulation tags	Passes SQL statements to a database
cfqueryparam	Database manipulation tags	Checks data type of a query parameter
cfregistry	Other tags, Variable manipulation tags	Reads, writes, and deletes keys and values in a Windows system registry
cfreport	Exception handling tags	Embeds a Crystal Reports report
cfrethrow	Exception handling tags	Rethrows currently active exception
cfreturn	Extensibility tags	Returns results from a component method
cfsavecontent	Variable manipulation tags	Saves generated content inside tag body in a variable
cfschedule	Variable manipulation tags	Schedules page execution; optionally, produces static pages
cfscript	Application framework tags	Encloses a set of cfscript statements
cfsearch	Extensibility tags	Executes searches against data indexed in Verity collections, using cfindex
cfselect	Forms tags	Creates a drop-down list box form element; used in cfform tag
cfset	Variable manipulation tags	Defines a variable
cfsetting	Other tags, Variable manipulation tags	Defines and controls ColdFusion settings
cfsilent	Data output tags, Page processing tags	Suppresses CFML output within tag scope
cfslider	Forms tags	Creates slider control; used in cfform

CFML tag	Category	Description
cfstoredproc	Database manipulation tags	Holds database connection information; identifies a stored procedure to execute
cfswitch	Flow-control tags	Evaluates passed expression; passes control to matching cfcase tag
cftable	Data output tags	Builds a table in a ColdFusion page
cftextinput	Forms tags	Puts a one-line text entry box in a form
cfthrow	Exception handling tags, Flow-control tags	Throws a developer-specified exception
cftrace	Debugging tags	Displays and logs application debugging data
cftransaction	Database manipulation tags	Groups cfquery operations into one transaction; performs rollback processing
cftree	Forms tags	Creates tree control element; used in cfform
cftreeitem	Forms tags	Populates a tree control element in a form; used with cftree
cftry	Exception handling tags, Flow-control tags	Catches exceptions in ColdFusion pages
cfupdate	Database manipulation tags	Updates rows in a database data source
cfwddx	Extensibility tags	Serializes and de-serializes CFML data structures to XML-based WDDX format
cfxml	Extensibility tags	Creates an XML document object

Application framework tags

cfapplication
cfassociate
cferror
cfimport
cflock
cfmodule
cfscript

Database manipulation tags

cfinsert
cfobjectcache
cfprocparam
cfprocresult
cfquery
cfqueryparam
cfstoredproc
cftransaction
cfupdate

Data output tags

cfchart
cfchartdata
cfchartseries
cfcol
cfcontent
cfflush
cfheader
cflog
cfoutput
cfprocessingdirective
cfsilent
cftable

Debugging tags

cfdump
cftrace

Exception handling tags

cfcatch
cferror
cfreport
cfrethrow
cfthrow
cftry

Extensibility tags

cfchart
cfchartdata
cfchartseries
cfcollection
cfcomponent
cfexecute
cfftp
cffunction
cfindex
cfinvoke
cfinvokeargument
cflogin
cfloginuser
cflogout
cfobject
cfproperty
cfreport
cfreturn
cfsearch
cfwddx
cfxml

File management tags

cfdirectory
cffile
cfftp

Flow-control tags

cfabort
cfbreak
cfcase
cfdefaultcase
cfelse
cfelseif
cfexecute
cfexit
cfif
cfinclude
cflocation
cfloop
cfrethrow
cfswitch
cfthrow
cftry

Forms tags

cfapplet
cfform
cfgrid
cfgridcolumn
cfgridrow
cfgridupdate
cfhtmlhead
cfinput
cfldap
cfmail
cfmailparam
cfpop
cfselect
cfslider
cftextinput
cftree
cftreeitem

Internet Protocol tags

cfftp
cfhttp
cfhttpparam
cfldap
cfmail
cfmailparam
cfpop

Page processing tags

cfcache
cfcontent
cfflush
cfheader
cfhtmlhead
cfinclude
cfsetting
cfsilent

Variable manipulation tags

cfcookie
cfdump
cfparam
cfregistry
cfsavecontent
cfschedule
cfset
cfsetting

Other tags

cflog
cfregistry

CHAPTER 3
ColdFusion Tags

This chapter describes ColdFusion Markup Language (CFML) tags. The tags are listed alphabetically.

cfabort

Description
Stops the processing of a ColdFusion page at the tag location. ColdFusion returns everything that was processed before the tag. The tag is often used with conditional logic to stop processing a page when a condition occurs.

Category
Flow-control tags

Syntax
```
<cfabort
   showError = "error_message">
```

See also
cfbreak, cfexecute, cfexit, cfif, cflocation, cfloop, cfswitch, cfthrow, cftry

Attributes

Attribute	Req/ Opt	Default	Description
showError	Optional		Error to display, in a standard ColdFusion error page, when tag executes.

Usage
When you use the cfabort and cferror tags together, the cfabort tag halts processing immediately; the cferror tag redirects output to a specified page.

If this tag does not contain a showError attribute value, processing stops when the tag is reached and ColdFusion returns the page contents up to the line that contains the cfabort tag.

When you use this tag with the showError attribute, but do not define an error page using cferror, page processing stops when the cfabort tag is reached. The message in showError displays to the client.

When you use this tag with the showError attribute and an error page using cferror, ColdFusion redirects output to the error page specified in the cferror tag.

Example
```
<!--- this example shows the use of cfabort to stop processing.
In the second example, where cfabort is used, the result never displays --->

<h3>Example A: Let the instruction complete itself</h3>
<!--- first, set a variable --->
<cfset myVariable = 3>
<!--- now, perform a loop that increments this value --->
<cfloop from = "1" to = "4" index = "Counter">
   <cfset myVariable = myVariable + 1>
</cfloop>

<cfoutput>
<p>The value of myVariable after incrementing through the loop #Counter# times is:
         #myVariable#
</cfoutput>

<h3>Example B: Use cfabort to halt the instruction</h3>
<!--- reset the variable and show the use of cfabort --->
<cfset myVariable = 3>
```

```
<!--- now, perform a loop that increments this value --->
<cfloop from = "1" to = "4" index = "Counter">
   <!--- on the second time through the loop, cfabort --->
   <cfif Counter is 2>
      <cfabort>
   <!--- processing is stopped, and subsequent operations are not carried out --->
   <cfelse>
   <cfset myVariable = myVariable + 1>
   </cfif>
</cfloop>

<cfoutput>
<p>The value of myVariable after incrementing through the loop
        #counter# times is: #myVariable#
</cfoutput>
```

cfapplet

Description

This tag references a registered custom Java applet. To register a Java applet, in the ColdFusion Administrator, click **Extensions > Java Applets**.

Using this tag within a `cfform` tag is optional. If you use it within `cfform`, and the `method` attribute is defined in the Administrator, the return value is incorporated into the form.

Category

Forms tags

Syntax

```
<cfapplet
   appletSource = "applet_name"
   name = "form_variable_name"
   height = "height_in_pixels"
   width = "width_in_pixels"
   vSpace = "space_above_and_below_in_pixels"
   hSpace = "space_on_each_side_in_pixels"
   align = "alignment_option"
   notSupported = "message_to_display_for_nonJava_browser"
   param_1 = "applet_parameter_name"
   param_2 = "applet_parameter_name"
   param_n = "applet_parameter_name">
```

See also

`cfform`, `cfobject`, `cfservlet`

History

New in ColdFusion MX: Using this tag within a `cfform` tag is optional. If you use it within `cfform`, and the `method` attribute is defined in the Administrator, the return value of the applet's method is incorporated into the form.

Attributes

Attribute	Req/Opt	Default	Description
appletSource	Required		Name of registered applet
name	Required		Form variable name for applet
height	Optional		Height of applet, in pixel
width	Optional		Width of applet, in pixels
vSpace	Optional		Space above and below applet, in pixels
hSpace	Optional		Space on left and right of applet, in pixels
align	Optional		Alignment: • Left • Right • Bottom • Top • TextTop • Middle • AbsMiddle • Baseline • AbsBottom

Attribute	Req/Opt	Default	Description
notSupported	Optional	See Description	Text to display if a page that contains a Java applet-based cfform control is opened by a browser that does not support Java or has Java support disabled. For example: notSupported = "Browser must support Java to view ColdFusion Java Applets" Default: Browser must support Java to view ColdFusion Java Applets!
param_n	Optional		Registered parameter for applet. Specify only to override values for applet in ColdFusion Administrator.

Usage

You can specify the applet `method` attribute only in the Administrator, Java Applets view. For other attributes, you can accept the default values in the Administrator view, or specify values in this tag and override the defaults.

If Java applet components are stored in a JAR file, enter the file name in the ColdFusion Administrator, Java Applets window, Archive text box.

Example

```
<p>cfapplet lets you reference custom Java applets that have been
   registered using the ColdFusion Administrator.
<p>To register a Java applet, open the ColdFusion Administrator and
   click "Applets" link under "extensions" section.
<p>This example applet copies text that you type into a form. Type
   some text, and then click "copy" to see the copied text.

<cfform action = "index.cfm">
   <cfapplet appletsource = "copytext" name = "copytext">
</cfform>
```

cfapplication

Description — Defines the scope of a ColdFusion application; enables and disables storage of Client variables; specifies the Client variable storage mechanism; enables Session variables; and sets Application variable timeouts.

Category — Application framework tags

Syntax

```
<cfapplication
   name = "application_name"
   clientManagement = "Yes" or "No"
   clientStorage = "datasource_name" or "Registry" or "Cookie"
   setClientCookies = "Yes" or "No"
   sessionManagement = "Yes" or "No"
   sessionTimeout = #CreateTimeSpan(days, hours, minutes, seconds)#
   applicationTimeout = #CreateTimeSpan(days, hours,minutes, seconds)#
   setDomainCookies = "Yes" or "No">
```

See also — cfassociate, cferror, cflock, cfmodule

History — New in ColdFusion MX: Server, Session, and Application scope variables are stored in memory as structures. In earlier releases, only Session and Application scope variables were stored this way. You cannot access the UDF function scope as a structure.

New in ColdFusion MX: ColdFusion sets the CFTOKEN variable value using this algorithm: if the registry key UUIDToken is a non-zero value, ColdFusion uses a number constructed from the UUID plus a random number. Otherwise, ColdFusion sets the CFTOKEN variable default value using a positive random integer. (In earlier releases, ColdFusion always used a number constructed from the UUID plus a random number.)

Attributes

Attribute	Req/Opt	Default	Description
name	See Description		Name of application. Up to 64 characters. For Application and Session variables: Required. For Client variables: Optional
clientManagement	Optional	No	• Yes: enables client variables • No
clientStorage	Optional	registry	How client variables are stored: • datasource_name: in ODBC or native data source. You must create storage repository in the Administrator. • registry: in the system registry. • cookie: on client computer in a cookie. Scalable. If client disables cookies in the browser, client variables do not work.

Attribute	Req/Opt	Default	Description
setClientCookies	Optional	Yes	• Yes: enables client cookies • No: ColdFusion does not automatically send CFID and CFTOKEN cookies to client browser; you must manually code CFID and CFTOKEN on the URL for every page that uses Session or Client variables.
sessionManagement	Optional	No	• Yes: enables session variables • No
sessionTimeout	Optional	Specified in Variables page of ColdFusion Administrator	Lifespan of session variables. CreateTimeSpan function and values in days, hours, minutes, and seconds, separated by commas.
applicationTimeout	Optional	Specified in Variables page of ColdFusion Administrator	Lifespan of application variables. CreateTimeSpan function and values in days, hours, minutes, and seconds, separated by commas.
setDomainCookies	Optional	No	• Yes: Sets CFID and CFTOKEN cookies for a domain (not a host). Required, for applications running on clusters. • No

Usage

This tag is typically used in the Application.cfm file, to set defaults for a ColdFusion application.

This tag enables application variables, unless they are disabled in the ColdFusion Administrator. The Administrator setting also overrides the `sessionManagement` attribute. For more information, see *Administering ColdFusion MX*.

Server, application, and session variables

When you display, set, or update variables in the server, application, and session scopes, use the `cflock` tag with the `scope` attribute set to the following value:

- For server variables, specify `"server"`
- For application variables, specify `"application"`
- For session variables, specify `"session"`

For information about locking scopes, see cflock on page 186.

If ColdFusion is running on a cluster, you must specify `clientStorage = "cookie"` or a data source name; you cannot specify `"registry"`.

If you use this tag to activate the Application and Client scopes, ColdFusion saves the application name as a key, whose maximum length is 64 characters. If an application name is longer than this, the client store fails during database processing.

The CFTOKEN variable is 8 bytes in length. Its range is 10000000 —99999999.

Example

```
<!--- This example shows how to use cflock to guarantee consistent data
        updates to variables in Application, Server, and Session scopes. --->
  <h3>cfapplication Example</h3>
  <p>cfapplication defines scoping for a ColdFusion application and
        enables or disables the storing of application and/or sessionvariables.
        This tag is placed in a special file calledApplication.cfm that is run
        before any other CF page in a directory where the Application.cfm file
        appears.

  <cfapplication name = "ETurtle"
    sessionTimeout = #CreateTimeSpan(0, 0, 0, 60)#
    sessionManagement = "Yes">
  <!--- Initialize session and application variables used by E-Turtleneck.
        Use session scope for session variables. --->
  <cflock scope = "Session" timeout = "30" type = "Exclusive">
    <cfif NOT IsDefined("session.size")>
      <cfset session.size = "">
    </cfif>
    <cfif NOT IsDefined("session.color")>
      <cfset session.color = "">
    </cfif>
  </cflock>
  <!--- Use the application scope for the application variable. This variable
        keeps track of total number of turtlenecks sold. --->
  <cflock scope = "Application" timeout = "30" type = "Exclusive">
    <cfif NOT IsDefined("application.number")>
      <cfset application.number = 1>
    </cfif>
  </cflock>
  <cflock scope = "Application" timeout = "30" type = "readOnly">
    <cfoutput>
    E-Turtleneck is proud to say that we have sold #application.number#
        turtlenecks to date.
    </cfoutput>
  </cflock>
<!--- End of Application.cfm --->
```

cfargument

Description Creates a parameter definition within a component definition. Defines a function argument. Used within a `cffunction` tag.

Category Extensibility tags

Syntax
```
<cfargument
   name="..."
   type="..."
   required="..."
   default="..."
   ...>
```

See also `cfcomponent`, `cffunction`, `cfinvoke`, `cfinvokeargument`, `cfobject`, `cfproperty`, `cfreturn`

Attributes

Attribute	Req/Opt	Default	Description
name	Required		String; an argument name.
type	Optional	any	String; a type name; data type of the argument. • any • array • binary • boolean • date • guid • numeric • query • string • struct • uuid • variableName • a component name If the value is not a recognized type, ColdFusion processes it as a component name.
required	Optional	no	Whether the parameter is required in order to execute the component method. • yes • no
default	Optional		A type; if no argument is passed, specifies a default argument value. If this attribute is present, the `required` attribute must be set to `"no"` or not specified.

Usage This tag must be positioned before any other tags within the `cffunction` tag.

Arguments that are passed when a method is invoked can be accessed from the method body in the following ways:

- With shorthand syntax: `#myargument#`
  (This example accesses the argument `myargument`.)
- Using the arguments scope as an array: `#arguments[1]#`
  (This example accesses the first defined argument in the cffunction)
- Using the arguments scope as a struct: `#arguments.myargument#`
  (This example accesses the argument `myargument` in the array)

cfassociate

Description Allows subtag data to be saved with a base tag. Applies only to custom tags.

Category Application framework tags

Syntax
```
<cfassociate
   baseTag = "base_tag_name"
   dataCollection = "collection_name">
```

See also `cfapplication`, `cferror`, `cflock`, `cfmodule`

Attributes

Attribute	Req/Opt	Default	Description
baseTag	Required		Base tag name
dataCollection	Optional	AssocAttribs	Structure in which base tag stores subtag data

Usage Call this tag within a subtag, to save subtag data in the base tag.

When ColdFusion passes subtag attributes back to the base tag, it saves them in a structure whose default name is `AssocAttribs`. To segregate subtag attributes (in a base tag that can have multiple subtags), specify a structure name, in the `DataCollection` attribute. The structure is appended to an array whose name is *thistag*.collectionName.

Within the custom tag code, the attributes passed with the `attributeCollection` attribute are saved as independent values, with no indication that they are grouped into a structure by the custom tag's caller.

If an attribute value is explicitly assigned to the attribute name, and is a member of the `attributecollection` attribute, the explicit value overrides the value in the `attributecollection` structure.

Example
```
<!--- Find the context --->
<cfif thisTag.executionMode is "start">
 <!--- Associate attributes --->
 <cfassociate baseTag = "CF_TAGBASE">

 <!--- Define defaults for attributes --->
 <cfparam name = "attributes.happy" default = "Yes">
 <cfparam name = "attributes.sad" default = "No">
...
```

cfauthenticate

Description This tag is obsolete. Use the newer security tools; see "Authentication functions" on page 328 and the Application Security chapter in *Developing ColdFusion MX Applications with CFML*.

History New in ColdFusion MX: this tag is obsolete. It does not work in ColdFusion MX and later releases.

cfbreak

Description: Used within a `cfloop` tag. Breaks out of a loop.

Category: Flow-control tags

Syntax: `<cfbreak>`

See also: cfabort, cfexecute, cfif, cflocation, cfloop, cfswitch, cfthrow, cftry

Example:

```
<!--- This shows the use of cfbreak to exit a loop when a condition is met --->
<!--- select courses; use cfloop to find a condition; then break the loop --->
<!--- check that number is numeric --->
<cfif IsDefined("form.number")>
   <cfif Not IsNumeric(form.number)>
      <cfabort>
      </cfif>
   </cfif>
<cfquery name="GetCourses" datasource="cfsnippets">
   SELECT *
   FROM Courses
   ORDER by Course_Number
</cfquery>
<p> This example uses CFLOOP to cycle through a query to find a value.
(In our example, a list of values corresponding to courses in the Snippets
datasource). When the conditions of the query are met, CFBREAK stops the loop.
<p> Please enter a Course Number, and hit the "submit" button:
<form action="index.cfm" method="POST">
      <select name="courseNum">
   <cfoutput query="GetCourses">
      <option value="#Course_Number#">#Course_Number#
      </cfoutput>
      </select>
   <input type="Submit" name="" value="Search on my Number">
</form>
<!---- if the courseNum variable is not defined, don't loop through the query --->
<cfif IsDefined ("form.courseNum") IS "True">
   <!--- loop through query until value found, then use CFBREAK to exit query--->
   <cfloop query="GetCourses">
      <cfif GetCourses.Course_Number IS form.courseNum>
         <cfoutput>
            <h4>Your Desired Course was found:</h4>
            <pre>#Course_Number# #Descript#</pre>
         </cfoutput>
         <cfbreak>
      <cfelse>
            <br> Searching...
      </cfif>
   </cfloop>
</cfif>
```

cfcache

Description

Stores a copy of a page on the server and/or client computer, to improve page rendering performance. To do this, the tag creates temporary files that contain the static HTML returned from a ColdFusion page.

Use this tag if it is not necessary to get dynamic content each time a user accesses a page.

You can use this tag for simple URLs and for URLs that contain URL parameters.

Category

Page processing tags

Syntax

```
<cfcache
   action = "action"
   directory = "directory_name"
   timespan = "value"
   expireURL = "wildcarded_URL_reference"
   username = "username"
   password = "password"
   port = "port_number"
   protocol = "protocol">
```

See also

cfflush, cfheader, cfhtmlhead, cfsetting, cfsilent

History

New in ColdFusion MX:

- The `timeout` and `cachedirectory` attributes are deprecated. Do not use them in new applications. They might not work, and might cause an error, in later releases.
- The `timespan` attribute is new.
- The default `action` attribute value, `cache`, caches a page on the server and the client. (In earlier releases, this option cached a page only on the server.) The default `protocol` and `port` values are taken from the current page URL. (In earlier releases, they were `"http"`and `"80"`, respectively.)
- This tag can cache pages that depend on session state, including pages that are secured with a ColdFusion login. (In earlier releases, the session state was cleared when caching the page, causing authentication to be lost.)
- This tag uses a hash() of the URL for the file name to cache a file. (In earlier releases, ColdFusion used the cfcache.map file.)

Attributes

Attribute	Req/Opt	Default	Description
action	Optional	cache	• cache: server-side and client-side caching. • flush: refresh cached page(s). • clientcache: browser-side caching only. To cache a personalized page, use this option. • servercache: server-side caching only. Not recommended. • optimal: same as "cache".
directory	Optional	cf_root/ cache	Absolute path of cache directory.

Attribute	Req/Opt	Default	Description
timespan	Optional	Page is flushed only when `cfcache action = "flush"` is executed	The interval until the page is flushed from the cache. • A decimal number of days. For example: - ".25", for one-fourth day (6 hours) - "1", for one day - "1.5", for one and one half days • A return value from the CreateTimeSpan function. For example: `"#CreateTimeSpan(0, 6, 0, 0)#"`
expireURL	Optional	Flush all cached pages	Used with `action = "flush"`. A URL reference. ColdFusion matches it against the mappings in the specified cache directory. Can include wildcards. For example: `"*/view.cfm?id=*"`.
username	Optional		A username. Provide this if the page requires authentication at the web server level.
password	Optional		A password. Provide this if the page requires authentication at the web server level.
port	Optional	The current page port	Port number of the web server from which the URL is requested. In the internal call from `cfcache` to `cfhttp`, ColdFusion resolves each URL variable in the page; this ensures that links in the page remain functional.
protocol	Optional	The current page protocol	Protocol that is used to create URL from cache. • http:// • https://

Usage

Use this tag in pages whose content is not updated frequently. Taking this action can greatly improve the performance of your application.

The output of a cached page is stored in a file on the client browser and/or the ColdFusion server. Instead of regenerating and redownloading the output of the page, each time it is requested, ColdFusion uses the cached output. ColdFusion regenerates and downloads the page only when the cache is flushed, as specified by the `timespan` attribute, or by invoking `cfcache action=flush`.

To enable a simple form of caching, put a `cfcache` tag, specifying the `timespan` attribute, at the top of a page. Each time the specified time span passes, ColdFusion flushes (deletes) the copy of the page from the cache and caches a new copy for users to access.

You can specify client-side caching or a combination of client-side and server-side caching (the default), using the action attribute. The advantage of client-side caching is that it requires no ColdFusion resources; the browser stores pages in its own cache, improving performance. The advantage of combination caching is that it optimizes server performance; if the browser does not have a cache of the page, the server can get data from its own cache. (Macromedia recommends that you do not use server-side only caching. Macromedia recommends that you use combination caching.)

If a page contains personalized content, use the `action = "clientcache"` option to avoid the possibility of caching a personalized copy of a page for other users.

Debug settings have no effect on `cfcache` unless the application page enables debugging. When generating a cached file, `cfcache` uses `cfsetting showDebugOutput = "No"`.

The cfcache tag evaluates each unique URL, including URL parameters, as a distinct page, for caching purposes. For example, the output of http://server/view.cfm?id=1 and the output of http://server/view.cfm?id=2 are cached separately.

The `cfcache` tag uses the `cfhttp` tag to get the contents of a page to cache. If there is an HTTP error accessing the page, the contents are not cached. If a ColdFusion error occurs, the error is cached.

Example

```
<!--- This example produces as many cached files as there are
         URL parameter permutations. --->
<cfcache
   timespan="#createTimeSpan(0,0,10,0)#">
<body>

<h3>This is a test of some simple output</h3>
<cfparam name = "URL.x" default = "no URL parm passed" >
<cfoutput>The value of URL.x = # URL.x #</cfoutput>
```

cfcase

Description Used with the cfswitch and cfdefaultcase tags.
For more information, see cfswitch on page 284.

Category Flow-control tags

cfcatch

Description Used with the cftry tag.

For more information, see cftry on page 307.

Category Exception handling tags

cfchart

Description Generates and displays a chart.

Category Data output tags, Extensibility tags

Syntax

```
<cfchart
   format = “flash, jpg, png”
   chartHeight = "integer number of pixels"
   chartWidth = "integer number of pixels"
   scaleFrom = "integer minimum value"
   scaleTo = "integer maximum value"
   showXGridlines = “yes" or "no”
   showYGridlines = “yes" or "no”
   gridlines = "integer number of lines"
   seriesPlacement = "default, cluster, stacked, percent"
   foregroundColor = "Hex value or Web color"
   dataBackgroundColor = "Hex value or Web color"
   borderBackgroundColor = “Hex value or Web color”
   showBorder = "yes" or "no”
   font = “font name”
   fontSize = “integer font size”
   fontBold = “yes" or "no”
   fontItalic = “yes" or "no”
   labelFormat = “number, currency, percent, date”
   xAxisTitle = "title text"
   yAxisTitle = "title text"
   sortXAxis = "yes/no"
   show3D = “yes" or "no”
   xOffset = "number between -1 and 1"
   yOffset = "number between -1 and 1"
   rotated = "yes/no"
   showLegend = "yes/no"
   tipStyle = "MouseDown, MouseOver, Off"
   tipBGColor = "hex value or web color"
   showMarkers = ”yes" or "no”
   markerSize = "integer number of pixels"
   pieSliceStyle = "solid, sliced"
   url = “onClick destination page"
   name = "String"
   </cfchart>
```

See also cfchartdata, cfchartseries

History New in ColdFusion MX: This tag is new.

Attributes

Attribute	Req/Opt	Default	Description
format		flash	File format in which to save graph. • flash • jpg • png
chartHeight		240	Chart height; integer number of pixels

Attribute	Req/Opt	Default	Description
chartWidth		320	Chart width; integer number of pixels
scaleFrom		Determined by data	Y-axis minimum value; integer.
scaleTo		Determined by data	Y-axis maximum value; integer
showXGridlines		no	• yes: display X-axis gridlines • no: hide X-axis gridlines
showYGridlines		yes	• yes: display Y-axis gridlines • no: hide Y-axis gridlines
gridlines		3 (top, bottom, and zero)	Number of grid lines to display on value axis, including axis; positive integer. 0: hide gridlines
seriesPlacement		default	Applies to charts that have more than one data series. Relative positions of series. • default: ColdFusion determines relative positions, based on graph types • cluster • stacked • percent
foregroundColor		black	Color of text, gridlines, and labels. Hex value or supported named color; see name list in the Usage section. For a hex value, use the form "##xxxxxx", where x = 0-9 or A-F; use two pound signs or none.
dataBackgroundColor		white	Color of area around chart data. Hex value or supported named color; see name list in the Usage section. For a hex value, use the form "##xxxxxx", where x = 0-9 or A-F; use two pound signs or none.
borderBackgroundColor		white	Color of area between data background and border, around labels and around legend. Hex value or supported named color; see name list in the Usage section. For a hex value, use the form "##xxxxxx", where x = 0-9 or A-F; use two pound signs or none.
showBorder		no	• yes • no

Attribute	Req/Opt	Default	Description
font		arial	Name of text font • arial • times • courier • arialunicodeMS. This option is required, if you are using a double-byte character set on UNIX, or using a double-byte character set on Windows with a file type of Flash.
fontSize		11	Font size; integer
fontBold		no	• yes • no
fontItalic		no	• Yes • No
labelFormat		number	Format for Y-axis labels. • number • currency • percent • date
xAxisTitle			text; X-axis title
yAxisTitle			text; Y-axis title
sortXAxis		no	• yes: display column labels in alphabetic order along X-axis • no
show3D		no	• yes: display chart with three-dimensional appearance • no
xOffset		0.1	Applies if `show3D="yes"`. Number of units by which to display the chart as angled, horizontally. • A number in the range -1 – 1, where "-1" specifies 90 degrees left and "1" specifies 90 degrees right
yOffset		0.1	Applies if `show3D="yes"`. Number of units by which to display the chart as angled, vertically. • A number in the range -1 – 1, where "-1" specifies 90 degrees down, and "1" specifies 90 degrees up
rotated		no	• yes: rotate chart 90 degrees. For a horizontal bar chart, use this option. • no
showLegend		yes	• yes: if chart contains more than one data series, display legend • no

Attribute	Req/Opt	Default	Description
tipStyle		mouseOver	Determines the action that opens a popup window to display information about the current chart element. • mouseDown: displays if the user positions the cursor at the element and clicks the mouse down. Applies only to Flash format graph file. (For other formats, this option functions the same as `mouseOver`.) • mouseOver: displays if the user positions the cursor at the element • off: suppresses display
tipbgcolor		white	Applies only to Flash format graph file. Hex value or supported named color; see name list in the Usage section. For a hex value, use the form "##xxxxxx", where x = 0-9 or A-F; use two pound signs or none.
showMarkers		yes	Applies to `chartseries type` attribute values `line`, `curve` and `scatter`. • yes: display markers at data points • no
markerSize		(Automatic)	Size of data point marker. in pixels. Integer.
pieSliceStyle		sliced	Applies to `chartseries type` attribute value `pie`. • solid: displays pie as if unsliced • sliced: displays pie as if sliced
url			URL to open if the user clicks item in a data series; the onClick destination page. You can specify variables within the URL string; ColdFusion passes current values of the variables. • `$VALUE$`: value of selected row. If none, value is empty string. • `$ITEMLABEL$`: label of selected item. If none, value is empty string. • `$SERIESLABEL$`: label of selected series. If none, value is empty string. For example: “`somepage.cfm?item=$ITEMLABEL$&series=$SERIESLABEL$&value=$VALUE$`” • "javascript:...": executes a client-side script

Attribute	Req/Opt	Default	Description
name			Page variable name. String. Outputs ISO-8859-1 encoded binary graph data and sets it as a variable in the page. Suppresses chart display. Used primarily by Flash Gateway users.

Usage

The `cfchart` tag defines a "container" in which a graph displays: its height, width, background color, labels, and so on. The `cfchartseries` tag defines the style in which data displays: bar, line, pie, and so on. The `cfchartdata` tag defines a data point.

Data is passed to the `cfchartseries` tag in the following ways:

- As a query
- As data points, using the `cfchartdata` tag

For the `font` attribute value "`ArialUnicodeMS`", the following rules apply:

- On Windows: to permit Flash charts (`type = "flash"`) to render double-byte character set, you must select this value
- On UNIX: for all `type` values, to render a double-byte character set, you must select this value.
- If this value is selected, the `fontbold andfontitalic` attributes have no effect

The color attributes accept the following W3C HTML 4 named color value or hex values:

Color name	RGB value
Black	#000000
Green	##008000
Silver	##C0C0C0
Lime	##00FF00
Gray	##808080
Olive	##808000
White	##FFFFFF
Yellow	##FFFF00
Maroon	##800000
Navy	##000080
Red	##FF0000
Blue	##0000FF
Purple	##800080
Teal	##008080
Fuchsia	##FF00FF
Aqua	##00FFFF

For all other color values, you must enter the hex value.

For more color names that are supported by popular browsers, see http://www.w3.org/TR/css3-color

Flash does not conform fully to UNIX X11 naming conventions.

You can specify whether charts are cached in memory, the number of charts to cache, and the number of chart requests that ColdFusion can process concurrently. To set these options: in the ColdFusion Administrator, select **Server Settings** > **Charting**.

cfchartdata

Description Used with the cfchart and cfchartseries tags. This tag defines chart data points. Its data is submitted to the cfchartseries tag.

Category Data output tags, Extensibility tags

Syntax
```
<cfchartdata
   item = "text"
   value = "number">
```

See also cfchart, cfchartseries

History New in ColdFusion MX: This tag is new.

Attributes

Attribute	Req/Opt	Default	Description
item			string; data point name
value			number or expression; data point value

cfchartseries

Description Used with the `cfchart` tag. This tag defines the style in which chart data displays: bar, line, pie, and so on.

Category Data output tags, Extensibility tags

Syntax
```
<cfchartseries
   type="type"
   query="queryName"
   itemColumn="queryColumn"
   valueColumn="queryColumn"
   seriesLabel="Label Text"
   seriesColor="Hex value or Web color"
   paintStyle="plain, raise, shade, light"
   markerStyle="style"
   colorlist = “list”>
</cfchartseries>
```

See also `cfchart`, `cfchartdata`

History New in ColdFusion MX: This tag is new.

Attributes

Attribute	Req/Opt	Default	Description
type	Required		Sets the chart display style: • bar • line • pyramid • area • cone • curve • cylinder • step • scatter • pie
query	Optional		Name of ColdFusion query from which to get data.
itemColumn	Required if `query` attribute is specified		Name of a column in the query specified in the `query` attribute; contains the item label for a data point to graph.
valueColumn	Required if `query` attribute is specified		Name of a column in the query specified in the `query` attribute; contains data values to graph.
seriesLabel	Optional		Text of data series label

Attribute	Req/Opt	Default	Description
seriesColor	Optional		Color of the main element (such as the bars) of a chart. For a pie chart, the color of the first slice. Hex value or supported named color; see name list in the `cfchart` Usage section. For a hex value, use the form "##xxxxxx", where x = 0-9 or A-F; use two pound signs or none.
paintStyle	Optional	plain	Sets the paint display style of the data series. • plain: solid color • raise: the appearance of a button • shade: gradient fill, darker at the edges • light: a lighter shade of color; gradient fill
markerStyle	Optional	rectangle	Applies to `chartseries type` attribute values `line`, `curve` and `scatter`, and `show3D` attribute value `no`. Sets the icon that marks a data point: • rectangle • triangle • diamond • circle • letter • mcross • snow • rcross
colorlist	Optional		Applies if `chartseries type` attribute = "`pie`". Sets pie slice colors. Comma-delimited list of hex values or supported, named web colors; see name list in the `cfchart` Usage section. For a hex value, use the form "##xxxxxx", where x = 0-9 or A-F; use two pound signs or none.

Usage

For a pie chart, ColdFusion sets pie slice colors as follows:

- If the `seriesColor` attribute is omitted, ColdFusion automatically colors the slices
- If the `seriesColor` attribute is specified, ColdFusion automatically colors the slices after the first one, starting with the specified color for the first slice
- If the `colorList` attribute is specified, ColdFusion colors the slices after the first one, according to the list

cfcol

Description: Defines table column header, width, alignment, and text. Used within a `cftable` tag.

Category: Data output tags

Syntax:

```
<cfcol
   header = "column_header_text"
   width = "number_indicating_width_of_column"
   align = "Left" or "Right" or "Center"
   text = "column_text">
```

See also: `cfcontent`, `cfoutput`, `cftable`

History: New in ColdFusion MX: You can construct dynamic `cfcol` statements.

Attributes

Attribute	Req/Opt	Default	Description
header	Required		Column header text. To use this attribute, you must also use the `cftable colHeaders` attribute.
width	Optional	20	Column width. If the length of data displayed exceeds this value, data is truncated to fit. To avoid this, use an HTML `table` tag. If the surrounding `cftable` tag includes the `htmltable` attribute, `width` specifies the percent of the table width and it does not truncate text; otherwise, `width` specifies the number of characters.
align	Optional	Left	Column alignment • Left • Right • Center
text	Required		Double-quotation mark-delimited text; determines what to display. Rules: same as for `cfoutput` sections. You can embed hyperlinks, image references, and input controls.

Usage: At least one `cfcol` tag is required within the `cftable` tag. You must put `cfcol` and `cftable` tags adjacent in a page. The only tag that you can nest within the `cftable` tag is the `cfcol` tag. You cannot nest `cftable` tags.

To display the `cfcol header` text, you must specify the `cfcol header` and the `cftable colHeader` attribute. If you specify either attribute without the other, the header does not display. No error is thrown.

Example:

```
<!--- This example shows the use of cfcol and cftable to align
         information returned from a query --->
<!--- query selects information from cfsnippets data source --->
<cfquery name = "GetEmployees" dataSource = "cfsnippets">
   SELECT Emp_ID, FirstName, LastName, EMail, Phone, Department
   FROM  Employees
</cfquery>
```

```
<html>
<body>
<h3>cfcol Example</h3>
<!--- Uses the HTMLTable attribute to display cftable as an HTML
         table, rather than PRE formatted information --->
<cftable
   query = "GetEmployees"
   startRow = "1" colSpacing = "3"
   HTMLTable colheaders>
<!--- each cfcol tag sets the width of a column in the table,
         the header information and the text/CFML for the cell --->
   <cfcol header = "<b>ID</b>"
      align = "Left"
      width = 2
      text  = "#Emp_ID#">
   <cfcol header = "<b>Name/Email</b>"
      align = "Left"
      width = 15
      text  = "<a href = 'mailto:#Email#'>#FirstName# #LastName#</A>">
   <cfcol header = "<b>Phone Number</b>"
      align = "Center"
      width = 15
      text  = "#Phone#">
</cftable>
```

cfcollection

Description

Creates, registers, and administers Verity search engine collections.

A collection that is created with the `cfcollection` tag is **internal**. A collection created any other way is **external**.

A collection that is registered with ColdFusion using the `cfcollection` tag or registered with the K2 Server by editing the k2server.ini file is **registered**. Other collections are **unregistered**.

An **internal** collection can be created in these ways:

- With the `cfcollection` tag
- In the ColdFusion Administrator, which calls the `cfcollection` tag

An **external** collection can be created using a native Verity indexing tool, such as Vspider or MKVDK.

Category

Extensibility tags

Syntax

```
<cfcollection
   action = "action"
   collection = "collection_name"
   path = "path_to_verity_collection"
   language = "language"
   name = "queryname" >
```

See also

`cfexecute`, `cfindex`, `cfobject`, `cfreport`, `cfsearch`, `cfwddx`

History

New in ColdFusion MX:

- The `action` attribute is required.
- The `action` attribute `list` value is new. It is the default.
- It is not necessary to specify the `action` attribute value `map`. (ColdFusion detects collections and creates maps collections as required.)
- This tag accepts collection names that include spaces.
- ColdFusion supports Verity operations on Acrobat PDF files.
- This tag can throw the SEARCHENGINE exception.

Attributes

Attribute	Req/Opt	Default	Description
action	Required; see Usage section	list	• create: registers the collection with ColdFusion. - If the collection is present: creates a map to it -If the collection is not present: creates it • repair: fixes data corruption in a collection. • delete: un-registers a collection. - If the collection was registered with `action = create`: deletes its directories - If the collection was registered and mapped: does not delete collection directories • map: creates a map to the collection. It is not necessary to specify this value. (ColdFusion maps collections automatically.) • optimize: optimizes the structure and contents of the collection for searching; recovers space. • list: returns a query result set, named from the `name` attribute value, of the attributes of the collections that are registered by ColdFusion and K2 Server.
collection	Required		• A collection name. The name can include spaces.
path	Optional		Absolute path to a Verity collection. To map an existing collection, specify a fully-qualified path to the collection (not including the collection name). For example, `"C:\MyCollections\"`
language	Optional	English	Options are listed in Usage section. Requires the appropriate (European or Asian) Verity Locales language pack.
name	Required if `action = "list"`		Name for the query results returned by the `list` action.

Usage

With this tag you can create, register a Verity collection and administer a collection that was created by ColdFusion or by a Verity application.

The following table shows the dependence relationships among this tag's attribute values:

Specifying this attribute is required, optional or unnecessary (blank):	**For this `action` attribute value:**					
	`list`	`create`	`map`	`optimize`	`repair`	`delete`
`collection`		Required	Required	Required	Required	Required
`path`		Optional	Optional			
`language`		Optional	Optional			
`name`	Required					

For all `action` values of this tag, use the `cflock` tag to protect the collection during tag execution.

To register a collection with K2Server, you update the k2server.ini file.

Before you attempt to delete or purge a collection that is also opened by the K2Server, you must stop the K2Server. If you do not, some files may be open, and ColdFusion might not complete the action.

The `list` action returns a result set that contains one row per collection:

Column	Contents
EXTERNAL	• Yes: the collection is external • No: the collection is not external • Not Found: the collection is registered but is not available in the defined path
LANGUAGE	The locale setting of the collection. This information is not available for K2Server collections.
MAPPED	• Yes: the collection is mapped • No: the collection is not mapped This information is not available for K2Server collections.
NAME	• For a ColdFusion registered collection: its name • For a K2Server registered collection: its alias, defined in the k2server.ini file. (ColdFusion saves registered K2Server collection information; it is available regardless of whether the K2Server is running)
ONLINE	• Yes: the collection can be searched • No: the collection cannot be searched If EXTERNAL = "Not Found", this value is "No".
PATH	Absolute path to the collection. If the collection is mapped or is registered by a K2Server, the path includes the collection name.
REGISTERED	• CF: collection is registered by ColdFusion • K2: collection is registered by K2Server

You can also display this information in the Administrator, under Verity > Collections.

If the K2 Server is not running when the `list` action is executed, the result set returned contains K2Server information that was current when the server became unavailable.

To determine whether a collection exists, use code such as the following, to execute a query of queries:

```
<cflock name="verity" timeout="60">
   <cfcollection action="list" name="myCollections" >
</cflock>
<cfquery name="qoq" dbtype="query">
      select * from myCollections
      where myCollections.name = 'myCollectionName'
      </cfquery>
<cfdump var = #qoq#>
```

To get a result set with values for all the collections that are registered with the ColdFusion and K2 servers, use code such as the following:

```
<cflock name="verity" timeout="60">
   <cfcollection action="list" name="myCollections">
</cflock>
<cfoutput query="myCollections">
   #name#<br>
</cfoutput>
```

To add content to a collection, use `cfindex`. To search a collection, use `cfsearch`.

With the European Verity Locales language pack installed, the `language` attribute of this tag supports the following options:

bokmal	french	norweg
danish	german	portug
dutch	italian	portuguese
english	nynorsk	spanish
finnish	norwegian	swedish

With the Asian Verity Locales language pack installed, the `language` attribute of this tag supports the following options:

arabic	hungarian	russian
czech	japanese	simplified_chinese
greek	korean	traditional_chinese
hebrew	polish	turkish

The default location of Verity collections is as follows:

- Windows: C:\CFusionMX\verity\collections
- Unix system: /opt/coldfusionmx/verity/collections

Example

```
<!--- for ACTION=UPDATE ------------------------------------------------>
<!--- for ACTION=UPDATE, #1 (TYPE=FILE) (key is a filename) ---->
<cfindex
     collection="snippets"
     action="update"
     type="file"
     key="c:\inetpub\wwwroot\cfdocs\snippets\abs.cfm"
     urlpath="http://localhost/cfdocs/snippets"
     custom1="custom1"
     custom2="custom2" >

<!--- for ACTION=UPDATE, #2 (TYPE=FILE) (key is a query result set column) ---->
<cfquery name="bookquery"
     datasource="book">
     select *from book where bookid='file'
</cfquery>
```

```
<cfoutput
     query="bookquery">
     --#url#,#description#-- <br>
</cfoutput>
<cfindex
     collection="snippets"
     action="update"
     type="file"
     query="bookquery"
     key="description"
     urlpath="url">

<!--- for ACTION=UPDATE, #3 (TYPE=PATH) (extensions .htm, .html,.cfm,.cfml) --->
<cfindex collection="snippets"
     action="update"
     type="path"
     key="c:\inetpub\wwwroot\cfdocs\snippets"
     urlpath="http://localhost/cfdocs/snippets"
     custom1="custom1"
     custom2="custom2"
     recurse="no"
     extensions=".htm, .html, .cfm, .cfml" >

<!--- for ACTION=UPDATE, #4 (TYPE=PATH)
     (extensions are files with no extension) ---->
<cfindex
     collection="snippets"
     action="update"
     type="path"
     key="c:\inetpub\wwwroot\cfdocs\snippets"
     urlpath="http://localhost/cfdocs/snippets"
     custom1="custom1"
     custom2="custom2"
     recurse="no"
     extensions="*." >

<!--- for ACTION=UPDATE, #5 (TYPE=PATH)
     (extensions are files with any extension) ---->
<cfindex
     collection="snippets"
     action="update"
     type="path"
     key="c:\inetpub\wwwroot\cfdocs\snippets"
     urlpath="http://localhost/cfdocs/snippets"
     custom1="custom1"
     custom2="custom2"
     recurse="no"
     extensions=".*">

<!--- for ACTION=UPDATE, #6 (TYPE=PATH) (where the key
     is a query result set column) ---->
<cfquery name="bookquery"
     datasource="book">
     select * from book where bookid='path1' or bookid='path2'
</cfquery>
```

```
<cfoutput
     query="bookquery">
     --#url#,#description#-- <br>
</cfoutput>
<cfindex
     collection="snippets"
     action="update"
     type="path"
     query="bookquery"
     key="description"
     urlpath="url" >

<!--- for ACTION=UPDATE, #7 (TYPE=CUSTOM) ---->
<cfquery name="book"
     datasource="book">
     select * from book
</cfquery>
<cfindex
     collection="custom_book"
     action="update"
     type="custom"
     body="description"
     key="bookid"
     query="book">

<!--- for ACTION=REFRESH------------------------------------------------>
<!--- ACTION=REFRESH, #1 (TYPE=FILE) ---->
<cflock name="verity"
     timeout="60">
<cfindex
     collection="snippets"
     action="Refresh"
     type="file"
     key="c:\inetpub\wwwroot\cfdocs\snippets\abs.cfm"
     urlpath="http://localhost/"
     custom1="custom1"
     custom2="custom2" >
</cflock>

<!--- ACTION=REFRESH, #2 (TYPE=PATH) ---->
<cflock name="verity"
     timeout="60">
<cfindex
     collection="snippets"
     action="refresh"
     type="path"
     key="c:\inetpub\wwwroot\cfdocs\snippets"
     urlpath="http://localhost/cfdocs/snippets/"
     custom1="custom1"
     custom2="custom2"
     recurse="yes"
     extensions=".htm,.html,.cfm,.cfml" >
</cflock>
```

```
<!--- ACTION=REFRESH, #3 (TYPE=CUSTOM) ---->
<cfquery name="book"
     datasource="book">
     select * from book
</cfquery>
<cfindex
     collection="custom_book"
     action="refresh"
     type="custom"
     body="description"
     key="bookid"
     query="book">

<!--- for ACTION=DELETE------------------------------------------------>

<!--- ACTION=DELETE, #1 (TYPE=FILE) ---->
<cflock name="verity"
     timeout="60">
<cfindex
     collection="snippets"
     action="delete"
     key="c:\inetpub\wwwroot\cfdocs\snippets\abs.cfm" >
</cflock>

<!--- ACTION=DELETE, #2 (TYPE=FILE) (the key is a query result set column) ---->
<cflock name="verity"
     timeout="60">
<cfquery name="book"
     datasource="book">
     select * from book where bookid='file'
</cfquery>
<cfoutput
     query="book">
     --#description#-- <br>
</cfoutput>
<cfindex
     collection="snippets"
     action="delete"
     type="file"
     query="book"
     key="description" >
</cflock>

<!--- ACTION=DELETE, #3 (TYPE=PATH) ---->
<cflock name="verity"
     timeout="60">
<cfindex
     collection="snippets"
     action="delete"
     type="path"
     key="c:\inetpub\wwwroot\cfdocs\snippets"
     extensions=".cfm"
     recurse="no">
</cflock>
```

```
<!--- ACTION=DELETE, #4 (TYPE=PATH) (key is a query result set column) ---->
<cflock name="verity"
     timeout="60">
<cfquery
     name="bookquery"
     datasource="book">
     select * from book where bookid='path1'
        </cfquery>
<cfoutput
     query="bookquery">
     --#url#,#description#-- <br>
</cfoutput>
<cfindex
     collection="snippets"
     action="delete"
     type="path"
     query="bookquery"
     key="description" >
</cflock>

<!--- ACTION=DELETE, #5 (TYPE=CUSTOM) ---->
<cflock name="verity"
     timeout="60">
<cfquery name="book"
     datasource="book">
     select * from book where bookid='bookid1'
</cfquery>
<cfindex
     collection="custom_book"
     action="delete"
     type="custom"
     query="book"
     key="bookid" >
</cflock>

<!--- for ACTION=PURGE------------------------------------------------>
<cflock name="verity"
     timeout="60">
<cfindex
     action="purge"
     collection="snippets">
</cflock>
```

cfcomponent

Description

Creates and defines a component object; encloses functionality that you build in CFML and enclose within `cffunction` tags. This tag contains one or more `cffunction` tags that define methods. Code within the body of this tag, other than `cffunction` tags, is executed when the component is instantiated.

A component file has the extension CFC and is stored in any directory of an application.

A component is invoked in the following ways:

- Within the `cfinvoke` tag in a ColdFusion page
- Within a URL that calls a CFC file and passes a method name as a URL parameter
- Within the `cfscript` tag
- As a web service
- From Flash code

Category

Extensibility tags

Syntax

```
<cfcomponent
   extends ="anotherComponent"
   output = "yes" or "no">
   <cffunction ...>
      ...
   </cffunction>

   <cffunction ...>
      ...
   </cffunction>
</cfcomponent>
```

See also

`cfargument`, `cffunction`, `cfinvoke`, `cfinvokeargument`, `cfobject`, `cfproperty`, `cfreturn`

Attributes

Attribute	Req/Opt	Default	Description
extends	Optional		Name of parent component from which to inherit methods and properties.
output	Optional		• yes: permits component method output • no: suppresses component method output

Usage

ColdFusion processes output expressions as follows:

- If "`output=no`", ColdFusion suppresses all output.
- If "`output=yes`", ColdFusion evaluates all `#expr#` tokens as if the function body was enclosed within `cfoutput` tags.
- If you do not specify the `output` attribute, you must enclose all tokens that are to be evaluated within `cfoutput` tags.

This tag requires an end tag.

Example

```
<cfcomponent>
   <cffunction name="getEmp">
      <cfquery
         name="empQuery" datasource="ExampleApps" >
       SELECT FIRSTNAME, LASTNAME, EMAIL
       FROM tblEmployees
      </cfquery>
      <cfreturn empQuery>
   </cffunction>

   <cffunction name="getDept">
     <cfquery
         name="deptQuery" datasource="ExampleApps" >
       SELECT *
       FROM tblDepartments
      </cfquery>
      <cfreturn deptQuery>
   </cffunction>
</cfcomponent>
```

cfcontent

Description

Downloads a file from the server to the client. Sets the file or MIME content encoding of content returned by the current page. Sets the name of a file that is downloaded by the current page.

Note: For this tag execute, it must be enabled in the ColdFusion Administrator. For more information, see *Administering ColdFusion MX*.

Category

Data output tags

Syntax

```
<cfcontent
   type = "file_type"
   deleteFile = "Yes" or "No"
   file = "filename"
   reset = "Yes" or "No">
```

See also

cfcol, cfoutput, cftable

Attributes

Attribute	Req/Opt	Default	Description
type	Optional. You must specify one of these: type, file, reset		A file or MIME content type, optionally including character encoding, in which to return the page. • text/html • (any valid type) The following character set values are typically used: • charset=UTF-8 • charset=ISO-8859-1 • charset=UTF-16 • charset=US-ASCII • charset=UTF-16BE • charset=UTF-16LE For example: type = "text/html" type = "text/html; charset = ISO-8859-1" The semi-colon is optional. For a list of character sets, see: http://www.w3.org/International/O-charset-lang.html
deleteFile	Optional	No	Applies only if you specify a file with the file attribute. • Yes: deletes a file after the download operation • No
file	Optional		Name of file to get. When using ColdFusion in a distributed configuration, the file attribute must refer to a path on the system on which the web server runs.
reset	Optional	Yes	The reset and file attributes are mutually exclusive. If you specify a file, this attribute has no effect. See Note. • Yes: discards output that precedes call to cfcontent • No: preserves output that precedes call to cfcontent

Note: If you call this tag from a custom tag, and you do not want the tag to discard the current page when it is called from another application or custom tag, set `reset` = "no".

Usage To set the character encoding of generated output, use code such as the following:

```
<cfcontent type="text/html; charset=ISO-8859-1">
```

If a file delete operation is unsuccessful, ColdFusion throws an error.

When ColdFusion processes an HTTP request, it determines the character set of the data returned in the HTTP response. By default, ColdFusion returns character data using the Unicode UTF-8 format (regardless of the value of an HTML `meta` tag in the page). Within a ColdFusion page, you can override the default character encoding of the response, using the `cfcontent` tag. To specify the MIME type and character set of the page output, use the `type` attribute, as follows:

```
<cfcontent type="text/html charset=EUC-JP">
```

The following example shows how you might display the results of a query, in an Excel spreadsheet within a ColdFusion page.

```
<!--- use cfsetting to block output of HTML that is not within cfoutput tags --->
<cfsetting enablecfoutputonly="Yes">

<!--- get Employee info --->
<cfquery name="GetEmps" datasource="CompanyInfo">
   SELECT * FROM Employee
</cfquery>

<!--- Set variables for special chars --->
<cfset TabChar = Chr(9)>
<cfset NewLine = Chr(13) & Chr(10)>

<!--- Set content type to invoke Excel --->
<cfcontent type="application/msexcel">

<!--- suggest default name for XLS file --->
<!--- use "Content-Disposition" in cfheader for Internet Explorer  --->
<cfheader name="Content-Disposition" value="filename=Employees.xls">

 <!--- output data, each row on one line--->
<cfloop query="GetEmps">
   <cfoutput>#Emp_ID##TabChar##LastName#
   #TabChar##FirstName##TabChar##Salary##NewLine#</cfoutput>
</cfloop>
```

If you use this tag after the `cfflush` tag on a page, ColdFusion throws an error.

Example

```
<!--- This example shows the use of cfcontent to return the contents of the CF
         Documentation page dynamically to the browser. You might need to change
         the path and/or drive letter. (graphics do not display) --->
<!--- Files may be set to delete after downloading, allowing
         for posting of changing content. --->
<cfcontent type = "text/html"
         file = "c:\inetpub\wwwroot\cfdocs\main.htm"
         deleteFile = "No">
```

```
<!--- This example shows how reset attribute changes text output. --->
<html>
<body>
<h3>cfcontent Example 2</h3>

<p>This example shows how reset attribute changes output for text.</p>
<p>reset = "Yes ": 123
         <cfcontent type = "text/html" reset = "Yes ">456</p>
<p>This example shows how reset attribute changes output for text.</p>
<p>reset = "No ": 123
         <cfcontent type = "text/html" reset = "No ">456</p>
```

cfcookie

Description Defines web browser cookie variables, including expiration and security options.

Category Forms tags, Variable manipulation tags

Syntax
```
<cfcookie
   name = "cookie_name"
   value = "text"
   expires = "period"
   secure = "Yes" or "No"
   path = "url"
   domain = ".domain">
```

See also `cfdump`, `cfparam`, `cfregistry`, `cfsavecontent`, `cfschedule`, `cfset`

Attributes

Attribute	Req/Opt	Default	Description
name	Required		Name of cookie variable.
value	Optional		Value to assign to cookie variable.
expires	Optional	now	Expiration of cookie variable. • A date (for example, 10/09/97) • A number of days (for example, 10, or 100) • now: deletes cookie from client cookie.txt file • never: never deletes cookie from client; writes cookie data to cookie.txt file
secure	Optional		If browser does not support Secure Sockets Layer (SSL) security, cookie is not sent. • Yes: variable must be transmitted securely • No
path	Optional		URL, within a domain to which the cookie applies. path = "/services/login" To specify multiple URLs, use multiple `cfcookie` tags. If you specify `path`, you must also specify `domain`.
domain	Required if path attribute is specified		Domain in which cookie is valid and to which cookie content can be sent. Must start with a period. If the value is a subdomain, the valid domain is: all domain names that end with this string. The cookie is set securely only if the page in which the cookie is used is referenced using the https:// protocol. For a `domain` value that ends in a country code, the specification must contain at least three periods; for example, "`mongo.state.us`". For special top-level domains, two periods are required; for example, "`.mgm.com`". You cannot use an IP address as a domain.

Usage

If this tag specifies that a cookie is to be saved beyond the current browser session, ColdFusion writes or updates the cookie to the cookies.txt file. Until the browser is closed, the cookie resides in memory. If the `expires` attribute is not specified, the cookie is written to the cookies.txt file.

If you use this tag after the `cfflush` tag on a page, ColdFusion throws an error.

To set cookies and execute a redirect in the same page, use the `cfheader` tag to specify the new target URL; for example:

```
<cfheader name="location" value="OtherPage.cfm?foo=bar">
<cfheader statusCode="302" statusText="Document Moved">
<cfabort>
```

You can use dots in names within the cookie and client variable scopes, as the following examples show:

```
<cfcookie name="person.name" value="wilson, john">
<cfset cookie.person.lastname="Santiago">
<cfcookie name="a.b.c" value="a value">
<cfset client.foo.bar="a_value">
```

Caution: Do not set a cookie variable on the same page on which you use the `cflocation` tag. If you do, the cookie is never saved on the browser.

Example

```
<!--- This example shows how to set/delete a cfcookie variable --->
<!--- Select users who have entered comments into sample database --->
<cfquery name = "GetAolUser" dataSource = "cfsnippets">
  SELECT EMail, FromUser, Subject, Posted
  FROM  Comments
</cfquery>
<html>
<body>
<h3>cfcookie Example</h3>
<!--- if the URL variable delcookie exists, set cookie expiration date to NOW --->
<cfif IsDefined("url.delcookie") is True>
  <cfcookie name = "TimeVisited"
  value = "#Now()#"
  expires = "NOW">
<cfelse>
<!--- Otherwise, loop through list of visitors; stop when you match
        the string aol.com in a visitor's e-mail address --->
<cfloop query = "GetAolUser">
  <cfif FindNoCase("aol.com", Email, 1) is not 0>
    <cfcookie name = "LastAOLVisitor"
    value = "#Email#"
    expires = "NOW" >
  </cfif>
</cfloop>
<!--- If the timeVisited cookie is not set, set a value --->
  <cfif IsDefined("Cookie.TimeVisited") is False>
    <cfcookie name = "TimeVisited"
    value = "#Now()#"
    expires = "10">
  </cfif>
</cfif>
```

```
<!--- show the most recent cookie set --->
<cfif IsDefined("Cookie.LastAOLVisitor") is "True">
   <p>The last AOL visitor to view this site was
   <cfoutput>#Cookie.LastAOLVisitor#</cfoutput>, on
   <cfoutput>#DateFormat(COOKIE.TimeVisited)#</cfoutput>
<!--- use this link to reset the cookies --->
<p><a href = "cfcookie.cfm?delcookie = yes">Hide my tracks</A>
<cfelse>
   <p>No AOL Visitors have viewed the site lately.
</cfif>
```

cfdefaultcase

Description Used with the cfswitch and cfcase tags.
For more information, see cfswitch on page 284.

Category Flow-control tags

cfdirectory

Description Manages interactions with directories.

Category File management tags

Note: For this tag execute, it must be enabled in the ColdFusion Administrator. For more information, see *Administering ColdFusion MX*.

If you put ColdFusion applications on a server that is used by multiple customers, you must consider the security of files and directories that could be uploaded or otherwise manipulated with this tag by unauthorized users. For more information about securing ColdFusion tags, see *Administering ColdFusion MX*.

Syntax

```
<cfdirectory
   action = "directory action"
   directory = "directory name"
   name = "query name"
   filter = "list filter"
   mode = "permission"
   sort = "sort specification"
   newDirectory = "new directory name">
```

See also `cffile`

History New in ColdFusion MX: On Windows, if `cfdirectory action = "list"`, this tag does not return the directory entries " . " (dot) or " . . " (dot dot), which represent "the current directory" and "the parent directory." (In earlier releases, ColdFusion returned these entries.)

New in ColdFusion MX:

- Code such as the following, which was acceptable in earlier releases, may cause incorrect output in ColdFusion MX:

```
<cfdirectory action = "list" directory ="c:\" name="foo>
Files in c:\<br>
<cfloop query = "foo" startrow = 3>
   #name#<br>
>
```

- Code such as the following, which was acceptable in earlier releases, is acceptable in ColdFusion MX, although it is unnecessary:

```
<cfdirectory directory="c:\" name="foo">
Files in c:\<br>
<cfoutput query="foo">
   <cfif NOT foo.name is "." AND NOT foo.name is "..">
      #name#<br>
   </cfif>
</cfoutput>
```

Attributes

Attribute	Req/Opt	Default	Description
action	Optional	List	• list: returns a query record set of the files in the specified directory. The directory entries "." (dot) and ".." (dot dot), which represent the current directory and the parent directory, are not returned. • create • delete • rename
directory	Required		Absolute pathname of directory against which to perform action.
name	Required if `action = "list"`		Name for output record set.
filter	`Optional if action = "list"`		File extension filter applied to returned names. For example: *.cfm. One filter can be applied.
mode	Optional		Used with `action = "create"`. Permissions. Applies only to Solaris and HP-UX. Octal values of chmod command. Assigned to owner, group, and other, respectively. For example: • 644: Assigns read/write permission to owner; read permission to group and other • 777: Assigns read/write/execute permission to all
sort	Optional; used if `action = "list"`	ASC	Query column(s) by which to sort directory listing. Delimited list of columns from query output. To qualify a column, use: • ac: ascending (a to z) sort order • desc: descending (z to a) sort order For example: sort = "dirname ASC, file2 DESC, size, datelastmodified"
newDirectory	Required if `action = "rename"`		New name for directory

Usage

If `action = "list"`, `cfdirectory` returns these result columns, which you can reference in a `cfoutput` tag:

- `name`: directory entry name. The entries `". "` and `". . "` are not returned.
- `size`: directory entry size
- `type`: file type: File, for a file; Dir, for a directory
- `dateLastModified`: the date that an entry was last modified
- `attributes`: file attributes, if applicable
- `mode`: (UNIX and Linux only) see the UNIX `man` pages, `chmod` shell command.

You can use the following result columns in standard CFML expressions, preceding the result column name with the query name:

```
#mydirectory.name#
#mydirectory.size#
#mydirectory.type#
#mydirectory.dateLastModified#
#mydirectory.attributes#
#mydirectory.mode#
```

Example

```
<h3>cfdirectory Example</h3>
<!--- use cfdirectory to give the contents of the snippets directory,
         order by name and size (you may need to modify this path) --->
<cfdirectory
   directory="#GetDirectoryFromPath(GetTemplatePath())#"
   name="myDirectory"
   sort="name ASC, size DESC">
<!---- Output the contents of the cfdirectory as a cftable ----->
<cftable
   query="myDirectory"
   htmltable
   colheaders>
   <cfcol
      header="NAME:"
      text="#Name#">
   <cfcol
      header="SIZE:"
      text="#Size#">
</cftable>
```

cfdump

Description

Outputs the elements, variables and values of most kinds of ColdFusion objects. Useful for debugging. You can display the contents of simple and complex variables, objects, components, user-defined functions, and other elements.

Category

Debugging tags, Variable manipulation tags

Syntax

```
<cfdump
   var = #variable#
   expand = "Yes or No"
   label = "text">
```

See also

cfcookie, cfparam, cfsavecontent, cfschedule, cfset, cfwddx

Attributes

Attribute	Req/Opt	Default	Description
var	Required		Variable to display. Enclose a variable name in pound signs. These kinds of variables yield meaningful `cfdump` displays: • array • CFC • Java object • simple • query • structure • UDF • wddx • xml
expand	Optional	Yes	• Yes: In Internet Explorer and Mozilla, expands views • No: contracts expanded views
label	Optional		A string; header for the dump output.

Usage

The expand/contract display capability is useful when working with large structures, such as XML document objects, structures, and arrays.

To display a construct, use code such as the following, in which *myDoc* is a variable of type XmlDocument:

```
<cfif IsXmlDoc(mydoc) is "True">
   <cfdump var=#mydoc#>
</cfif>
```

The tag output is color-coded according to data type.

If a table cell is empty, this tag displays "[empty string]".

Example

This example shows how to use this tag to display a URL variable. URL variables contain parameters that are passed in a URL string in a page request.

```
<cfdump var="#URL#">
```

cfelse

Description Used with the cfif and cfelseif tags.
For more information, see cfif on page 154.

Category Flow-control tags

cfelseif

Description Used with the cfif and cfelse tags.
For more information, see cfif on page 154.

Category Flow-control tags

cferror

Description: Displays a custom HTML page when an error occurs. This lets you maintain a consistent look and feel among an application's functional and error pages.

Category: Exception handling tags, Extensibility tags, Application framework tags

Syntax:

```
<cferror
   type = "a type"
   template = "template_path"
   mailTo = "email_address"
   exception = "exception_type">
```

See also: cfrethrow, cfthrow, cftry

History: New in ColdFusion MX: the monitor option of the exception attribute is deprecated. Do not use it in new applications. It might not work, and might cause an error, in later releases.

Attributes

Attribute	Req/Opt	Default	Description
type	Required		Type of error that the custom error page handles: • application: application exceptions • database: database exceptions • template: ColdFusion page exceptions • security: security exceptions • object: object exceptions • missinginclude: missing include file exceptions • expression: expression exceptions • lock: lock exceptions • custom_type: developer-defined exceptions, defined in the cfthrow tag • any: all exception types
template	Required		Relative path to the custom error page. (A ColdFusion page was formerly called a template.)
mailTo	Optional		E-mail address. Value for the error page variable error.mailto. Available to a custom error page; for example: #error.mailTo#.
exception			Type of exception that the tag generates: • an exception of the cftry tag; see cftry • a custom exception, coded in a cfthrow tag

Usage: Use this tag to provide custom error messages for pages in an application. You generally embed this tag in the Application.cfm file. For more information, see *Administering ColdFusion MX*.

In exception error handling pages, you can access the error variables of the cfcatch tag; see cftry on page 307. To do this, prefix these variables with "cferror."

To ensure that error pages display successfully, avoid using the `cfencode` tag to encode pages that include the `cferror` tag.

Templates (pages)

The following table describes the page to use for each type of error. (A ColdFusion page was formerly commonly called a template.)

Page type	Description	Use
Exception or Error	Dynamically invoked by the CFML language processor when it detects an unhandled exception condition. You can specify exception-handling pages in an application, using `cferror type = "exception"`. In the ColdFusion Administrator, you can set a site-wide error handler, to handle exceptions that are not handled by an exception-handling page.	Uses the full range of CFML tags
Request	Includes the error variables described in the Error Variables section.	As a backup error handler for sites with high user interface requirements.
Validation	Handles data input validation errors that occur when submitting a form. Handles only hidden form-field style validation errors. You can specify the validation page in a `cferror` tag, directly on the action page.	You must include the validation error handler in the Application.cfm file.

Error variables

The exception-handling page specified in the `cferror` tag `template` attribute, contains one or more error variables. ColdFusion substitutes the value of the error variable when an error displays.

The following table lists error variables:

Page type	Error variable	Description
Exception	error.diagnostics	Detailed error diagnostics from ColdFusion Server.
	error.mailTo	E-mail address (same as value in `cferror.MailTo`).
	error.dateTime	Date and time when error occurred.
	error.browser	Browser that was running when the error occurred.
	error.generatedContent	Failed request's generated content.
	error.remoteAddress	IP address of remote client.
	error.HTTPReferer	Page from which client accessed link to page where error occurred.
	error.template	Page executing when error occurred.
	error.queryString	URL query string of client's request.

Page type	Error variable	Description
Validation	error.validationHeader	Validation message header text.
	error.invalidFields	Unordered list of validation errors.
	error.validationFooter	Validation message footer text.
Request and Exception	error.diagnostics	Detailed error diagnostics from ColdFusion Server.
	error.mailTo	E-mail address (same as value in `cferror.MailTo`).
	error.dateTime	Date and time when error occurred.
	error.browser	Browser that was running when error occurred.
	error.remoteAddress	IP address of remote client.
	error.HTTPReferer	Page from which client accessed link to page where error occurred.
	error.template	Page executing when error occurred.
	error.queryString	URL query string of client's request.
Exception only	error.messge	Error message associated with the exception.
	error.rootCause	Java servelet exception reported by the JVM as the cause of the "root cause" of the exception. This variable is a Java object.
	error.tagContext	Array of structures containing information for each tag in the tag stack. The tag stack consists of each tag that is currently open.
	error.type	Exception type.

Note: If `type = "exception"` or `"monitor"`, you can substitute the prefix `cferror` for `Error`; for example, `cferror.diagnostics`, `cferror.mailTo`, or `cferror.dateTime`.

Example

```
<h3>cferror Example</h3>
<p>cferror lets you display custom HTML pages when errors occur. This lets you
         maintain a consistent look and feel within the application even when
         errors occur. No CFML can be displayed in the pages, except specialized
         error variables.
<p>cftry/cfcatch is a more interactive way to handle CF errors within a CF page
         than cferror, but cferror is a good safeguard against general errors.
<p>You can also use cferror within Application.cfm to specify error
         handling responsibilities for an entire application.
<!--- Example of cferror call within a page --->
<cferror type = "REQUEST"
    template = "request_err.cfm"
    mailTo = "admin@mywebsite.com">
<!--- Example of the page to handle this error --->
<!---
<html>
<head>
  <title>We're sorry -- An Error Occurred</title>
</head>
```

```
<body>
<ul>
<cfoutput>
   <li><b>Your Location:</b> #error.remoteAddress#
   <li><b>Your Browser:</b> #error.browser#
   <li><b>Date and Time the Error Occurred:</b> #error.dateTime#
   <li><b>Page You Came From:</b> #error.HTTPReferer#
   <li><b>Message Content</b>: <br><HR width = 50%>
    <p>#error.diagnostics#<HR width = 50%><p>
   <li><b>Please send questions to:</b>
    <a href = "mailto:#error.mailTo#">#error.mailTo#</A>
</cfoutput>
</ul>
```

cfexecute

Description Executes a ColdFusion developer-specified process on a server computer.

Category Extensibility tags, Flow-control tags

Syntax

```
<cfexecute
   name = " ApplicationName "
   arguments = "CommandLine Arguments"
   outputFile = "Output file name"
   timeout = "Timeout interval">
   ...
</cfexecute>
```

See also cfcollection, cfindex, cfobject, cfreport, cfsearch, cfwddx

Attributes

Attribute	Req/Opt	Default	Description
name	Required		Absolute path of the application to execute. On Windows, you must specify an extension; for example, C:\myapp.exe.
arguments	Optional		Command-line variables passed to application. If specified as string, it is processed as follows: • Windows: passed to process control subsystem for parsing. • UNIX: tokenized into an array of arguments. The default token separator is a space; you can delimit arguments that have embedded spaces with double quotation marks. If passed as array, it is processed as follows: • Windows: elements are concatenated into a string of tokens, separated by spaces. Passed to process control subsystem for parsing. • UNIX: elements are copied into an array of exec() arguments.
outputFile	Optional		File to which to direct program output. If not specified, output is displayed on page from which it was called.
timeout	Optional	0	Length of time, in seconds, that ColdFusion waits for output from the spawned program. • 0: equivalent to non-blocking mode. • A very high value: equivalent to blocking mode If the value is 0: • ColdFusion starts a process and returns immediately. ColdFusion may return control to the calling page before any program output displays. To ensure that program output displays, set the value to 2 or higher. • If the `outputFile` attribute is not specified, any program output is discarded

Usage Do not put other ColdFusion tags or functions between the start and end tags of `cfexecute`. You cannot nest `cfexecute` tags.

Exception Throws the following exceptions:

- If the application name is not found: Application File Not Found
- If the output file cannot be opened: Output File Cannot
- If the effective user of the ColdFusion executing thread does not have permissions to execute the process: a security exception

The time out values must be between zero and the longest time out value supported by the operating system.

Example

```
<h3>cfexecute</h3>
<p>This example executes the Windows NT version of the netstat network
         monitoring program, and places its output in a file.

<cfexecute name = "C:\WinNT\System32\netstat.exe"
  arguments = "-e"
  outputFile = "C:\Temp\output.txt"
  timeout = "1">
</cfexecute>
```

cfexit

Description

This tag aborts processing of the currently executing CFML custom tag, exits the page within the currently executing CFML custom tag, or re-executes a section of code within the currently executing CFML custom tag.

Category

Debugging tags, Flow-control tags

Syntax

```
<cfexit
   method = "method">
```

See also

cfabort, cfbreak, cfexecute, cfif, cflocation, cfloop, cfswitch, cfthrow, cftry

Attributes

Attribute	Req/Opt	Default	Description
method	Optional	exitTag	• exittag: aborts processing of currently executing tag • exittemplate: exits page of currently executing tag • loop: reexecutes body of currently executing tag

Usage

If this tag is encountered outside the context of a custom tag, for example in the base page or an included page, it executes in the same way as `cfabort`. The `cfexit` tag can help simplify error checking and validation logic in custom tags.

The `cfexit` tag function depends on its location and execution mode:

Method value	Location of cfexit call	Behavior
exitTag	Base page	Terminate processing
	Execution mode = Start	Continue after end tag
	Execution mode = End	Continue after end tag
exitTemplate	Base page	Terminate processing
	Execution mode = Start	Continue from first child in body
	Execution mode = End	Continue after end tag
loop	Base page	Error
	Execution mode = Start	Error
	Execution mode = End	Continue from first child in body

Example

```
<h3>cfexit Example</h3>
<p>cfexit can be used to abort the processing of the currently executing
         CFML custom tag. Execution resumes following the invocation of
         the custom tag in the page that called the tag.
<h3>Usage of cfexit</h3>
<p>cfexit is used primarily to perform a conditional stop of processing
         inside a custom tag. cfexit returns control to the page that
         called that custom tag, or in the case of a tag called by another
         tag, to the calling tag.
```

```
<!--- cfexit can be used within a CFML custom tag, as follows: --->
<!--- Place this code (uncomment the appropriate sections) within the
         CFUSION/customtags directory --->

<!--- MyCustomTag.cfm --->
<!--- This simple custom tag checks for the existence
of myValue1 and myValue2. If they are both defined,
the tag adds them and returns the result to the calling
page in the variable "result". If either or both of the
expected attribute variables is not present, an error message
is generated, and cfexit returns control to the
calling page. --->

<!--- <cfif NOT IsDefined("attributes.myValue2")>
        <cfset caller.result = "Value2 is not defined">
        <cfexit method = "exitTag">
    <cfelseif NOT IsDefined("attributes.myValue1")>
        <cfset caller.result = "Value1 is not defined">
        <cfexit method = "exitTag">
    <cfelse>
        <cfset value1 = attributes.myValue1>
        <cfset value2 = attributes.myValue2>
        <cfset caller.result = value1 + value2>
    </cfif> --->
<!--- End MyCustomTag.cfm --->

<!--- Place this code within your page --->

<!--- <p>The call to the custom tag, and then the result:
<CF_myCustomTag
     myvalue2 = 4>
<cfoutput>#result#</cfoutput> --->
<p>If cfexit is used outside a custom tag, it functions like a cfabort.
         For example, the text after this message is not processed:
<cfexit>
<p>This text is not executed because of the cfexit tag above it.
```

cffile

Description

Manages interactions with files on the ColdFusion server.

The following sections describe the actions of the `cffile` tag:

- cffile action = "upload" on page 89
- cffile action = "move" on page 92
- cffile action = "rename" on page 94
- cffile action = "copy" on page 95
- cffile action = "delete" on page 97
- cffile action = "read" on page 98
- cffile action = "readBinary" on page 100
- cffile action = "write" on page 101
- cffile action = "append" on page 103

Note: To execute, this tag must be enabled in the ColdFusion Administrator. For more information, see *Administering ColdFusion MX*.

If your ColdFusion applications run on a server used by multiple customers, consider the security of the files that could be uploaded or manipulated by `cffile`. For more information, see *Administering ColdFusion MX*.

Category

File management tags

Syntax

The tag syntax depends on the `action` attribute value. See the following sections.

See also

`cfdirectory`

History

New in ColdFusion MX: ColdFusion does not require that you put files and directories that you manipulate with this tag below the root of the web server document directory.

New in ColdFusion MX: A directory path that you specify in the `destination` attribute does not require a trailing slash.

New in ColdFusion MX: on Windows platforms, the `temporary`, `archive`, and `system` values of the `attributes` attribute are deprecated. Do not use them in new applications. They might not work, and might cause an error, in later releases.

New in ColdFusion MX: The `action` attribute options `read`, `write`, `append` and `move` support a new attribute, `charset`.

Example

```
<!--- This shows how to write, read, update, and delete a file using CFFILE --->
This is a view-only example.
<!---
<cfif IsDefined("form.formsubmit") is "Yes">
   <!--- form has been submitted, now do the action --->
   <cfif form.action is "new">
      <!--- make a new file --->
      <cffile action="Write"
         file="#GetTempDirectory()#foobar.txt"
         output="#form.the_text#">
   </cfif>
```

```
   <cfif form.action is "read">
      <!--- read existing file --->
      <cffile action="Read"
           file="#GetTempDirectory()#foobar.txt"
           variable="readText">
   </cfif>

   <cfif form.action is "add">
      <!--- update existing file --->
      <cffile action="Append"
         file="#GetTempDirectory()#foobar.txt"
         output="#form.the_text#">
   </cfif>

   <cfif form.action is "delete">
      <!--- delete existing file --->
      <cffile action="Delete"
         file="#GetTempDirectory()#foobar.txt">
   </cfif>
</cfif>
<!--- set some variables --->
<cfparam name="fileExists" default="no">
<cfparam name="readText" default="">
<!--- first, check if canned file exists --->
<cfif FileExists("#GetTempDirectory()#foobar.txt") is "Yes">
   <cfset fileExists="yes">
</cfif>
<!--- now, make the form that runs the example --->
<form action="index.cfm" method="POST">
<h4>Type in some text to include in your file:</h4> <p>
<cfif fileExists is "yes">
   A file exists (foobar.txt, in <cfoutput>#GetTempDirectory()#</cfoutput>).
   You may add to it, read from it, or delete it.
</cfif> <
!--- if reading from a form, let that information display in textarea --->
<textarea name="the_text" cols="40" rows="5">
   <cfif readText is not "">
      <cfoutput>#readText#</cfoutput>
   </cfif></textarea>
<!--- select from the actions depending on whether the file exists --->
<select name="action">
<cfif fileExists is "no">
   <option value="new">Make new file
</cfif>
<cfif fileExists is "yes">
   <option value="add">Add to existing file
   <option value="delete">Delete file
   <option value="read">Read existing file
</cfif>
</select>
<input type="Hidden" name="formsubmit" value="yes">
<input type="Submit" name="" value="make my changes">
</form> --->
```

cffile action = "upload"

Description: Copies a file to a directory on the server.

Syntax:

```
<cffile
   action = "upload"
   fileField = "formfield"
   destination = "full_path_name"
   nameConflict = "behavior"
   accept = "mime_type/file_type"
   mode = "permission"
   attributes = "file_attribute_or_list">
```

See also: cfdirectory

History: New in ColdFusion MX: A directory path that you specify in the `destination` attribute does not require a trailing slash.

New in ColdFusion MX: on Windows platforms, the `temporary`, `archive`, and `system` options of the `attributes` attribute are deprecated. Do not use them in new applications. They might not work, and might cause an error, in later releases.

Attributes

Attribute	Req/Opt	Default	Description
action	Required		Type of file manipulation that the tag performs.
fileField	Required		Name of form field used to select the file. Do not use pound signs (#) to specify the field name.
destination	Required		Absolute pathname of directory or file on web server. ColdFusion 5 and earlier: trailing slash in directory path is required. ColdFusion MX: trailing slash in directory path is optional. On Windows, use backward slashes; on UNIX, use forward slashes.
nameConflict	Optional	Error	Action to take if filename is the same as that of a file in the directory. • Error: file is not saved. ColdFusion stops processing the page and returns an error. • Skip: file is not saved. This option permits custom behavior based on file properties. • Overwrite: replaces file. • MakeUnique: forms a unique filename for the upload; name is stored in the file object variable `serverFile`.
accept	Optional		Limits the MIME types to accept. Comma-delimited list. For example, to permit JPG and Microsoft Word file uploads: `accept = "image/jpg, application/msword"` The browser uses file extension to determine file type.

Attribute	Req/Opt	Default	Description
mode	Optional		Applies only to Solaris and HP-UX. Permissions. Octal values of chmod command. Assigned to owner, group, and other, respectively. For example: • 644: Assigns read/write permission to owner; read permission to group and other • 777: Assigns read/write/execute permission to all
attributes	Optional		One attribute (Windows) or a comma-delimited list of attributes (other platforms) to set on the file. If omitted, the file's attributes are maintained. Each value must be specified explicitly. For example, if you specify `attributes = "readOnly"`, all other attributes are overwritten. • readOnly • hidden • normal (if you use this option with other attributes, it is overridden by them)

Usage

After a file upload is completed, you can get status information using file upload parameters. The status parameters use the `cffile` prefix; for example, `cffile.clientDirectory`. Status parameters can be used anywhere other ColdFusion parameters can be used.

Note: The `file` prefix is deprecated, in favor of the `cffile` prefix. Do not use the `file` prefix in new applications.

The following file upload status parameters are available after an upload.

Parameter	Description
attemptedServerFile	Initial name ColdFusion used when attempting to save a file
clientDirectory	Directory location of the file uploaded from the client's system
clientFile	Name of the file uploaded from the client's system
clientFileExt	Extension of the uploaded file on the client system (without a period)
clientFileName	Name of the uploaded file on the client system (without an extension)
contentSubType	MIME content subtype of the saved file
contentType	MIME content type of the saved file
dateLastAccessed	Date and time the uploaded file was last accessed
fileExisted	Whether the file already existed with the same path (Yes or No)
fileSize	Size of the uploaded file
fileWasAppended	Whether ColdFusion appended uploaded file to a file (Yes or No)
fileWasOverwritten	Whether ColdFusion overwrote a file (Yes or No)
fileWasRenamed	Whether uploaded file renamed to avoid a name conflict (Yes or No)
fileWasSaved	Whether Cold Fusion saves a file (Yes or No)

Parameter	Description
oldFileSize	Size of a file that was overwritten in the file upload operation
serverDirectory	Directory of the file saved on the server
serverFile	Filename of the file saved on the server
serverFileExt	Extension of the uploaded file on the server (without a period)
serverFileName	Name of the uploaded file on the server (without an extension)
timeCreated	Time the uploaded file was created
timeLastModified	Date and time of the last modification to the uploaded file

Tip: To refer to parameters, use the `cffile` prefix: for example, `#cffile.fileExisted#`.

Note: File status parameters are read-only. They are set to the results of the most recent `cffile` operation. (If two `cffile` tags execute, the results of the second overwrite the first.)

Example

The following example creates a unique filename, if there is a name conflict when the file is uploaded on Windows:

```
<cffile action = "upload"
   fileField = "FileContents"
   destination = "c:\web\uploads\"
   accept = "text/html"
   nameConflict = "MakeUnique">
```

The following examples show the use of the `mode` attribute. The first example creates the file /tmp/foo with permissions defined as: owner=read/write, group=read, other=read.

```
<cffile action = "write"
         file = "/tmp/foo"
         mode = 644
         output = "some text">
```

This example appends to a file and sets permissions to read/write (rw) for all.

```
<cffile action = "append"
         destination = "/home/tomj/testing.txt"
         mode = 666
         output = "Is this a test?">
```

This example uploads a file and sets permissions to owner/group/other = read/write/execute.

```
<cffile action = "upload"
         fileField = "fieldname"
         destination = "/tmp/program.exe"
         mode = 777>
```

cffile action = "move"

Description Moves a file from one location to another on the server.

Syntax
```
<cffile
   action = "move"
   source = "full_path_name"
   destination = "full_path_name"
   mode = "mode"
   attributes = "file_attributes_list"
   charset = "charset_option">
```

See also `cfdirectory`

History New in ColdFusion MX: this `action` attribute option supports the `charset` attribute.

New in ColdFusion MX: A directory path that you specify in the `destination` attribute does not require a trailing slash.

New in ColdFusion MX: on Windows platforms, the `temporary`, `archive`, and `system` options of the `attributes` attribute are deprecated. Do not use them in new applications. They might not work, and might cause an error, in later releases.

Attributes

Attribute	Req/Opt	Default	Description
action	Required		Type of file manipulation that the tag performs.
source	Required		Absolute pathname of file on web server. On Windows, use backward slashes; on UNIX, use forward slashes.
destination	Required		Absolute pathname of directory or file on web server. ColdFusion 5 and earlier: trailing slash in directory path is required. ColdFusion MX: trailing slash in directory path is optional. On Windows, use backward slashes; on UNIX, use forward slashes.
mode	Optional		Applies only to Solaris and HP-UX. Permissions. Octal values of UNIX chmod command. Assigned to owner, group, and other, respectively. For example: • 644: Assigns read/write permission to owner; read permission to group and other • 777: Assigns read/write/execute permission to all
attributes	Optional		One attribute (Windows) or a comma-delimited list of attributes (other platforms) to set on the file. If omitted, the file's attributes are maintained. Each value must be specified explicitly. For example, if you specify `attributes = "readOnly"`, all other attributes are overwritten. • readOnly • hidden • normal

Attribute	Req/Opt	Default	Description
charset	Optional	UTF-8	A Java character set name for the file contents. The following values are typically used: • UTF-8 • ISO-8859-1 • UTF-16 • US-ASCII • UTF-16BE • UTF-16LE For a list of character sets, see: http://www.w3.org/International/O-charset-lang.html

Example

The following example moves the keymemo.doc file from the c:\files\upload\ directory to the c:\files\memo\ directory on Windows:

```
<cffile
   action = "move"
   source = "c:\files\upload\keymemo.doc"
   destination = "c:\files\memo\">
```

In this example, the destination directory is "memo."

cffile action = "rename"

Description Renames a file on the server.

Syntax
```
<cffile
   action = "rename"
   source = "full_path_name"
   destination = "full_path_name"
   mode = "mode"
   attributes = "file_attributes_list">
```

See also cfdirectory

History New in ColdFusion MX: a directory path that you specify in the destination attribute does not require a trailing slash.

New in ColdFusion MX: on Windows platforms, the temporary, archive, and system options of the attributes attribute are deprecated. Do not use them in new applications. They might not work, and might cause an error, in later releases.

Attributes

Attribute	Req/Opt	Default	Description
action	Required		Type of file manipulation that the tag performs.
source	Required		Absolute pathname of file on web server. On Windows, use backward slashes; on UNIX, use forward slashes.
destination	Required		Absolute pathname of file on web server. On Windows, use backward slashes; on UNIX, use forward slashes.
mode	Optional		Applies only to Solaris and HP-UX. Permissions. Octal values of UNIX chmod command. Assigned to owner, group, and other. For example: • 644: Assigns read/write permission to owner; read permission to group and other • 777: Assigns read/write/execute permission to all
attributes	Optional		One attribute (Windows) or a comma-delimited list of attributes (other platforms) to set on the file. If omitted, the file's attributes are maintained. Each value must be specified explicitly. For example, if attributes = "readOnly", all other attributes are overwritten. • readOnly • hidden • normal

Example The following example renames the file keymemo.doc to oldmemo.doc:

```
<cffile action = "rename"
   source = "c:\files\memo\keymemo.doc"
          destination = "c:\files\memo\oldmemo.doc">
```

cffile action = "copy"

Description Copies a file from one directory to another on the server.

Syntax
```
<cffile
   action = "copy"
   source = "full_path_name"
   destination = "full_path_name"
   mode = "mode"
   attributes = "file_attributes_list">
```

See also cfdirectory

History New in ColdFusion MX: a directory path that you specify in the `destination` attribute does not require a trailing slash.

New in ColdFusion MX: on Windows platforms, the `temporary`, `archive`, and `system` options of the `attributes` attribute are deprecated. Do not use them in new applications. They might not work, and might cause an error, in later releases.

Attributes

Attribute	Req/Opt	Default	Description
action	Required		Type of file manipulation that the tag performs.
source	Required		Absolute pathname of file on web server. On Windows, use backward slashes; on UNIX, use forward slashes.
destination	Required		Absolute pathname of directory or file on web server. ColdFusion 5 and earlier: trailing slash in directory path is required. ColdFusion MX: trailing slash in directory path is optional. On Windows, use backward slashes; on UNIX, use forward slashes.
mode	Optional		Applies only to Solaris and HP-UX. Permissions. Octal values of Unix chmod command. Assigned to owner, group, and other, respectively. For example: • 644: Assigns read/write permission to owner; read permission to group and other • 777: Assigns read/write/execute permission to all
attributes	Optional		One attribute (Windows) or a comma-delimited list of attributes (other platforms) to set on the file. If omitted, the file's attributes are maintained. Each value must be specified explicitly. For example, if you specify `attributes = "readOnly"`, all other attributes are overwritten. • readOnly • hidden • normal

Example This example copies the keymemo.doc file to the c:\files\backup\ directory:

```
<cffile action = "copy"
          source = "c:\files\upload\keymemo.doc"
          destination = "c:\files\backup\">
```

cffile action = "delete"

Description Deletes a file on the server.

Syntax
```
<cffile
   action = "delete"
   file = "full_path_name">
```

See also cfdirectory

Attributes

Attribute	Req/Opt	Default	Description
action	Required		Type of file manipulation that the tag performs.
file	Required		Absolute pathname of file on web server. On Windows, use backward slashes; on UNIX, use forward slashes.

Example The following example deletes the specified file:

```
<cffile action = "delete"
   file = "c:\files\upload\#Variables.DeleteFileName#">
```

cffile action = "read"

Description

Reads a text file on the server. The file is read into a dynamic, local parameter that you can use in the page. For example:

- Read a text file; insert the file's contents into a database
- Read a text file; use the find and replace function to modify the file's contents

Note: This action reads the file into a variable in the local Variables scope. It is not intended for use with large files, such as logs, because this can bring down the server.

Syntax

```
<cffile
   action = "read"
   file = "full_path_name"
   variable = "var_name"
   charset = "charset_option" >
```

See also

cfdirectory

New in ColdFusion MX: this `action` attribute option supports the `charset` attribute.

Attributes

Attribute	Req/Opt	Default	Description
action	Required		Type of file manipulation that the tag performs.
file	Required		Absolute pathname of file. On Windows, use backward slashes; on UNIX, use forward slashes.
variable	Required		Name of variable to contain contents of text file.
charset	Optional	UTF-8	The Java character set name used for the file contents. The following values are typically used: • UTF-8 • ISO-8859-1 • UTF-16 • US-ASCII • UTF-16BE • UTF-16LE For a list of character sets, see: http://www.w3.org/International/O-charset-lang.html

Usage

The following example creates a variable named Message for the contents of the file message.txt:

```
<cffile action = "read"
      file = "c:\web\message.txt"
      variable = "Message">
```

The variable `Message` can be used in the page. For example, you could display the contents of the message.txt file in the final web page as follows:

```
<cfoutput>#Message#</cfoutput>
```

ColdFusion supports functions for manipulating the contents of text files. You can also use the variable that is created by a `cffile action = "read"` operation in the ArrayToList and ListToArray functions.

cffile action = "readBinary"

Description

Reads a binary file (such as an executable or image file) on the serve, into a binary object parameter that you can use in the page. To send it through a web protocol (such as HTTP or SMTP) or store it in a database, first convert it to Base64 using the ToBase64 function.

Note: This action reads the file into a variable in the local Variables scope. It is not intended for use with large files, such as logs, because they can bring down the server.

Syntax

```
<cffile
   action = "readBinary"
   file = "full_path_name"
   variable = "var_name">
```

See also

cfdirectory

Attributes

Attribute	Req/Opt	Default	Description
action	Required		Type of file manipulation that the tag performs.
file	Required		Absolute pathname of file. On Windows, use backward slashes; on UNIX, use forward slashes.
variable	Required		Name of variable to contain contents of binary file.

Usage

You convert the binary file to Base64 to transfer it to another site.

Example

The following example creates a variable named `aBinaryObj` to contain the ColdFusion Server executable:

```
<cffile action = "readBinary"
         file = "c:\cfusion\bin\cfserver.exe"
         variable = "aBinaryObj">
```

cffile action = "write"

Description

Writes a text file on the server, based on dynamic content. You can create static HTML files from the content, or log actions in a text file.

Syntax

```
<cffile
   action = "write"
   file = "full_path_name"
   output = "content"
   mode = "permission"
   addNewLine = "Yes" or "No"
   attributes = "file_attributes_list"
   charset = "charset_option" >
```

See also

cfdirectory

History

New in ColdFusion MX: on Windows platforms, the `attributes` attribute values `temporary`, `archive`, and `system` are deprecated. Do not use them in new applications. They might not work, and might cause an error, in later releases.

New in ColdFusion MX: this `action` attribute option supports the `charset` attribute.

Attributes

Attribute	Req/Opt	Default	Description
action	Required		Type of file manipulation that the tag performs.
file	Required		Absolute pathname of file on web server. On Windows, use backward slashes; on UNIX, use forward slashes.
output	Required		Content of the file to be created.
mode	Optional		Applies only to Solaris and HP-UX. Permissions. Octal values of Unix chmod command. Assigned to owner, group, and other, respectively. For example: • 644: Assigns read/write permission to owner; read permission to group and other • 777: Assigns read/write/execute permission to all
addNewLine	Optional	Yes	• Yes: appends newline character to text written to file • No
attributes	Optional		One attribute (Windows) or a comma-delimited list of attributes (other platforms) to set on the file. If omitted, the file's attributes are maintained. Each value must be specified explicitly. For example, if you specify `attributes = "readOnly"`, all other attributes are overwritten. • readOnly • hidden • normal

Attribute	Req/Opt	Default	Description
charset	Optional	UTF-8	The Java character set name used for the file contents. The following values are typically used: • UTF-8 • ISO-8859-1 • UTF-16 • US-ASCII • UTF-16BE • UTF-16LE For a list of character sets, see: http://www.w3.org/International/O-charset-lang.html

Example This example creates a file with information a user entered in an HTML insert form:

```
<cffile action = "write"
   file = "c:\files\updates\#Form.UpdateTitle#.txt"
          output = "Created By: #Form.FullName#
          Date: #Form.Date#
          #Form.Content#">
```

If the user submitted a form with the following:

```
UpdateTitle = "FieldWork"
FullName = "World B. Frueh"
          Date = "10/30/01"
          Content = "We had a wonderful time in Cambridgeport."
```

ColdFusion would create a file named FieldWork.txt in the c:\files\updates\ directory and the file would contain the following text:

```
Created By: World B. Frueh
Date: 10/30/01
          We had a wonderful time in Cambridgeport.
```

This example shows the use of the `mode` attribute for UNIX. It creates the file /tmp/foo with permissions rw-r--r-- (owner = read/write, group = read, other = read):

```
<cffile action = "write"
   file = "/tmp/foo"
          mode = 644>
```

This example appends to the file and sets permissions to read/write (rw) for all:

```
<cffile action = "append"
   destination = "/home/tomj/testing.txt"
          mode = 666
          output = "Is this a test?">
```

This example uploads a file and gives it the permissions owner/group/other = read/write/execute):

```
cffile action = "upload"
   fileField = "fieldname"
          destination = "/tmp/program.exe"
          mode = 777>
```

cffile action = "append"

Description Appends text to a text file on the server.

Syntax
```
<cffile
   action = "append"
   file = "full_path_name"
   output = "string"
   addNewLine = "Yes" or "No"
   attributes = "file_attributes_list">
   mode = "mode"
   charset = "charset_option" >
```

See also cfdirectory

History New in ColdFusion MX: on Windows platforms, the `temporary`, `archive`, and `system` options of the `attributes` attribute are deprecated. Do not use them in new applications. They might not work, and might cause an error, in later releases.

New in ColdFusion MX: this `action` attribute option supports the `charset` attribute.

Attributes

Attribute	Req/Opt	Default	Description
action	Required		Type of file manipulation that the tag performs.
file	Required		Absolute pathname of file on web server to which to append content of `output` attribute. On Windows, use backward slashes; on UNIX, use forward slashes.
mode	Optional		Applies only to Solaris and HP-UX. Permissions. Octal values of Unix chmod command. Assigned to owner, group, and other, respectively. For example: • 644: Assigns read/write permission to owner; read permission to group and other • 777: Assigns read/write/execute permission to all
output	Required		String to append to the file.
addNewLine	Optional	Yes	• Yes: appends newline character to text written to file • No
attributes	Optional		One attribute (Windows) or a comma-delimited list of attributes (other platforms) to set on the file. If omitted, the file's attributes are maintained. Each value must be specified explicitly. For example, if you specify `attributes = "readOnly"`, all other attributes are overwritten. • readOnly • hidden • normal

Attribute	Req/Opt	Default	Description
charset	Optional	UTF-8	The Java character set name used for the file contents. The following values are typically used: • UTF-8 • ISO-8859-1 • UTF-16 • US-ASCII • UTF-16BE • UTF-16LE For a list of character sets, see: http://www.w3.org/International/O-charset-lang.html

Example This example appends a text string to the file fieldwork.txt:

```
<cffile action = "append"
         file = "c:\files\updates\fieldwork.txt"
         output = "<b>But Davis Square is the place to be.</b>">
```

cfflush

Description Flushes currently available data to the client.

Category Data output tags, Page processing tags

Syntax
```
<cfflush
   interval = "integer number of bytes">
```

See also cfcache, cfheader, cfinclude, cfsetting, cfsilent

Attributes

Attribute	Req/Opt	Default	Description
interval	Optional		Integer. Flushes output each time this number of bytes becomes available. HTML headers, and data that is already available when the tag is executed, are omitted from the count.

Usage The first occurrence of this tag on a page sends back the HTML headers and any other available HTML. Subsequent `cfflush` tags on the page send only the output that was generated after the previous flush.

When you flush data, ensure that enough information is available, as some browsers might not respond if you flush only a small amount. Similarly, set the `interval` attribute for a few hundred bytes or more, but not thousands of bytes.

Use the `interval` attribute only when a large amount of output will be sent to the client, such as in a `cfloop` or a `cfoutput` of a large query. Using this form globally (such as in the Application.cfm file) might cause unexpected errors when CFML tags that modify HTML headers are executed.

Caution: Once you have used this tag on a page, any CFML function or tag on the page that modifies the HTML header causes an error. (These include: `cfcontent`, `cfcookie`, `cfform`, `cfheader`, `cfhtmlhead`, and `cflocation`.)

Using the `cfset` tag to set a cookie scope variable causes an error. Cookie errors can be caught with the `cfcatch type = "Any" tag`. Other errors can be caught with `cfcatch type = "template"`.

Using the `cfflush` tag within a `cfsavecontent` tag (if the `cfsavecontent` tag has content) causes an error.

Note: Normally, the `cferror` tag discards the current output buffer and replaces it with the contents of the error page. The `cfflush` tag discards the current buffer. As a result, the `Error.GeneratedContent` variable resulting from a `cferror` tag after a `cfflush` contains any contents of the output buffer that has not been flushed. This content is not sent to the client. The content of the error page displays to the client after the bytes that have been sent.

The following example uses `cfloop` tags and the `rand` random number generating function to delay data display. It simulates a page that is slow to generate data.

Example

```
<h1>Your Magic numbers</h1>
<p>It will take us a little while to calculate your ten magic numbers.
         It takes a lot of work to find numbers that truly fit your
         personality. So relax for a minute or so while we do the hard
         work for you.</p>
<H2>We are sure you will agree it was worth the short wait!</H2>
<cfflush>

<cfflush interval=10>
<!--- Delay Loop to make it seem harder --->
<cfloop index="randomindex" from="1" to="200000" step="1">
  <cfset random=rand()>
</cfloop>

<!--- Now slowly output 10 random numbers --->
<cfloop index="Myindex" from="1" to="10" step="1">
  <cfloop index="randomindex" from="1" to="100000" step="1">
    <cfset random=rand()>
  </cfloop>
  <cfoutput>
    Magic number number #Myindex# is:  #RandRange(
        100000, 999999)#<br><br>
  </cfoutput>
</cfloop>
```

cfform

Description Builds a form with CFML custom control tags; these provide more functionality than standard HTML form input elements.

Category Forms tags

Syntax
```
<cfform
   name = "name"
   action = "form_action"
   preserveData = "Yes" or "No"
   onSubmit = "javascript"
   target = "window_name"
   encType = "type"
   passThrough = "HTML_attribute(s)"
   codeBase = "URL"
   archive = "URL"
   scriptSrc = "path">
...
</cfform>
```

See also `cfapplet`, `cfgrid`, `cfinput`, `cfselect`, `cfslider`, `cftextinput`, `cftree`, `cftreeitem`

History New in ColdFusion MX: the `enableCAB` attribute is deprecated. Do not use it in new applications. It might not work, and might cause an error, in later releases.

New in ColdFusion MX: the `name` and `action` attributes are optional.

Attributes

Attribute	Req/Opt	Default	Description
name	Optional	CFForm_1 [, ...]	A name for the form.
action	Optional		Name of ColdFusion page to execute when the form is submitted for processing.
scriptSrc	Optional	/cfide/ scripts/ cfform.js	Lets the user control the URL of the script file; useful if you do not keep the file in the /cfide directory.

Attribute	Req/Opt	Default	Description
preserveData	Optional	No	When the cfform action attribute posts back to the same page as the form, this determines whether to override the control values with the submitted values. • false: values specified in the control tag attributes are used • true: corresponding submitted values are used Applies to these controls: • `cfform` controls `cfinput`, `cfslider`, `cftextinput`; overrides `value` attribute value • `cfselect` controls that are populated from queries. Overrides the `selected` attribute. See cfselect on page 267. • `cftree` controls: Overrides the `cftreeitem expand` attribute. If `true`, expands previously-selected elements. The `cftree completePath` attribute must be set to `Yes`. • `cfgrid` controls: has no effect. (This avoids confusion as to whether data has been resubmitted to the database by the control.)
onSubmit	Optional		JavaScript function to execute after input validation. Use to for preprocessing data before form is submitted. See Developing ColdFusion MX Applications with CFML.
passThrough	Optional		For HTML attributes that are not supported by `cfform`. Attributes and values are passed to the HTML code that is generated for the tag. For example: `"style=""font-weight:bold;"""`
codeBase	Optional	See Description	URL of downloadable JRE plug-in (for Internet Explorer only). Default: /CFIDE/classes/cf-j2re-win.cab
archive	Optional	See Description	URL of downloadable Java classes for ColdFusion controls. Default: /CFIDE/classes/CFJava2.jar

Usage

This tag requires an end tag.

Some custom control tags that you can use within this tag require the client to download a Java applet; they might execute slightly more slowly than using an HTML form element to get the same information. In addition to regular HTML form elements, you can use the following custom control tags within the `cfform` tag:

- `cfinput`: Creates and validates an input element (radio button, text box, checkbox)
- `cfselect`: Creates a drop-down list box
- `cfslider`: Creates a slider control (Java support required)
- `cftextinput`: Creates a text input box (Java support required)
- `cftree`: Creates a tree control (Java support required)

- `cfgrid`: Creates a grid control to display tabular data (Java support required)
- `cfapplet`: Embeds a registered Java applet (Java support required)

All of these control tags require that the browser is JavaScript-enabled.

If you use this tag after the `cfflush` tag on a page, an error is thrown.

The `method` attribute is automatically set to `post`; if you specify a value, it is ignored.

If you specify a value in quotation marks, you must escape them by doubling them; for example: `passThrough = "readonly = ""Yes"" "`.

Any form field name, from the `cfform` tag or an HTML form, that ends in one of the following suffixes invokes server-side form validation:

- **_integer**: Verifies that the user entered a number.
- **_float**: Verifies that the user entered a number.
- **_range**: Verifies that a numeric value entered is within specified boundaries.
- **_date**: Verifies that the user entered a date; converts to ODBC date format.
- **_time**: Verifies that the user correctly entered a time; converts to ODBC time format.
- **_eurodate**: Verifies that the user entered a date in a standard European date format; converts to ODBC date format.

Do not use these suffixes for your field names.

For more information, see the Retrieving and Formatting Data chapter in *Developing ColdFusion MX Applications with CFML*.

Incorporating HTML form tags

The `cfform` tag lets you incorporate these standard HTML elements:

- Standard form tag attributes and values. The attributes and values are included in the `form` tag that `cfform` outputs in the page. For example, you can use `form` tag attributes like `target` with `cfform`. Other pass-through attributes include CLASS, ENCTYPE, ID, ONLOAD, ONRESET, and STYLE.
- HTML tags that can ordinarily be put within the HTML `form` tag. For example, you can use the HTML `input` tag to create a submit button in a `cfform`, without the other features of `cfinput`:

```
<cfform>
   <input type = "Submit" value = " update... ">
</cfform>
```

Example

```
<h3>cfform Example</h3>
<cfif IsDefined("form.oncethrough") is "Yes">
  <cfif IsDefined("form.testVal1") is True>
  <h3>Results of Radio Button Test</h3>
  <cfif form.testVal1 is "Yes">Your radio button answer was yes</cfif>
  <cfif form.testVal1 is "No">Your radio button answer was no</cfif>
  </cfif>
  <cfif IsDefined("form.chkTest2") is True>
  <h3>Results of Checkbox Test</h3>
    Your checkbox answer was yes
  <cfelse>
    <h3>Results of Checkbox Test</h3>
    Your checkbox answer was no
  </cfif>
```

```
   <cfif IsDefined("form.textSample") is True
    AND form.textSample is not "">
   <h3>Results of Credit Card Input</h3>
      Your credit card number, <cfoutput>#form.textSample#</cfoutput>,
          was valid under the MOD 10 algorithm.
   </cfif>
   <cfif IsDefined("form.sampleSlider") is "True">
   <h3>You gave this page a rating of <cfoutput>#form.sampleSlider#
   </cfoutput></h3>
   </cfif>
   <hr noshade>
</cfif>
<!--- begin by calling the cfform tag --->
<cfform action = "cfform.cfm">
<table>
<tr>
   <td>
   <h4>This example displays radio button input type for cfinput.</h4>
   Yes <cfinput type = "Radio" name = "TestVal1" value = "Yes" checked>
   No <cfinput type = "Radio" name = "TestVal1" value = "No">
   </td>
</tr>
<tr>
   <td>
   <h4>This example displays checkbox input type for cfinput.</h4>
   <cfinput type = "Checkbox" name = "ChkTest2" value = "Yes">
   </td>
</tr>
<tr>
   <td>
   <h4>This shows client-side validation for cfinput text boxes.</h4>
   <br>(<I>This item is optional</i>)<br>
   Please enter a credit card number:
   <cfinput type = "Text" name = "TextSample"
      message = "Please enter a Credit Card Number"
      validate = "creditcard" required = "No">
   </td>
</tr>
<tr>
   <td>
   <h4>This example shows the use of the cfslider tag.</h4>
   <p>Rate your approval of this example from 1 to 10 by sliding control.
   <p>1 <cfslider name = "sampleSlider"
        label = "Sample Slider" range = "1,10"
        message = "Please enter a value from 1 to 10"
        scale = "1" bold = "No"
        italic = "No" refreshlabel = "No"> 10
   </td>
</tr>
</table>
<p><input type = "submit" name = "submit" value = "show me the result">
<input type = "hidden" name = "oncethrough" value = "Yes">
</cfform>
</body>
</html>
```

cfftp

Description: Lets users implement File Transfer Protocol (FTP) operations.

Category: File management tags, Internet Protocol tags

Syntax: The tag syntax depends on the `action` attribute value. See the following sections:

- cfftp: Connecting to an FTP server on page 112
- cfftp: Connection caching on page 114
- cfftp: Connection: File and directory operations on page 115
- cfftp action = "listDir" on page 118

See also: `cfhttp`, `cfldap`, `cfmail`, `cfpop`

Usage: Use this tag to move files between a ColdFusion server and an FTP server.

This tag does not move files between a ColdFusion server and a client browser. You do this as follows:

- To transfer files from a client to a ColdFusion server: `cffile action = "upload"`
- To transfer files from a ColdFusion server to a client: the `cfcontent` tag

Security settings

ColdFusion Server Basic security settings can prevent the `cfftp` tag from executing. If you run ColdFusion applications on a server that is used by multiple customers, consider the security of the files that the customer can move. For more information, see *Administering ColdFusion MX*.

cfftp: Connecting to an FTP server

Description To establish a connection with an FTP server, you use the `connection` attribute.

Syntax

```
<cfftp
   action = "action"
   username = "name"
   password = "password"
   server = "server"
   timeout = "timeout in seconds"
   port = "port"
   connection = "name"
   proxyServer = "proxyserver"
   retryCount = "number"
   stopOnError = "Yes" or "No"
   passive = "Yes" or "No">
```

See also `cfhttp`, `cfldap`, `cfmail`, `cfpop`

Attributes

Attribute	Req/Opt	Default	Description
action	Required		FTP operation to perform. • open: create an FTP connection • close: terminate an FTP connection For more information, see "cfftp: Connection caching" on page 114.
username	Required if `action = "open"`		User name to pass in the FTP operation.
password	Required if `action = "open"`		Password to log in the user.
server	Required if `action = "open"`		FTP server to which to connect; for example, `ftp.myserver.com`
timeout	Optional	30	Value in seconds for the timeout of all operations, including individual data request operations.
port	Optional	21	Remote port to which to connect.
connection	Optional		Name of the FTP connection. Used to cache a new FTP connection or to reuse a connection. If you specify the `username`, `password`, and `server` attributes, and if no connection exists for them, ColdFusion creates one. Calls to `cfftp` with the same connection name reuse the connection information.
proxyServer	Optional		String. Name of proxy server (or servers) to use, if proxy access is specified.
retryCount	Optional	1	Number of retries until failure is reported.

Attribute	Req/Opt	Default	Description
stopOnError	Optional	No	• Yes: halts processing, displays an appropriate error. • No: populates these variables: cfftp.succeeded – Yes or No. cfftp.errorCode – Error number. See the IETF Network Working Group RFC 959: File Transfer Protocol (FTP): www.ietf.org/rfc/rfc0959.txt. cfftp.errorText – Message text For conditional operations, use `cfftp.errorCode`. Do not use `cfftp.errorText` for this purpose.
passive	Optional	No	• Yes: enable passive mode • No

Usage

If you use connection caching to an active FTP connection, you do not have to respecify the `username`, `password`, or `server` connection attributes:

Changing a cached connection, such as changing `retryCount` or `timeout` values, might require reestablishing the connection.

Example

```
<p>cfftp lets users implement File Transfer Protocol operations.
         By default, cfftp caches an open connection to an FTP server.
<p>cfftp operations are usually of two types:
<ul>
   <li>Establishing a connection
   <li>Performing file and directory operations
</ul>
<p>This example opens and verifies a connection, lists the files in a
         directory, and closes the connection.
<p>Open a connection
<cfftp action = "open"
   username = "anonymous"
   connection = "My_query"
   password = "youremail@email.com"
   server = "ftp.tucows.com"
   stopOnError = "Yes">
<p>Did it succeed? <cfoutput>#cfftp.succeeded#</cfoutput>
<p>List the files in a directory:
<cfftp action = "LISTDIR"
    stopOnError = "Yes"
    name = "ListFiles"
    directory = "/"
    connection = "my_query">
<cfoutput query = "ListFiles">
   #name#<br>
</cfoutput>

<p>Close the connection:
<cfftp action = "close"
connection = "My_query"
stopOnError = "Yes">
<p>Did it succeed? <cfoutput>#cfftp.succeeded#</cfoutput>
```

cfftp: Connection caching

Description

After you establish a connection with `cfftp`, you can reuse it to perform additional FTP operations. To do this, you use the `connection` attribute to define and name an FTP connection object that stores information about the connection. Any FTP operations that use the same `connection` name automatically use the information stored in the connection object. This helps save connection time and improves file transfer performance.

Usage

To keep the connection open throughout a session or longer, you can use a session or application variable as the connection name. However, if you do this, you must specify the full variable name, with the `close` action, when you are finished. Keeping a connection open prevents others from using the FTP server; so close a connection as soon as possible.

Changes to a cached connection, such as changing `retryCount` or `timeout` values, might require reestablishing the connection.

Example

The following example opens a connection and gets a file listing showing file or directory name, path, URL, length, and modification date.

```
<p>Open a connection
<cfftp connection = "myConnection"
   username = "myUserName"
  password = "myUserName@allaire.com"
  server = "ftp.allaire.com"
   action = "open"
   stopOnError = "Yes">

<p>Did it succeed? <cfoutput>#cfftp.succeeded#</cfoutput>
<cfftp connection = "myConnection"
    action = "LISTDIR"
    stopOnError = "Yes"
    name = "ListDirs"
    directory = "/">

<p>FTP Directory Listing:<br>
<cftable query = "ListDirs" HTMLTable = "Yes" colHeaders = "Yes">
   <cfcol header = "<b>Name</b>" text = "#name#">
   <cfcol header = "<b>Path</b>" text = "#path#">
   <cfcol header = "<b>URL</b>" text = "#url#">
   <cfcol header = "<b>Length</b>" text = "#length#">
   <cfcol header = "<b>LastModified</b>"
    text = "Date(Format#lastmodified#)">
   <cfcol header = "<b>IsDirectory</b>" text = "#isdirectory#">
</cftable>

<p>Close the connection:
<cfftp connection = "myConnection"
    action = "close"
    stopOnError = "Yes">
<p>Did it succeed? <cfoutput>#cfftp.succeeded#</cfoutput>
```

cfftp: Connection: File and directory operations

Description Use this form of the `cfftp` tag to perform file and directory operations with `cfftp`.

Syntax

```
<cfftp
   action = "action"
   username = "name"
   password = "password"
   name = "query_name"
   server = "server"
   ASCIIExtensionList = "extensions"
   transferMode = "mode"
   failIfExists = "Yes" or "No"
   directory = "directory name"
   localFile = "filename"
   remoteFile = "filename"
   item = "directory or file"
   existing = "file or directory name"
   new = "file or directory name"
   proxyServer = "proxyserver"
   passive = "Yes" or "No">
```

See also `cfhttp`, `cfldap`, `cfmail`, `cfpop`

Attributes

Attribute	Req/Opt	Default	Description
action	Required if connection is not cached		FTP operation to perform. • changedir • createDir • listDir • removeDir • getFile • putFile • rename • remove • getCurrentDir • getCurrentURL • existsDir • existsFile • exists
username	Required if connection is not cached		User name to pass in the FTP operation.
password	Required if `action = "open"`		Password to log in the user.
name	Required if `action = "listDir"`		Query name of directory listing.

Attribute	Req/Opt	Default	Description
server	Required if FTP connection is not cached		FTP server to which to connect; for example, `ftp.myserver.com`.
ASCIIExtension List	Optional	txt;htm;html; cfm;cfml; shtm;shtml; css;asp;asa	Delimited list of file extensions that force ASCII transfer mode, if `transferMode = "auto"`.
transferMode	Optional	Auto	• ASCII FTP transfer mode • Binary FTP transfer mode • Auto FTP transfer mode
failIfExists	Optional	Yes	• Yes: if a local file with same name exists, getFile fails • No
directory	Required if `action = "changedir", "createDir", "listDir", or "existsDir"`		Directory on which to perform an operation.
localFile	Required if `action = "getFile" or putFile`		Name of the file on the local file system.
remoteFile	Required if `action = "getFile", "putFile",` or `"existsFile"`		Name of the file on the FTP server file system.
item	Required if `action = "exists"` or `"remove"`		Object of these actions: file or directory.
existing	Required if `action = "rename"`		Current name of the file or directory on the remote server.
new	Required if `action = "rename"`		New name of file or directory on the remote server
proxyServer	Optional		String. Name of the proxy server (s) to use, if proxy access is specified
passive	Optional	No	• Yes: enable passive mode • No

Usage If you use connection caching to an active FTP connection, you do not have to respecify the `username`, `password`, or `server` connection attributes:

Changing a cached connection, such as changing `retryCount` or `timeout` values, might require reestablishing the connection.

If `action = "listDir"`, the `attributes` column returns `directory` or `normal`. Other platform-specific values, such as `hidden` and `system`, are no longer supported.

If `action = "listDir"`, a `mode` column is returned. The column contains an octal string representation of UNIX permissions; for example, "`777`."

The `cfftp.returnValue` variable provides the return value for these actions:

- getCurrentDir
- getCurrentURL
- existsDir
- existsFile
- exists

For more information, see *Developing ColdFusion MX Applications with CFML*.

Caution: Object (file and directory) names are case-sensitive.

Action (cfftp.ReturnValue variable)

The results of an action determine the value of the `cfftp.returnValue` variable.

cfftp action	Value of cfftp.returnValue
getCurrentDir	String. Current directory.
getCurrentURL	String. Current URL.
existsDir	Yes or No.
existsFile	Yes or No.
exists	Yes or No.

cfftp action = "listDir"

Description To access the columns in a query object, use this tag with `action = "listDir"`.

Usage When you use this action, you must specify a value for the `name` attribute. This value holds the results of the `listDir` action in a query object. The query object consists of columns that you can reference, in the form `queryname.columname[row]`, where `queryname` is the name of the query, specified in the `name` attribute; and `columnname` is a column returned in the query object. The value *row* is the row number of each file/directory entry returned by the `listDir` operation. A separate row is created for each entry:

cfftp query object column	Description
Name	Filename of the current element.
Path	File path (without drive designation) of the current element.
URL	Complete URL for the current element (file or directory).
Length	File size of the current element.
LastModified	Unformatted date/time value of the current element.
Attributes	String. Attributes of the current element: normal or Directory.
IsDirectory	Boolean. Whether object is a file or directory.
Mode	Applies only to Solaris and HP-UX. Permissions. Octal string.

Note: Previously supported query column values that pertain to system-specific information are not supported; for example, `hidden` and `system`.

cffunction

Description Defines functionality that you build in CFML.

Category Extensibility tags

Syntax

```
<cffunction
   name = "methodName"
   returnType = "dataType"
   roles = "securityRoles"
   access = "methodAccess"
   output = "yes" or "no" >
```

See also `cfargument`, `cfcomponent`, `cfinvoke`, `cfinvokeargument`, `cfobject`, `cfproperty`, `cfreturn`

Attributes

Attribute	Req/Opt	Default	Description
name	Required		A string; a component method that is used within the `cfcomponent` tag.
returnType	Required for a web service; Optional, otherwise.	any	String; a type name; data type of the function return value. • any • array • binary • boolean • date • guid • numeric • query • string • struct • uuid • variableName • void (this option does not return a value) • a return type If the value is not a recognized type, ColdFusion processes it as a component name.
roles	Optional	"" (empty)	This attribute is used only for a component. A comma-delimited list of ColdFusion security roles that can invoke the method. If this attribute is omitted, all roles can invoke the method.

Attribute	Req/Opt	Default	Description
access	Optional	public	This attribute is used only for a component. The client security context from which the method can be invoked: • private: available only to the component that declares the method • package: available only to the component that declares the method or to another component in the package • public: available to a locally executing page or component method • remote: available to a locally or remotely executing page or component method, or a remote client through a URL, Flash, or a web service. To publish the function as a web service, this option is required.
output	Optional	Function is processed as standard CFML	This attribute is used only for a component. • yes: the function is processed as if it were within a `cfoutput` tag • no: the function is processed as if it were within a `cfsilent` tag

Usage

Components that are stored in the same directory are members of a component package.

Within a `cffunction` tag, if you specify the `roles` attribute, the method executes only if a user is logged in and belongs to one of the specified roles.

For more information, see the "Building and Using ColdFusion Components" chapter in *Developing ColdFusion MX Applications with CFML*.

The following example shows a typical use of this tag:

```
<cffunction
   name="getEmployees"
   access="remote"
   returnType="query"
   hint="This query returns all records in the employee database. It can
         drill-down or narrow the search, based on optional input parameters.">
```

Example

```
<cfcomponent>
   <cffunction name="getEmp">
      <cfquery
          name="empQuery" datasource="ExampleApps" >
          SELECT FIRSTNAME, LASTNAME, EMAIL
          FROM tblEmployees
      </cfquery>
      <cfreturn empQuery>
   </cffunction>
   <cffunction name="getDept">
     <cfquery
         name="deptQuery" datasource="ExampleApps" >
       SELECT *
       FROM tblDepartments
      </cfquery>
```

```
        <cfreturn deptQuery>
    </cffunction>
</cfcomponent>
```

cfgraph

Description This tag is deprecated. Use the cfchart, cfchartdata, and cfchartseries tags instead.

Displays data graphically.

History New in ColdFusion MX: this tag is deprecated. Do not use it in new applications. It might not work, and might cause an error, in later releases.

The incompatibilities between the ColdFusion MX implementation and earlier implementations of this tag are as follows:

cfgraph tag attribute	ColdFusion MX functionality
Title	Ignored
Titlefont	Ignored
Barspacing	Ignored
Bordercolor	Color used for border, gridlines, and text displays
Colorlist	• Pie chart: list of colors to use for each data point • Other chart types: ignored
Valuelabelfont	Sets value label text font. If the `Valuelabelfont`, `Itemlabelfont`, and `Legendfont` values differ, ColdFusion uses the last value that you specify in the tag. `Arial` is not supported; it is mapped to `Dialog`.
Itemlabelfont	Sets item label text font. If the `Valuelabelfont`, `Itemlabelfont`, and `Legendfont` values differ, ColdFusion uses the last value that you specify in the tag `Arial` is not supported; it is mapped to `Dialog`.
Legendfont	Sets legend text font. If the `Valuelabelfont`, `Itemlabelfont`, and `Legendfont` values differ, ColdFusion uses the last value that you specify in the tag `Arial` is not supported; it is mapped to `Dialog`.
ShowLegend	• above, below, left, right: these options cause the legend to display, but have no effect on its location. • none: prevents display of a legend
Valuelabelsize	Sets value label text size. If the `Valuelabelsize` and `Itemlabelsize` values differ, ColdFusion uses the last value that you specify in the tag
Itemlabelsize	Sets item label text size
Itemlabelorientation	Ignored. ColdFusion calculates best orientation based on label and graph size.
Borderwidth	• a non-zero number: default-width border, regardless of number value • 0: no border
Depth	• 0: displays graph with two-dimensional appearance • any other value: displays graph with threedimensional appearance

cfgraph tag attribute	ColdFusion MX functionality
Linewidth	Ignored
Showvaluelabel	• yes: displays values on mouse-click; • no: suppresses value displays • rollover: displays values on mouse-over.
Valuelocation	Ignored
url	URL of page to open if any item in the graph is clicked. The following variables may be used within the URL; they are substituted with real values before the URL is accessed: • "$value$": selected row/column value or an empty string • "$itemlabel$": selected item (column) value or an empty string • "$serieslabel$": selected series (row) value or an empty string • "javascript:...": executes client side scripts.
Urlcolumn	Ignored. This attribute generates a graph, but the url link is missing data that the linked page may expect. Thus, drilling-down in the graph may result in an error.
Type="HorizontalBar"	The (0,0) coordinate is located at the lower-left.
ScaleFrom	If the smallest value in the data is less than scaleFrom or the largest value in the data is greater than scaleTo, the respective data value is used as the minimum or maximum on the Y scale. Therefore, regardless of the scaleFrom or scaleTo value, all data values display.

cfgraphdata

Description This tag is deprecated. Use the cfchart, cfchartdata, and cfchartseries tags instead.

Displays a data point in a graph. Used within the `cfgraph` tag.

History New in ColdFusion MX: this tag is deprecated. Do not use it in new applications. It might not work, and might cause an error, in later releases.

cfgrid

Description Used within the `cfform` tag. Puts a grid control (a table of data) in a ColdFusion form. To specify grid columns and row data, use the `cfgridcolumn` and `cfgridrow` tags, or use the `query` attribute, with or without `cfgridcolumn` tags.

Category Forms tags

Syntax

```
<cfgrid
  name = "name"
  height = "integer"
  width = "integer"
  autoWidth = "Yes" or "No"
  vSpace = "integer"
  hSpace = "integer"
  align = "value"
  query = "query_name"
  insert = "Yes" or "No"
  delete = "Yes" or "No"
  sort = "Yes" or "No"
  font = "column_font"
  fontSize = "size"
  italic = "Yes" or "No"
  bold = "Yes" or "No"
  textColor = "web color"
  href = "URL"
  hrefKey = "column_name"
  target = "URL_target"
  appendKey = "Yes" or "No"
  highlightHref = "Yes" or "No"
  onValidate = "javascript_function"
  onError = "text"
  gridDataAlign = "position"
  gridLines = "Yes" or "No"
  rowHeight = "pixels"
  rowHeaders = "Yes" or "No"
  rowHeaderAlign = "position"
  rowHeaderFont = "font_name"
  rowHeaderFontSize = "size"
  rowHeaderItalic = "Yes" or "No"
  rowHeaderBold = "Yes" or "No"
  rowHeaderTextColor = "web color"
  colHeaders = "Yes" or "No"
  colHeaderAlign = "position"
  colHeaderFont = "font_name"
  colHeaderFontSize = "size"
  colHeaderItalic = "Yes" or "No"
  colHeaderBold = "Yes" or "No"
  colHeaderTextColor = "web color"
  bgColor = "web color"
  selectColor = "web color"
  selectMode = "mode"
  maxRows = "number"
  notSupported = "text"
  pictureBar = "Yes" or "No"
```

```
   insertButton = "text"
   deleteButton = "text"
   sortAscendingButton = "text"
   sortDescendingButton = "text">
</cfgrid>
```

See also `cfapplet`, `cfform`, `cfinput`, `cfselect`, `cfslider`, `cftextinput`, `cftree`, `cftreeitem`

History New in ColdFusion MX: ColdFusion does not use the `rowHeaderWidth` attribute. You can omit it.

Attributes

Attribute	Req/Opt	Default	Description
name	Required		Name of grid element.
height	Optional	300	Height of grid control, in pixels.
width	Optional	300	Width of grid control, in pixels.
autoWidth	Optional; see description	No	• Yes: sets column widths so that all columns display within grid width. • No: sets columns to equal widths. User can resize columns. Horizontal scroll bars are not available, because if you specify a column width and set `autoWidth = "Yes"`, ColdFusion sets to this width, if possible.
vSpace	Optional		Vertical space above and below grid control, in pixels.
hSpace	Optional		Horizontal space to left and right of grid control, in pixels.
align	Optional		Alignment of the grid cell contents: • Top • Left • Bottom • Baseline • Texttop • Absbottom • Middle • Absmiddle • Right
query	Optional		Name of query associated with grid control.
insert	Optional	No	• Yes: user can insert row data in grid. Takes effect only if `selectmode="edit"` • No
delete	Optional	No	• Yes: user can delete row data from grid. Takes effect only if `selectmode="edit"` • No

Attribute	Req/Opt	Default	Description
sort	Optional	No	The sort button performs simple text sort on column. User can sort columns by clicking column head or by clicking sort buttons. Not valid with `selectmode=browse`. • Yes: sort buttons display on grid control • No
font	Optional		Font of column data in the grid control.
fontSize	Optional		Size of text in the grid control, in points.
italic	Optional	No	• Yes: displays grid control text in italic • No
bold	Optional	No	• Yes: displays grid control text in bold • No
textColor	Optional	Black	Color of text in grid control; hex or text. Hex value or supported named color; see name list in the Usage section. For a hex value, use the form "##xxxxxx", where x = 0-9 or A-F; use two pound signs or none. For a list of the supported named colors, see cfchart on page 45.
href	Optional		URL o r query column name that contains a URL to hyperlink each grid cell with.
hrefKey	Optional		The query column to use for the value appended to the `href` URL of each cell, instead of the cell's value.
target	Optional		Target of `href` URL.
appendKey	Optional	Yes	• Yes: When used with `href`, appends "GFGRIDKEY=" and the value of the cell to each cell's URL. • No
highlightHref	Optional	Yes	• Yes: highlights links associated with a `cfgrid` with an `href` attribute value • No
onValidate	Optional		A JavaScript function to validate user input. The form object, input object, and input object value are passed to routine, which returns True if validation succeeds; False otherwise.
onError	Optional		A JavaScript function to execute if validation fails.

Attribute	Req/Opt	Default	Description
gridDataAlign	Optional	Left	• Left: left-aligns data within column. • Right: right-aligns data within column. • Center: center-aligns data within column.
gridLines	Optional	Yes	• Yes: enables row and column rules in grid control • No
rowHeight	Optional		Minimum row height, in pixels, of grid control. Used with `cfgridcolumn type = "Image";` defines space for graphics to display in row.
rowHeaders	Optional	Yes	• Yes: displays a column of numeric row labels in grid control • No
rowHeaderAlign	Optional	Left	• Left: left-aligns data within row header • Right: right-aligns data within row header • Center: center-aligns data within row header
rowHeaderFont	Optional		Row label font.
rowHeaderFontSize	Optional		Row label text size in grid control, in points.
rowHeaderItalic	Optional	No	• Yes: displays row label text in italic • No
rowHeaderBold	Optional	No	• Yes: displays row label text in bold • No
rowHeaderTextColor	Optional	Black	Text color of grid control row headers. • Options: same as for `textColor` attribute
colHeaders	Optional; see description	Yes	• Yes: displays column headers in grid control • No
colHeaderAlign	Optional	Left	• Left • Right • Center
colHeaderFont	Optional		Font of column header in grid control.
colHeaderFontSize	Optional		Size of column header text in grid control, in points.
colHeaderItalic	Optional	No	• Yes: displays column headers in italics • No
colHeaderBold	Optional	No	• Yes: displays column headers in bold • No

Attribute	Req/Opt	Default	Description
colHeaderTextColor	Optional		Color of grid control column headers. • Options: same as for `textColor` attribute
bgColor	Optional		Background color of grid control. • Options: same as for `textColor` attribute
selectColor	Optional		Background color for a selected item. • Options: same as for `textColor` attribute
selectMode	Optional	Browse	Selection mode for items in grid control. • Edit: user can edit grid data. Selecting a cell opens the editor for the cell type. • Single: user selections are limited to selected cell. • Row: user selections automatically extend to the row that contains selected cell. • Column: user selections automatically extend to column that contains selected cell. • Browse: user can only browse grid data
maxRows	Optional		Maximum number of rows to display in grid.
notSupported	Optional	(See Description)	Text to display if page that contains Java applet-based cfform control is opened by a browser that does not support Java or has Java support disabled. Default: " Browser must support Java to view ColdFusion Java Applets"
pictureBar	Optional	No	• Yes: images for Insert, Delete, Sort buttons • No
insertButton	Optional	Insert	Insert button. Takes effect only if `selectmode="edit"`.
deleteButton	Optional	Delete	Text of Delete button text. Takes effect only if `selectmode="edit"`.
sortAscendingButton	Optional	A -> Z	Sort button text
sortDescendingButton	Optional	Z -> A	Sort button text

Usage You can populate a `cfgrid` with data from a `cfquery`. If you do not specify any `cfgridcolumn` entries, ColdFusion generates a default set of columns, which includes each column in the query. A default header for each column is created by replacing hyphen or underscore characters in the table column name with spaces. The first character, and any character after a space, are changed to uppercase; all other characters are lowercase.

This tag requires the client to download a Java applet; therefore, this tag might be slightly slower than using an HTML form element or the `cfinput` tag to get the same information.

For this tag to work properly. the browser must be JavaScript-enabled.

This tag requires an end tag.

How data is returned from cfgrid

This tag returns data by setting form variables in the data submitted to the form's action page, as an HTML form control does. Because the data can vary, depending on the tag's `SelectMode` attribute value, the form variables that are returned also vary depending on this value.

In general, the data returned falls into one of these categories:

- Simple data, returned from simple select operations
- Complex data, returned from insert, update and delete operations

Simple selection data (SelectMode = Single, Column, or Row)

The data that form variables return to the `cfform`'s action page contains information about which cells the user selected. In general, ColdFusion makes this data available in the action page, as ColdFusion variables in the Form scope, with the naming convention `form.#GridName#.#ColumnName#`

Each `SelectMode` returns these form variable(s):

- `SelectMode="single"`
  `form.#GridName#.#ColumnName# = "SelectedCellValue"`
- `SelectMode="column"`
  `form.#GridName#.#ColumnName# = "ValueOfCellRow1, ValueOfCellRow2, ValueOfCellRowN"`
- `SelectMode="row"`
  `form.#GridName#.#Column1Name# ="ValueOfCellInSelectedRow"`
  `form.#GridName#.#Column2Name# = "ValueOfCellInSelectedRow"`
  `form.#GridName#.#ColumnNName# = "ValueOfCellInSelectedRow"`

Complex update data (SelectMode = Edit)

The grid returns a large amount of data, to inform the action page of inserts, updates or deletes that the user made to the grid. In most cases, you can use the `cfgridupdate` tag to automatically gather the data from the form variables; the tag collects data, writes SQL calls, and updates the data source.

If you cannot use `cfgridupdate` (if, for example, you must distribute the returned data to more than one data source), you must write code to read form variables. In this mode, ColdFusion creates the following array variables in the Form scope for each `cfgrid`:

```
form.#GridName#.#ColumnName#
form.#GridName#.original.#ColumnName#
form.#GridName#.RowStatus.Action
```

Each table row that contains an update, insert, or deletion has a parallel entry in each of these arrays. To view all the information for all the changes, you can traverse the arrays, as in this example:

```
<cfloop index="ColName" list="#ColNameList#">
   <cfif IsDefined("form.#GridName#.#ColName#")>
      <cfoutput><br>form.#GridName#.#ColName#:<br></cfoutput>

      <cfset Array_New = evaluate("form.#GridName#.#ColName#")>
      <cfset Array_Orig = evaluate("form.#GridName#.original.#ColName#")>
      <cfset Array_Action = evaluate("form.#GridName#.RowStatus.Action")>

      <cfif NOT IsArray(Array_New)>
         <b>The form variable is not an array!</b><br>
      <cfelse>
         <cfset size = ArrayLen(Array_New)>
         <cfoutput>
         Result Array Size is #size#.<br>
         Contents:<br>
         </cfoutput>

         <cfif size IS 0>
            <b>The array is empty.</b><br>
         <cfelse>
            <table BORDER="yes">
               <tr>
                  <th>Loop Index</TH>
                  <th>Action</TH>
                  <th>Old Value</TH>
                  <th>New Value</TH>
               </tr>
               <cfloop index="LoopCount" from="1" to=#size#>
                <cfset Val_Orig= Array_Orig[#LoopCount#]>
                <cfset Val_New = Array_New[#LoopCount#]>
               <cfset Val_Action= Array_Action[#LoopCount#]>
               <cfoutput>
               <tr>
                  <td>#LoopCount#</td>
                  <td>#Val_Action#</td>
                  <td>#Val_Orig#</td>
                  <td>#Val_New#</td>
               </tr>
               </cfoutput>
               </cfloop>
            </table>
         </cfif>
      </cfif>
```

```
  <cfelse>
    <cfoutput>form.#GridName#.#ColName#: NotSet!</cfoutput><br>
  </cfif>
</cfloop>
```

Using the href attribute

When specifying a URL with grid items using the `href` attribute, the `selectMode` attribute value determines whether the appended key value is limited to one grid item or extends to a grid column or row. When a user clicks a linked grid item, a `cfgridkey` variable is appended to the URL, in this form:

```
http://myserver.com?cfgridkey = selection
```

If the `appendKey` attribute is set to No, no grid values are appended to the URL.

The value of *selection* is determined by the value of the `selectMode` attribute:

- If `selectMode = "Single"`, *selection* is the value of the column clicked.
- If `selectMode = "Row"`, *selection* is a delimited list of column values in the clicked row, beginning with the value of the first cell in the row.
- If `selectMode = "Column"`, *selection* is a delimited list of row values in the clicked column, beginning with the value of the first cell in the column.

Clicking the submit button while editing a grid cell occasionally causes the cell changes to be lost. To ensure that changes are submitted properly, Macromedia recommends that after user updates data in a cell, they click another cell before submitting the form.

Example

```
<!--- This shows cfgrid, cfgridcolumn, cfgridrow, and cfgridupdate --->
<!--- use a query to show the useful qualities of cfgrid --->
<!--- If the gridEntered form field has been tripped, perform gridupdate
         on table specified in database. Using default value keyonly = yes
         lets us change only information that differs from previous grid --->
<cfif IsDefined("form.gridEntered") is True>
  <cfgridupdate grid = "FirstGrid" dataSource = "cfsnippets"
  tableName = "CourseList" keyOnly = "Yes">
</cfif>
<!--- query the database to fill up the grid --->
<cfquery name = "GetCourses" dataSource = "cfsnippets">
SELECT Course_ID, Dept_ID, CorNumber,
     CorName, CorLevel, CorDesc
FROM  CourseList
ORDER by Dept_ID ASC, CorNumber ASC
</cfquery>

<html>
<head>
<title>cfgrid Example</title>
</head>
<body>
<h3>cfgrid Example</h3>
<I>Try adding a course to the database, and then deleting it.</i>
<!--- call the cfform to allow us to use cfgrid controls --->
<cfform action = "cfgrid.cfm">
<!--- We include Course_ID in cfgrid, but do not allow selection or display --->
```

```
<!--- cfgridcolumn tags are used to change the parameters involved in
         displaying each data field in the table--->
<cfgrid name = "FirstGrid" width = "450"
   query = "GetCourses" insert = "Yes" delete = "Yes" sort = "Yes"
   font = "Tahoma" bold = "No" italic = "No" appendKey = "No" highlightHref = "No"
   gridDataAlign = "LEFT" gridLines = "Yes"rowHeaders = "Yes"
   rowHeaderAlign = "LEFT" rowHeaderItalic = "No" rowHeaderBold = "No"
   colHeaders = "Yes" colHeaderAlign = "LEFT"
   colHeaderItalic = "No" colHeaderBold = "No"
   selectColor = "Red" selectMode = "EDIT" pictureBar = "No"
   insertButton = "To insert" deleteButton = "To delete"
   sortAscendingButton = "Sort ASC" sortDescendingButton = "Sort DESC">
   <cfgridcolumn name = "Course_ID" dataAlign = "LEFT"
      bold = "No" italic = "No" select = "No" display = "No"
      headerBold = "No" headerItalic = "No">
   <cfgridcolumn name = "Dept_ID" header = "Department"
      headerAlign = "LEFT" dataAlign = "LEFT"
      bold = "Yes" italic = "No" select = "Yes" display = "Yes"
      headerBold = "No" headerItalic = "Yes">
   <cfgridcolumn name = "CorNumber" header = "Course ##"
      headerAlign = "LEFT" dataAlign = "LEFT"
      bold = "No" italic = "No" select = "Yes" display = "Yes"
      headerBold = "No" headerItalic = "No">
   <cfgridcolumn name = "CorName" header = "Name"
      headerAlign = "LEFT" dataAlign = "LEFT" font = "Times" bold = "No"
      italic = "No" select = "Yes" display = "Yes" headerBold = "No"
      headerItalic = "No">
   <cfgridcolumn name = "CorLevel" header = "Level"
      headerAlign = "LEFT" dataAlign = "LEFT"
      bold = "No" italic = "No" select = "Yes" display = "Yes"
      headerBold = "No" headerItalic = "No">
   <cfgridcolumn name = "CorDesc" header = "Description"
      headerAlign = "LEFT" dataAlign = "LEFT"
      bold = "No" italic = "No" select = "Yes" display = "Yes"
      headerBold = "No" headerItalic = "No">
</cfgrid>
</cfform>
</body>
</html>
...
```

cfgridcolumn

Description Used with the cfgrid tag in a cfform. Use this tag to specify column data in a cfgrid control. The font and alignment attributes used in cfgridcolumn override global font or alignment settings defined in cfgrid.

Category Forms tags

Syntax
```
<cfgridcolumn
   name = "column_name"
   header = "header"
   width = "column_width"
   font = "column_font"
   fontSize = "size"
   italic = "Yes" or "No"
   bold = "Yes" or "No"
   textColor = "web color" or "expression"
   bgColor = "web color" or "expression"
   href = "URL"
   hrefKey = "column_name"
   target = "URL_target"
   select = "Yes" or "No"
   display = "Yes" or "No"
   type = "type"
   headerFont = "font_name"
   headerFontSize = "size"
   headerItalic = "Yes" or "No"
   headerBold = "Yes" or "No"
   headerTextColor = "web color"
   dataAlign = "position"
   headerAlign = "position"
   numberFormat = "format"
   values = "Comma separated strings and/or numeric range"
   valuesDisplay = "Comma separated strings and/or numeric range"
   valuesDelimiter = "delimiter character">
```

See also cfapplet, cfform, cfgrid tags, cfinput, cfselect, cfslider, cftextinput, cftree

History New in ColdFusion MX: if select = "no", a user cannot select and edit the cell data, regardless of the cfgrid selectmode attribute value. When clicked, the cell border (and, depending on the selectColor value, the cell background) changes color, but the cell data cannot be edited.

Attributes

Attribute	Req/Opt	Default	Description
name	Required		Name of grid column element. If grid uses a query, column name must specify name of a query column.
header	Optional	Yes	Column header text. Used only if cfgrid colHeaders = "Yes".

Attribute	Req/Opt	Default	Description
width	Optional; see description	Column head width	Column width, in pixels.
font	Optional	As specified by `cfgrid`	Font of data in column.
fontSize	Optional	As specified by `cfgrid`	Size of text in column.
italic	Optional	As specified by `cfgrid`	• Yes: displays grid control text in italics • No
bold	Optional	As specified by `cfgrid`	• Yes: displays grid control text in bold • No
textColor	Optional		Color of grid element text in column, or an expression to manipulate color; hex or text. To enter hex value, use the form "##xxxxxx", where x = 0-9 or A-F; use two pound signs or none. You can enter an expression; for example: `textColor= "(C2 LT 0 ? red : pink)"` This means: If value in Column 2 is less than 0, display value in red; otherwise, display value in pink. See "Using expressions in textColor and bgColor attributes" on page 137. • Any color, in hex format • Black • Red • Blue • Magenta • Cyan • Orange • Darkgray • Pink • Gray • White • Lightgray • Yellow
bgColor	Optional		Color of background of grid column, or an expression to manipulate color. See "Using expressions in textColor and bgColor attributes" on page 137. • Options: same as for `textColor` attribute

Attribute	Req/Opt	Default	Description
href	Optional		URL o r query column name that contains a URL to hyperlink each grid column with.
hrefKey	Optional		The query column to use for the value appended to the `href` URL of each column, instead of the column's value.
target	Optional		Frame in which to open link specified in `href`.
select	Optional		• Yes: user can select the column in grid control. • No: user cannot edit column, regardless of `cfgrid insert` and `delete` values. If `cfgrid selectMode = "Row"` or `"Browse"`, this value is ignored.
display	Optional	Yes	• Yes • No: hides column
type	Optional		• image: grid displays image that corresponds to value in column (a built-in ColdFusion image name, or an image in cfide\classes directory or subdirectory referenced with relative URL). If image is larger than column cell, it is clipped to fit. Built-in image names are as follows: - cd - computer - document - element - folder - floppy - fixed - remote • numeric: user can sort grid data numerically • boolean: column displays as checkbox; if cell is editable, user can change checkmark • string_noCase: user can sort grid data as case-insensitive text
headerFont	Optional	as specified by cfgrid	Column header font
headerFontSize	Optional	as specified by cfgrid	Column header text size, in pixels
headerItalic	Optional	as specified by cfgrid	• Yes: displays column header in italics • No
headerBold	Optional	as specified by cfgrid	• Yes: displays header in bold • No
headerTextColor	Optional		Color of grid control column header text. • Options: same as for textColor attribute

Attribute	Req/Opt	Default	Description
dataAlign	Optional	as specified by cfgrid	Column data alignment: • Left • Right • Center
headerAlign	Optional	as specified by cfgrid	Column header text alignment: • Left • Right • Center
numberFormat	Optional		Format for displaying numeric data in grid. See "numberFormat mask characters" on page 138.
values	Optional		Formats cells in column as drop-down list boxes; specify items in drop-down list. Example: `values = "arthur, scott, charles, 1-20, mabel"`
valuesDisplay	Optional		Maps elements in values attribute to string to display in drop-down list. Delimited strings and/or numeric range(s).
valuesDelimiter	Optional	, [comma]	Delimiter in `values` and `valuesDisplay` attributes.

Using expressions in textColor and bgColor attributes

The `textColor` and `bgColor` attributes accept the following kinds of values:

- A color value literal
- A hex value
- An expression that selects a text color based on the evaluation of a boolean expression

The syntax for an expression is as follows:

`(CX` *`operator string ? true_condition : false_condition`*`)`

The symbol meanings are as follows:

- `CX`: the column that contains the value to test. For the current column, use `CX`; if *n* is the column to evaluate, use `Cn`; for example, `C2`
- `operator`: One of these operators: `EQ` (equal), `GT` (greater than), `LT` (less than)
- `string`: Value to compare against. A literal, such as `(C2 EQ Johnson ? blue : green)`; or numeric: `(C2 LT 0 ? red : black)`
- `true_condition`: Value for `textColor` if condition evaluates to `"true"`
- `false_condition`: Value for `textColor` if condition evaluates to `"false"`

If the string in the expression can be interpreted as a number, the comparisons in the expression are interpreted as numeric. Otherwise, the comparison is a string comparison.

This code shows an expression that displays the grid element in blue if the grid element contains the string "Pam"; or black, otherwise:

```
<cfgridcolumn name = "FirstName" textColor = "(CX EQ Pam ? blue : black)">
```

This example displays the text in red if the value in column 1 is greater than four; or black, otherwise:

```
<cfgridcolumn name = "FirstName" textColor = "(C1 GT 4 ? blue : black)">
```

numberFormat mask characters

You can use the following `numberFormat` attribute mask characters, which correspond to those in the `NumberFormat` function, to format output in U.S. numeric and currency styles. For more information, see NumberFormat on page 567. (This tag does not support international number formatting.)

Character	Meaning
_	(Underscore) Digit placeholder.
9	Digit placeholder.
.	(Period) Location of mandatory decimal point.
0	Located to left or right of mandatory decimal point; pads with zeros.
( )	Puts parentheses around mask if number is less than 0.
+	Puts plus sign before positive numbers, minus sign before negative numbers.
-	Puts space before positive numbers, minus sign before negative numbers.
,	(Comma) Separates every third decimal-place with a comma.
L,C	Left-justilfy or center-justify number within width of mask column. First character of mask must be L or C. Default: right-justified.
$	Puts dollar sign before formatted number. Must be the first character of mask.
^	(Caret) Separates left from right formatting.

Example For a code example, see cfgrid on page 125.

cfgridrow

Description Lets you define a cfgrid that does not use a query as source for row data. If a query attribute is specified in cfgrid, the cfgridrow tags are ignored.

Category Forms tags

Syntax
```
<cfgridrow
   data = "col1, col2, ...">
```

See also cfapplet, cfform, cfgrid tags, cfinput, cfselect, cfslider, cftextinput, cftree

Attributes

Attribute	Req/Opt	Default	Description
data	Required		Delimited list of column values. If a value contains a comma, it must be escaped with another comma.

Usage The following code shows how to populate a grid from a list with the cfgridrow tag:

```
<cfset MyList1 = "Rome,Athens,Perth,Brasilia">
<cfset MyList2 = "Italy,Greece,Australia,Brazil">
<cfform
   name = "someform" action = "cfform.cfm">
<cfgrid name="GeoGrid"  autowidth = "yes" vspace = "4"
         height = "120" font="tahoma" gridlines="yes"
         rowheaders="yes" rowheaderalign="left" colheaders="yes" >

   <cfgridcolumn NAME="City" header="City">
   <cfgridcolumn NAME="Country" header="Country">

   <cfloop index="Counter" from="1" to="#ListLen(MyList1)#">
      <cfgridrow data =
         "#ListGetAt(MyList1,Counter)#,#ListGetAt(MyList2,(Counter))#">
   </cfloop>
</cfgrid>
</cfform>
```

Example For a code example, see cfgrid on page 125.

The following example populates two grids from the results of a query, as follows:

- One uses cfgridrow within the cfquery tag, using the query to generate rows
- One uses cfgridrow outside the cfquery tag, using a cfloop tag to generate rows.

```
<!--- This example shows cfgrid, cfgridcolumn, cfgridrow, cfgridupdate tags --->
<!--- If the gridEntered form field has been tripped, perform the gridupdate
         on the table specified in the database. Using the default value
         keyonly=yes lets us change only the information that differs from
         the previous grid --->
<cfif isdefined("form.gridentered") is true>
<cfgridupdate grid="FirstGrid" datasource="cfsnippets"
   tablename="CourseList"  keyonly="Yes">
</cfif>
```

```
<!--- query the database to fill up the grid --->
<cfquery name="GetCourses" datasource="cfsnippets">
   select   Course_ID, Dept_ID, CorNumber, CorName, CorLevel, CorDesc
   FROM     CourseList
   ORDER by Dept_ID ASC, CorNumber ASC
</cfquery>
<html>
<head>
<title>cfgridrow example</title>
</head>
<body>
<h3>cfgridrow Example</h3>
<I>Try adding a course to the database, and then deleting it.</I>
<!--- call the cfform to allow us to use cfgrid controls --->
<cfform action="cfgridrow.cfm">

<!--- When inserting rows while running under UNIX, you must also specify
         a value for Course_ID --->
<!--- cfgridcolumn tags are used to change the parameters involved in
         displaying each data field in the table--->

<cfgrid name="FirstGrid" width="600" query="GetCourses" insert="yes"
   delete="yes" sort="yes" font="tahoma" bold="no" italic="no"
   appendkey="no" highlighthref="no" griddataalign="left" gridlines="yes"
   rowheaders="yes" rowheaderalign="left" rowheaderitalic="no"
   rowheaderbold="no" colheaders="yes" colheaderalign="left"
   colheaderitalic="no" colheaderbold="no" selectcolor="red"
   selectmode="edit" picturebar="no" insertbutton="to insert"
   deletebutton="to delete" sortascendingbutton="sort asc"
   sortdescendingbutton="sort desc">
   <cfgridcolumn name="Dept_ID" header="Department" headeralign="left"
          dataalign="left" bold="yes" italic="no" select="yes"
          display="yes" headerbold="no" headeritalic="yes">
   <cfgridcolumn name="CorNumber" header="Course ##" headeralign="left"
          dataalign="left" bold="no" italic="no" select="yes"
          display="yes" headerbold="no" headeritalic="no">
   <cfgridcolumn name="CorName" header="Name" headeralign="left"
          dataalign="left" font="times" bold="no" italic="no" select="yes"
          display="yes" headerbold="no" headeritalic="no">
   <cfgridcolumn name="CorLevel" header="Level" headeralign="left"
          dataalign="left" bold="no" italic="no" select="yes" display="yes"
          headerbold="no" headeritalic="no">
   <cfgridcolumn name="CorDesc" header="Description" headeralign="left"
          dataalign="left" bold="no" italic="no" select="yes" display="yes"
          headerbold="no" headeritalic="no">
   <cfgridcolumn name="Course_ID" header="Course ID (Do Not Specify on NT)"
          dataalign="left" bold="no" italic="no" select="yes" display="yes"
          headerbold="no" headeritalic="no">
</cfgrid>

<!--- send the grid back to this page, where we  determine whether anything has
         changed, and thus whether to run the cfgridupdate --->
<input type="Submit" name="submit" value="Apply Changes">
<input type="hidden" name="gridEntered" value="yes">
```

```
<h3>Example Two</h3>
<p>This grid shows how the same grid can be built using cfgridrow with cfloop
        (i.e., defining query external to cfgrid, rather than within cfgrid).</p>
<!--- cfgridcolumn is used to define  container columns.
    cfgridrow is used to define the data put into those containers --->
<cfgrid name="SecondGrid" width=600 insert="no"
  delete="no" sort="yes" bold="no" italic="no"
  appendkey="no" highlighthref="no" griddataalign="left" gridlines="yes"
  rowheaders="no" rowheaderalign="left" rowheaderitalic="no"
  rowheaderbold="no" colheaders="yes" colheaderalign="left"
  colheaderitalic="no" colheaderbold="no" selectmode="browse"
  picturebar="yes">
  <cfgridcolumn name="Course_ID" dataalign="left" bold="no" italic="no"
        select="no" display="no" headerbold="no" headeritalic="no">
  <cfgridcolumn name="Dept_ID" header="Department" headeralign="left"
        dataalign="left" bold="yes" italic="no" select="yes"
        display="yes" headerbold="no" headeritalic="yes">
  <cfgridcolumn name="CorNumber" header="Course ##" headeralign="left"
        dataalign="left" bold="no" italic="no" select="yes"
        display="yes" headerbold="no" headeritalic="no">
  <cfgridcolumn name="CorName" header="Name" headeralign="left"
        dataalign="left" font="times" bold="no" italic="no"
        select="yes" display="yes" headerbold="no" headeritalic="no">
  <cfgridcolumn name="CorLevel" header="level" headeralign="left"
        dataalign="left" bold="no" italic="no" select="yes"
        display="yes" headerbold="no" headeritalic="no">
  <cfgridcolumn name="CorDesc" header="Description" headeralign="LEFT"
        dataalign="left" bold="no" italic="no" select="yes" display="yes"
        headerbold="no" headeritalic="no">
<!--- use cfloop, loop through query, define cfgridrow data each time through --->
  <cfloop query="GetCourses">
    <cfgridrow data="#Course_ID#,#Dept_ID#,#CorNumber#,#CorName#,
        #CorLevel#,#CorDesc#">
  </cfloop>
</cfgrid>
</cfform>
</body>
</html>
```

cfgridupdate

Description

Used within a cfgrid tag. Updates data sources directly from edited grid data. This tag provides a direct interface with your data source.

This tag applies delete row actions first, then insert row actions, then update row actions. If it encounters an error, it stops processing rows.

Category

Forms tags

Syntax

```
<cfgridupdate
   grid = "gridname"
   dataSource = "data source name"
   tableName = "table name"
   username = "data source username"
   password = "data source password"
   tableOwner = "table owner"
   tableQualifier = "qualifier"
   keyOnly = "Yes" or "No">
```

See also

cfapplet, cfform, cfgrid tags, cfinput, cfselect, cfslider, cftextinput, cftree

History

New in ColdFusion MX: The connectString, dbName, dbServer, dbtype, provider and providerDSN attributes are deprecated. Do not use them. They do not work, and might cause an error, in releases later than ColdFusion 5.

Attributes

Attribute	Req/Opt	Default	Description
grid	Required		Name of cfgrid form element that is the source for the update action.
dataSource	Required		Name of data source for the update action.
tableName	Required		Name of table to update. For ORACLE drivers, entry must be upper-case. For Sybase driver, entry is case-sensitive; must be same case as used when table was created
username	Optional		Overrides username value specified in ODBC setup.
password	Optional		Overrides password value specified in ODBC setup.
tableOwner	Optional		Table owner, if supported.
tableQualifier	Optional		Table qualifier, if supported. Purpose: • SQL Server and Oracle driver: name of database that contains table • Intersolv dBASE driver: directory of DBF files

Attribute	Req/Opt	Default	Description
keyOnly		No	Applies to the `update` action: • Yes: the WHERE criteria are limited to the key values • No: the WHERE criteria include key values and the original values of changed fields

Example

```
<!--- This example shows the cfgridupdate tag--->
...
<!--- If the gridEntered form field has been tripped, perform
         the gridupdate on the table specified in the database.
Using the default value keyonly = yes lets us change only the
         information that differs from the previous grid --->
<cfif IsDefined("form.gridEntered") is True>
<cfgridupdate grid = "FirstGrid" dataSource = "cfsnippets"
 tableName = "CourseList" keyOnly = "Yes">
</cfif>
...
```

cfheader

Description Generates custom HTTP response headers to return to the client.

Category Data output tags, Page processing tags

Syntax
```
<cfheader
   name = "header_name"
   value = "header_value">
or
<cfheader
   statusCode = "status_code"
   statusText = "status_text">
```

See also `cfcache`, `cfflush`, `cfhtmlhead`, `cfinclude`, `cfsetting`, `cfsilent`

Attributes

Attribute	Req/Opt	Default	Description
name	Required if statusCode not specified		Header name
value	Optional		HTTP header value
statusCode	Required if `name` not specified		Number. HTTP status code
statusText	Optional		Explains status code

Usage If you use this tag after the `cfflush` tag on a page, an error is thrown.

Example
```
<h3>cfheader Example</h3>

<p>cfheader generates custom HTTP response headers to return to the client.
<p>This example forces browser client to purge its cache of requested file.
<cfheader name = "Expires" value = "#Now()#">
```

cfhtmlhead

Description: Writes text to the `head` section of a generated HTML page. It is useful for embedding JavaScript code, or putting other HTML tags, such as `meta`, `link`, `title`, or `base` in an HTML page header.

Category: Page processing tags

Syntax:

```
<cfhtmlhead
  text = "text">
```

See also: `cfcache`, `cfflush`, `cfheader`, `cfheader`, `cfinclude`, `cfsetting`, `cfsilent`

Attributes:

Attribute	Req/Opt	Default	Description
text	Required		Text to add to the <head> area of an HTML page.

Usage: If you use this tag after the `cfflush` tag on a page, an error is thrown.

Example:

```
<body>
<!--- This example shows the use of cfhtmlhead --->
<cfhtmlhead
  text="<meta name=""Description""
  content=""This is an example of a generated header"">">

<p>cfhtmlhead writes the text specified in the text attribute to the
        &lt;HEAD&gt;section of a generated HTML page. cfhtmlhead can be
        useful for embedding JavaScript code, or placing other HTML tags
        such as META, LINK, TITLE, or BASE in an HTML header.
<p>View the source of this frame to see that the title of the page is
        generated by the cfhtmlhead tag.
</body>
```

cfhttp

Description

Executes HTTP POST and GET operations on files. You can execute standard GET operations and create a query object from a text file. POST operations upload MIME file types to a server, or post cookie, formfield, URL, file, or CGI variables directly to a server.

Category

Forms tags, Internet Protocol tags

Syntax

```
<cfhttp
   url = "hostname"
   port = "port_number"
   method = "get_or_post"
   username = "username"
   password = "password"
   name = "queryname"
   columns = "query_columns"
   firstrowasheaders = "yes" or "no"
   path = "path"
   file = "filename"
   delimiter = "character"
   textQualifier = "character"
   resolveURL = "yes" or "no"
   proxyServer = "hostname"
   proxyPort = "port_number"
   userAgent = "user_agent"
   throwOnError = "yes" or "no"
   redirect = "yes" or "no"
   timeout = "timeout_period"
   charset = "character set">
</cfhttp>
```

See also

cfftp, cfhttpparam, cfldap, cfmail, cfmailparam, cfpop

History

New in ColdFusion MX:

- The `firstrowasheaders` attribute is new.
- The `charset` attribute is new.
- ColdFusion uses the Sun JSSE library, which supports 128-bit encryption, to support Secure Sockets Layer (SSL).

Attributes

Attribute	Req/Opt	Default	Description
url	Required	http	Absolute URL of host name or IP address of server on which file resides. URL must include protocol (http or https) and hostname. It can contain a port number. This value overrides `port` value.
port	Optional	80	Port number on server from which object is requested. When used with `resolveURL`, URLs of retrieved documents that specify a port number are automatically resolved, so that links in retrieved document remain functional. Value in `url` attribute overrides this value.

Attribute	Req/Opt	Default	Description
method	Optional	get	• get: downloads text or binary file, or create query from the contents of a text file. • post: sends information to server page or CGI program for processing. Requires a `cfhttpparam` tag.
username	Optional		A username. May be required by server.
password	Optional		A password. May be required by server
name	Optional		Name to assign to a query if it is constructed from returned HTTP results.
columns	Optional		Column names for a query, when creating it as a result of a `cfhttp method = "get"`. By default, the first row of a text file is interpreted as column heads. If there are column heads in the text file from which the query is drawn, do not specify this attribute, except to overwrite them. If a duplicate column heading is encountered, ColdFusion appends an underscore to the name to make it unique. If there are no column headers in the text file, or to override those in the file, specify the `columns` attribute. ColdFusion never processes the first row of a file as data,.
firstrowas headers	Optional	yes	Determines how ColdFusion processes the first row of the query record set: • yes: - If the `columns` attribute is not specified: processes as column heads - If the `columns` attribute is specified: this attribute is not used • no: - If the `columns` attribute is not specified: processes as data, and generates column names; for example, "column_1" - If the `columns` attribute is specified: this attribute is not used
path	Optional		Path to directory in which to store file. If path is not specified in POST or GET, a variable (`cfhttp.fileContent`) is created; you can use it to display the `post` results, in a `cfoutput` tag.
file	Required if `post` and if path is specified		Name of file that is accessed. For a `get` operation, defaults to name specified in `url`. Enter path information in the `path` attribute.
delimiter	Required to create query	, [comma]	• tab • comma

Attribute	Req/Opt	Default	Description
textQualifier	Required to create query	"" [double quotation mark]	Start and end of a column. If it is embedded in column, this value must be escaped. For example, if qualifier is a double quotation mark, it is escaped as """". If there is no text qualifier in the file, specify " ".
resolveURL	Optional	No	Applies to `get` and `post`. • Yes: internal URLs in a page reference returned into `fileContent` internal variable are fully resolved, including port number, so that links in the page remain functional. Applies to these HTML tags: - img src - a href - form action - applet code - script src - embed src - embed pluginspace - body background - frame src - bgsound src - object data - object classid - object codebase - object usemap
proxyServer	Optional		Host name or IP address of a proxy server.
proxyPort	Optional	80	Port number on proxy server from which object is requested. When used with `resolveURL`, the URLs of retrieved documents that specify a port number are automatically resolved, so that links in the retrieved document remain functional.
userAgent	Optional		User agent request header.
throwOnError	Optional	No	• Yes: throw an exception that can be caught with the `cftry` and `cfcatch` tags • No
redirect	Optional	Yes	If No, and `throwOnError = "yes"`: if `cfhttp` fails, execution stops, and the status code and associated error message are returned in the variable `cfhttp.statuscode`. To determine where execution would be redirected, use the variable `cfhttp.responseHeader[LOCATION]`. The key `LOCATION` shows the redirection path. ColdFusion follows a maximum of four redirects on a request. If there are more, ColdFusion functions as if `redirect = "no"`. • Yes: redirect execution • No: stop execution

Attribute	Req/Opt	Default	Description
timeout	Optional		Value, in seconds. When a URL timeout is specified in the browser, this setting takes precedence over the ColdFusion Administrator timeout, and ColdFusion uses the lesser of the URL timeout and the timeout passed in the `timeout` attribute, so that the request always times out before, or at the same time as, the page. If URL timeout is not specified, ColdFusion uses the lesser of the Administrator timeout and the timeout passed in the `timeout` attribute. If the timeout is not set in any of these, ColdFusion waits indefinitely for the cfhttp request to process. This attribute does not function with JDK 1.3.
charset	Optional	The server charset; if none, UTF-8	A Java character set name for the file or url in a GET or POST. The following values are typically used: • UTF-8 • ISO-8859-1 • UTF-16 • US-ASCII • UTF-16BE • UTF-16LE For a list of character sets, see: http://www.w3.org/International/O-charset-lang.html

Usage If this value includes the substring "https" (which indicates that the object of the request is a secure server) but does not include a port number, the default value of the `port` attribute is 443. (If the `URL` value includes a port number, its value overrides the value of `port`.)

For the ColdFusion Administrator timeout and the URL timeout to take effect, you must enable the timeout in the ColdFusion Administrator, Server Settings page. For more information, see *Administering ColdFusion MX*.

Variables returned by a cfhttp get operation

The `cfhttp` tag returns data in variables. For example, if you specify a URL that points to a text or binary file in a `cfhttp method = "get"` operation, the file is downloaded and stored in a ColdFusion variable or file.

- The `fileContent` variable is available for text and MIME file types.
- The `mimeType` variable is available for all file manipulations.
- The Header and `responseHeader` variables display response headers.

These variables can be accessed in this way:

```
#cfhttp.fileContent#
#cfhttp.mimeType#
#cfhttp.header#
#cfhttp.responseHeader[http_header_key]#
```

The `responseHeader` variable is returned as a CFML structure; the others, as strings.

Building a query from a delimited text file

To download a file in a ColdFusion page so that a query can be built using the file, the file must be either comma- or tab-delimited. Macromedia recommends that you include text qualification. The file is parsed and an appropriate query is built from it. You can specify columns in the attribute list so the client can override the columns specified in the file.

There is error checking within the tag that prevents a user from entering an invalid column name, or using an invalid column name that was specified in the original file. If an illegal filename is encountered, the illegal characters are stripped. This action could produce duplicate column names, so duplicate columns are renamed and inserted into the query header.

The query has all of the functionality of a standard `cfquery` object, as follows:

- HTTP POST: You can nest `cfhttpparam` tags within a `cfhttp` tag. The browser can be pointed to a URL that specifies a CGI executable or a ColdFusion page. Because `cfhttpparam` tags can be nested, you can construct a multipart/form-data style post. A file content variable is created; you can use it in a `cfoutput` tag. If `path` and `file` attributes are specified, the data returned from the server is saved to the specified location.
- Authentication: `cfhttp` supports Windows NT Basic Authentication for GET and POST operations. However, Basic Authentication does not work if your web server has Windows NT Challenge/Response (Microsoft IIS) enabled.
- Encryption: `cfhttp` uses SSL to negotiate secure transactions.
- `cfhttp.statuscode: cfhttp` provides the `cfhttp.statuscode` variable for access to the HTTP error string associated with the error if the `throwOnError` attribute is set to No (or if it is not specified, because it defaults to No).

The `cfhttp` tag returns the following variables:

Name	Description
`#cfhttp.fileContent#`	File contents, for text and MIME files.
`#cfhttp.mimeType#`	MIME type.
`#cfhttp.responseHeader[http_hd_key]#`	Response headers. If there is one instance of a header key, value can be accessed as simple type. If there is more than one instance, values are put in an array in `responseHeader` structure.
`#cfhttp.header#`	Raw response header.
`#cfhttp.statuscode#`	HTTP error code and associated error string, if `throwOnError` = "no".

Terminate a `cfhttp method = "post"` operation with `</cfhttp>`.

Example

```
<!--- This example shows the use of CFHTTP to pull information dynamically from
the snippets directory --->
<p>This view-only example shows the ability of CFHTTP to pull the contents of
a web resource from the Internet or from a local directory.
<!---
<form action="index.cfm" method="POST">
Try entering a URL for the tag to return:
   <input type="Text" size="25" name="urladdress"
      value="http://www.macromedia.com">
   <input type="Submit" name="" value="get page">
</form>
<!----- sets a default value for a URL to retrieve ----->
<cfparam name="urladdress"
   default="http://localhost/cfdocs/index.htm">
<!--- if we have passed a url address in the form, display the address --->
<cfif IsDefined("form.urladdress") is True>
   <!--- do simple error check to avoid crashing the tag --->
   <cfif Trim(Form.urladdress) is ""
      or Trim(Form.urladdress) is "http://">
      <!--- if error condition tripped, set alternative --->
      <cfset urlAddress ="http://localhost/cfdocs/index.htm">
      <h4>because you entered no url or an empty string, the tag will
      return the following address: http://localhost/cfdocs/index.htm</h4>
   <cfelse>
      <!--- otherwise use address passed from form --->
      <cfset urlAddress = form.urladdress>
   </cfif>
   <!--- use CFHTTP tag to get the file content represented by urladdress --->
   <cfhttp url="#urladdress#"
      method="GET" resolveurl="YES">
   </cfhttp>
<cfelse>
   <!--- the first time through, retrieve a URL that we know exists ---->
   <cfhttp url="http://localhost/cfdocs/index.htm"
      method="GET" resolveurl="YES">
   </cfhttp>
</cfif>
<!---- Now, output the file, including the mimetype and content ------>
<h3>Show the file</h3>
<cfoutput>
   <p>Your file was of type: #CFHTTP.MimeType#
   <p>#HTMLCodeFormat(CFHTTP.FileContent)#
</cfoutput> --->
```

cfhttpparam

Description Required for a `cfhttp` POST operation. Specifies parameters to build the operation.

Category Forms tags, Internet Protocol tags

Syntax
```
<cfhttpparam
   name = "name"
   type = "type"
   value = "transaction type"
   file = "filename">
```

See also `cfftp`, `cfhttp`, `cfldap`, `cfmail`, `cfmailparam`, `cfpop`

Attributes

Attribute	Req/Opt	Default	Description
name	Required		Variable name for data that is passed
type	Required		Transaction type: • URL • FormField • Cookie • CGI • File
value	Optional if `type = "File"`		Value of URL, FormField, Cookie, File, or CGI variables that are passed.
file	Required if `type = "File"`		File name

Usage This tag's parameter values are URL-encoded, so the values of special characters (such as the ampersand) are preserved when they are passed to the server. For more information, see the function URLEncodedFormat on page 658.

Example
```
<p>This view-only example shows the use of cfhttpparam to show the values
         of passed variables on another HTML reference, accessed by cfhttp.
         The other file could output the value of form.formtest, url.url_test,
         cgi.cgi_test, and cookie.cookie_test to prove that this page is working:

<h3>Sample Other File Listing</h3>
<cfoutput>#HTMLCodeFormat("

<html>
<head>
<title>Sample Page</title>
</head>

<body>
<h3>Output the passed variables</h3>
<cfoutput>
   Form variable: ##form.form_test##
```

```
   <br>URL variable: ##URL.url_test##
   <br>Cookie variable: ##Cookie.cookie_test##
   <br>CGI variable: ##CGI.cgi_test##
</cfoutput>
</body>
</html>

")#</cfoutput>

<!--- <p>
<cfhttp
     method = "post" url = "http://localhost/someotherfile.cfm">
   <cfhttpparam
     name = "form_test" type = "FormField" value = "This is a form variable">
   <cfhttpparam
     name = "url_test" type = "URL" value = "This is a URL variable">
   <cfhttpparam
     name = "cgi_test" type = "CGI" value = "This is a CGI variable">
   <cfhttpparam
     name = "cookie_test" type = "Cookie" value = "This is a cookie">
</cfhttp>

<cfoutput>
    #cfhttp.fileContent#
</cfoutput> --->
```

cfif

Description Creates simple and compound conditional statements in CFML. Tests an expression, variable, function return value, or string. Used, optionally, with the cfelse and cfelseif tags, which are also described in this section.

Category Flow-control tags

Syntax
```
<cfif expression>
   HTML and CFML tags
<cfelseif expression>
   HTML and CFML tags
<cfelse>
   HTML and CFML tags
</cfif>
```

See also cfabort, cfbreak, cfexecute, cfexit, cflocation, cfloop, cfswitch, cfthrow, cftry

Usage When testing the return value of a function that returns a Boolean, you do not have to define the True condition explicitly. This example uses the IsArray function:

```
<cfif IsArray(myarray)>
```

If successful, IsArray evaluates to Yes, the string equivalent of the Boolean True. This is preferred over explicitly defining the True condition this way:

```
<cfif IsArray(myarray) IS True>
```

For example:

```
<cfif "11/23/1998 " GT "11/15/1998">
```

This switch is set on the ColdFusion Administrator Server Settings page. For more information, see *Administering ColdFusion MX*.

This tag requires an end tag.

Example In this example, variables are shown within pound signs. This is not required.

```
<!--- This example shows the interaction of cfif, cfelse, and cfelseif --->
<!----- first, perform a query to get some data ----->
<cfquery name="getCenters" datasource="cfsnippets">
   SELECT Center_ID, Name, Address1, Address2, City, State, Country,
      PostalCode, Phone, Contact
   FROM Centers
   ORDER by City, State, Name
</cfquery>
<p>CFIF gives us the ability to perform conditional logic based on a condition
or set of conditions.
<p>For example, we can output the list of Centers from the snippets datasource
by group and only display them <b>IF</b> City = San Diego.
<hr>
<!---- use CFIF to test a condition when outputting a query ----->
<p>The following centers are in San Diego:
<cfoutput query="getCenters">
     <cfif Trim(City) is "San Diego">
        <br><b>Name/Address:</b>#Name#, #Address1#, #City#, #State#
        <br><b>Contact:</b> #Contact#
```

```
        <br>
      </cfif>
</cfoutput>
<hr>
<p>If we would like more than one condition to be the case, we can ask for
a list of the centers in San Diego <b>OR</b> Santa Ana. If the center
does not follow this condition, we can use CFELSE to show only
the names and cities of the other centers.
<p>Notice how a nested CFIF is used to specify the location of
the featured site (Santa Ana or San Diego).
<!----- use CFIF to specify a conditional choice for multiple options;
also note the nested CFIF --->
<p>Complete information is shown for centers in San Diego or Santa Ana.
All other centers are listed in italics:
<cfoutput query="getCenters">
   <cfif Trim(City) is "San Diego" OR Trim(City) is "Santa Ana">
      <h4>Featured Center in
         <cfif Trim(City) is "San Diego">
            San Diego
         <cfelse>
            Santa Ana
         </cfif>
         </h4> <b>Name/Address:</b>#Name#, #Address1#, #City#, #State#
         <br><b>Contact:</b> #Contact#<br>
   <cfelse>
      <br><i>#Name#, #City#</i>
   </cfif>
</cfoutput>
<hr>
<p>Finally, we can use CFELSEIF to cycle through a number of conditions and
produce varying output. Note that you can use CFCASE and CFSWITCH for a more
elegant representation of this behavior.
<p> <!------- use CFIF in conjunction with CFELSEIF to specify more than one
branch in a conditional situation --->
<cfoutput query="getCenters">
   <cfif Trim(City) is "San Diego" OR Trim(City) is "Santa Ana">
      <br><i>#Name#, #City#</i> (this one is in
           <cfif Trim(City) is "San Diego">San Diego
           <cfelse>Santa Ana
           </cfif>)
        <cfelseif Trim(City) is "San Francisco">
           <br><i>#Name#, #City#</i> (this one is in San Francisco)
        <cfelseif Trim(City) is "Suisun">
           <br><i>#Name#, #City#</i> (this one is in Suisun)
        <cfelse> <br><i>#Name#</i>
           <b>Not in a city we track</b>
        </cfif>
</cfoutput>
```

cfimpersonate

Description: This tag is obsolete. Use the newer security tools; see "Authentication functions" on page 328 and the Application Security chapter in *Developing ColdFusion MX Applications with CFML*.

History: New in ColdFusion MX: this tag is obsolete. It does not work in ColdFusion MX and later releases.

cfimport

Description
You can use the `cfimport` tag to import custom tags from a directory as a tag library, or to import a Java Server Page (JSP) tag library. A JSP tag library is a packaged set of tag handlers that conform to the JSP 1.1 tag extension API.

Category
Application framework tags

Syntax
```
<cfimport
   taglib = "taglib-location"
   prefix = "custom"
   webservice = "URL">
```

See also
`cfapplication`

History
New in ColdFusion MX: This tag is new.

Attributes

Attribute	Req/Opt	Default	Description
taglib	Required		Tag library URI: • A directory in which customer tags are stored • A URL path to a JAR in a web-application; for example, "/WEB-INF/lib/sometags.jar" • A path to a tag library descriptor; for example, "/sometags.tld"
prefix			Prefix by which to access imported JSP tags on a CFML page. To import tags directly, specify an empty value, " ". Use this to create server-side versions of common HTML tags.
webservice	Optional		URL of a WSDL file.

Usage
The following example imports the tags from the directory myCustomTags:

```
<cfimport
   prefix="mytags"
   taglib="myCustomTags">
```

You can import multiple tag libraries using one prefix. If there are duplicate tags in a library, the first one takes precedence.

JSP tags have fixed attributes; however, if the tag supports runtime attribute expressions, most tag libraries support the use of the syntax `#expressions#`.

To reference a JSP tag in a CFML page, use the syntax `<prefix:tagname>`. Set the prefix value in the `prefix` attribute.

To use JSP custom tags in a ColdFusion page, follow these steps:

1 Put a JSP tag library JAR file (for example, myjsptags.jar) into the ColdFusion server directory wwwroot/WEB-INF/lib.

2 At the top of a CFML page, insert code such as the following:

```
<cfimport
  prefix="mytags"
  taglib="/WEB-INF/lib/myjsptags.jar">
```

To reference a JSP tag from a JAR file, use the following syntax:

```
<cfoutput>
  <mytags:helloTag message="#mymessage#" />
<cfoutput>
```

The `cfimport` tag does not work within a `cfinclude` call. ColdFusion does not throw an error in this situation. To include tags from a JSP tag library, you must put the `cfimport` tag at the top of every page that is called by the `cfinclude` tag. (Putting the `cfimport` tag into an application.cfm file does not propagate the import.) For example, put a statement such as the following in each page that is called by the `cfinclude` tag:

```
<cfimport taglib="/WEB-INF/webcomponent" prefix="c">
```

You cannot use this tag to suppress output from a tag library.

For more information, see the Java Server Page 1.1 specification.

Example

```
<h3>cfimport example - view only</h3>
<cftry>
  <!--- Import the JRun Custom Tag Libraries --->
  <cfimport
        taglib="jruntags.jar"
        prefix="customtag">

  <customtag:sql datasrc="#regdatasource#" id="x">
    SELECT employee_id, firstname, lastname, email, phone, department
    FROM employees
    WHERE employee_id < 3
  </customtag:sql>

  <customtag:param id="x" type="allaire.taglib.querytable"/>
  <table>
    <customtag:foreach group="#x#">
    <tr>
      <cfoutput>
        <TD>#x.get("employee_id")#</TD>
        <TD>#x.get("firstname")#</TD>
        <TD>#x.get("lastname")#</TD>
        <TD>#x.get("email")#</TD>
        <TD>#x.get("phone")#</TD>
        <TD>#x.get("department")#</TD>
      </cfoutput>
    </tr>

    </customtag:foreach>
  </table>
```

```
   <cfcatch>
      <cfoutput>#CFCATCH.message# - #CFCATCH.DETAIL# </cfoutput>
   </cfcatch>
</cftry>

<h3>id= cfimport-jruntags-sql-pos-1-t1</h3>
<table>
   <tr>
      <td>1</td>
      <td>carolynn</td>
      <td>peterson</td>
      <td>cpeterson</td>
      <td>(612) 832-7654</td>
      <td>sales</td>
   </tr>
   <tr>
      <td>2</td>
      <td>dave</td>
      <td> heartsdale</td>
      <td>fheartsdale</td>
      <td>(612) 832-7201</td>
      <td>accounting</td>
   </tr>
</table>
```

cfinclude

Description
Embeds references to ColdFusion pages in CFML. You can embed `cfinclude` tags recursively. For another way to encapsulate CFML, see cfmodule on page 208. (A ColdFusion page was formerly sometimes called a ColdFusion template or a template.)

Category
Flow-control tags, Page processing tags

Syntax
```
<cfinclude
   template = "template_name">
```

See also
`cfcache`, `cfflush`, `cfheader`, `cfhtmlhead`, `cfsetting`, `cfsilent`

History
New in ColdFusion MX: If you use this tag to include a CFML page whose length is zero bytes, an empty page displays. (In earlier releases, an error message displays.)

Attributes

Attribute	Req/Opt	Default	Description
template	Required		A logical path to a ColdFusion page.

Usage
ColdFusion searches for included files in the following sequence:

1 In the directory of the current page

2 In directories mapped in the ColdFusion Administrator for the included file

Example
```
<!--- This example shows the use of cfinclude to paste CFML
         or HTML code into another page dynamically --->

<h4>This example includes the main.htm page from the CFDOCS directory.
         The images do not display, because they are located in
         a separate directory. However, the page appears fully rendered
         within the contents of this page.</h4>
<cfinclude template = "/cfdocs/main.htm">
```

cfindex

Description

Populates a Verity search engine collection with an index of documents on a file system or of ColdFusion query result sets.

A collection must exist before it can be populated.

A collection can be **indexed** in the following ways:

- In ColdFusion, with the `cfindex` tag
- In the ColdFusion Administrator, which calls the `cfindex` tag
- Using a native Verity indexing tool, such as Vspider or MKVDK

For more information, see *Developing ColdFusion MX Applications with CFML*.

Category

Extensibility tags

Syntax

```
<cfindex
   collection = "collection_name"
   action = "action"
   type = "type"
   title = "title"
   key = "ID"
   body = "body"
   custom1 = "custom_value"
   custom2 = "custom_value"
   URLpath = "URL"
   extensions = "file_extensions"
   query = "query_name"
   recurse = "Yes" or "No"
   language = "language">
```

See also

`cfcollection`, `cfexecute`, `cfobject`, `cfreport`, `cfsearch`, `cfwddx`

History

New in ColdFusion MX:

- The `action` attribute value `optimize` is obsolete. It does not work, and might cause an error, in ColdFusion MX.
- It is not necessary to specify the `external` attribute. (ColdFusion automatically detects whether a collection is internal or external.)
- ColdFusion supports Verity operations on Acrobat PDF files.
- This tag can throw the SEARCHENGINE exception.
- This tag accepts collection names that include spaces.

Attributes

Attribute	Req/Opt	Default	Description
collection	Required		• Name of a collection that is registered by ColdFusion; for example, "personnel" • Name and absolute path of a collection that is not registered by ColdFusion; for example: "e:\collections\personnel"
action	Depends on collection attribute value; see Usage section		• update: updates a collection and adds key to the index. Do not use the cflock tag with this option. • delete: deletes data in the entities specified by the type attribute • purge: deletes all keys from a collection • refresh: purges all keys from a collection, then updates it.
type	Optional	custom, if query attribute is specified. file, otherwise.	• file: using the key attribute value of the query result as input, applies action value to filenames or filepaths. • path: using the key attribute value of the query result as input, applies action to filenames or filepaths that pass the extensions filter • custom: If action = "update" or "delete": applies action to custom entities in query results.
title	Optional		• Title for collection • Query column name for type and a valid query name Permits searching collections by title or displaying a separate title from the key
key	Depends on action attribute value; see Usage section	(empty string)	• Absolute path and file name, if type = "file" • Absolute path, if type = "path" • A query column name (typically, the primary key column name), if type = "custom" • A query column name, if type = any other value This attribute is required for the actions listed, unless you intend for its value to be an empty string.
body	Required if type = "custom"; see Usage section		• ASCII text to index • Query column name(s), if name is specified in query You can specify columns in a delimited list. For example: "emp_name, dept_name, location"
custom1	Optional		Custom field in which you can store data during an indexing operation. Specify a query column name for type, and a query name.
custom2	Optional		Usage is the same as for custom1.
URLpath	Optional		If type="file" or "path", specifies the URL path. When the collection is searched with cfsearch, this pathname is prefixed to file names and returned as the url attribute.

Attribute	Req/Opt	Default	Description
extensions	Optional	HTM, HTML, CFM, CFML, DBM, DBML	Delimited list of file extensions that ColdFusion uses to index files, if `type = "Path"`. `"*."` returns files with no extension. For example: the following code returns files with a listed extension or no extension: `extensions = ".htm, .html, .cfm, .cfml, "*."`
query	Required if `type = "custom"`; see Usage section		Query against which collection is generated
recurse	Optional	No	• Yes: if `type = "path"`, directories below the path specified in key are included in indexing operation • No
language	Optional	English	For options, see cfcollection on page 56. Requires the appropriate (European or Asian) Verity Locales language pack.

Usage

This tag populates Verity search engine collections with metadata from the following sources:

- Documents stored on a file system
- ColdFusion query result sets

The following table shows the dependent relationships among this tag's attribute values:

Specifying this attribute is required, optional or unnecessary (blank):	**For this `action` attribute value:**		
	`purge`	`delete`	`update` or `refresh`
`collection`	Required	Required	Required
`type`		Optional	If type = "file", "path", or "custom": Optional.
`title`			If type = "file", "path", or "custom": Optional. Otherwise: unnecessary.
`key`		Required	If type = "file", "path", or "custom": Required. Otherwise: unnecessary.
`body`			If type = "custom": Required. Otherwise: unnecessary.
`custom1`			If type = "file", "path", or "custom": Optional. Otherwise: unnecessary.

Specifying this attribute is required, optional or unnecessary (blank):	For this `action` attribute value:		
	`purge`	`delete`	`update` or `refresh`
`custom2`			If type = "file", "path", or "custom": Optional. Otherwise: unnecessary.
`URLPath`			If type = "file" or "path": Optional. Otherwise: unnecessary.
`extensions`			If type = "path": Optional. Otherwise: unnecessary.
`query`		If type = "file", "path": Optional. If type = "custom": Required.	If type = "file", "path": Optional. If type = "custom": Required.
`recurse`			If type = "path": Optional. Otherwise: unnecessary.
`language`		If type = "file", "path", or "custom": Optional.	If type = "file", "path", or "custom": Optional.

For all `action` values of this tag except `update`, use the `cflock` tag to protect the collection during tag execution.

For information on the file types you can use with the Verity search engine, see Article 22492, *ColdFusion Server (versions 4.5 and higher): Supported File Types for Verity*, on the Macromedia ColdFusion Support Center, at http://www.macromedia.com/support/coldfusion/.

Example

```
<!--- for ACTION=UPDATE ------------------------------------------------>
<!--- for ACTION=UPDATE, #1 (TYPE=FILE) (key is a filename) ---->
<cfindex
     collection="snippets"
     action="update"
     type="file"
     key="c:\inetpub\wwwroot\cfdocs\snippets\abs.cfm"
     urlpath="http://localhost/cfdocs/snippets"
     custom1="custom1"
     custom2="custom2" >

<!--- for ACTION=UPDATE, #2 (TYPE=FILE) (key is a query result set column) ---->
<cfquery name="bookquery"
     datasource="book">
     select *from book where bookid='file'
</cfquery>
<cfoutput
     query="bookquery">
     --#url#,#description#-- <br>
```

```
</cfoutput>
<cfindex
     collection="snippets"
     action="update"
     type="file"
     query="bookquery"
     key="description"
     urlpath="url">

<!--- for ACTION=UPDATE, #3 (TYPE=PATH) (extensions .htm, .html,.cfm,.cfml) --->
<cfindex collection="snippets"
     action="update"
     type="path"
     key="c:\inetpub\wwwroot\cfdocs\snippets"
     urlpath="http://localhost/cfdocs/snippets"
     custom1="custom1"
     custom2="custom2"
     recurse="no"
     extensions=".htm, .html, .cfm, .cfml" >

<!--- for ACTION=UPDATE, #4 (TYPE=PATH)
     (extensions are files with no extension) ---->
<cfindex
     collection="snippets"
     action="update"
     type="path"
     key="c:\inetpub\wwwroot\cfdocs\snippets"
     urlpath="http://localhost/cfdocs/snippets"
     custom1="custom1"
     custom2="custom2"
     recurse="no"
     extensions="*." >

<!--- for ACTION=UPDATE, #5 (TYPE=PATH)
     (extensions are files with any extension) ---->
<cfindex
     collection="snippets"
     action="update"
     type="path"
     key="c:\inetpub\wwwroot\cfdocs\snippets"
     urlpath="http://localhost/cfdocs/snippets"
     custom1="custom1"
     custom2="custom2"
     recurse="no"
     extensions=".*">

<!--- for ACTION=UPDATE, #6 (TYPE=PATH) (where the key
     is a query result set column) ---->
<cfquery name="bookquery"
     datasource="book">
     select * from book where bookid='path1' or bookid='path2'
</cfquery>
<cfoutput
     query="bookquery">
     --#url#,#description#-- <br>
</cfoutput>
```

```
<cfindex
     collection="snippets"
     action="update"
     type="path"
     query="bookquery"
     key="description"
     urlpath="url" >

<!--- for ACTION=UPDATE, #7 (TYPE=CUSTOM) ---->
<cfquery name="book"
     datasource="book">
     select * from book
</cfquery>
<cfindex
     collection="custom_book"
     action="update"
     type="custom"
     body="description"
     key="bookid"
     query="book">

<!--- for ACTION=REFRESH------------------------------------------------>
<!--- ACTION=REFRESH, #1 (TYPE=FILE) ---->
<cflock name="verity"
     timeout="60">
<cfindex
     collection="snippets"
     action="Refresh"
     type="file"
     key="c:\inetpub\wwwroot\cfdocs\snippets\abs.cfm"
     urlpath="http://localhost/"
     custom1="custom1"
     custom2="custom2" >
</cflock>

<!--- ACTION=REFRESH, #2 (TYPE=PATH) ---->
<cflock name="verity"
     timeout="60">
<cfindex
     collection="snippets"
     action="refresh"
     type="path"
     key="c:\inetpub\wwwroot\cfdocs\snippets"
     urlpath="http://localhost/cfdocs/snippets/"
     custom1="custom1"
     custom2="custom2"
     recurse="yes"
     extensions=".htm,.html,.cfm,.cfml" >
</cflock>

<!--- ACTION=REFRESH, #3 (TYPE=CUSTOM) ---->
<cfquery name="book"
     datasource="book">
     select * from book
</cfquery>
```

```
<cfindex
     collection="custom_book"
     action="refresh"
     type="custom"
     body="description"
     key="bookid"
     query="book">

<!--- for ACTION=DELETE------------------------------------------------>

<!--- ACTION=DELETE, #1 (TYPE=FILE) ---->
<cflock name="verity"
     timeout="60">
<cfindex
     collection="snippets"
     action="delete"
     key="c:\inetpub\wwwroot\cfdocs\snippets\abs.cfm" >
</cflock>

<!--- ACTION=DELETE, #2 (TYPE=FILE) (the key is a query result set column) ---->
<cflock name="verity"
     timeout="60">
<cfquery name="book"
     datasource="book">
     select * from book where bookid='file'
</cfquery>
<cfoutput
     query="book">
     --#description#-- <br>
</cfoutput>
<cfindex
     collection="snippets"
     action="delete"
     type="file"
     query="book"
     key="description" >
</cflock>

<!--- ACTION=DELETE, #3 (TYPE=PATH) ---->
<cflock name="verity"
     timeout="60">
<cfindex
     collection="snippets"
     action="delete"
     type="path"
     key="c:\inetpub\wwwroot\cfdocs\snippets"
     extensions=".cfm"
     recurse="no">
</cflock>

<!--- ACTION=DELETE, #4 (TYPE=PATH) (key is a query result set column) ---->
<cflock name="verity"
     timeout="60">
<cfquery
     name="bookquery"
     datasource="book">
```

```
      select * from book where bookid='path1'
         </cfquery>
<cfoutput
      query="bookquery">
      --#url#,#description#-- <br>
</cfoutput>
<cfindex
      collection="snippets"
      action="delete"
      type="path"
      query="bookquery"
      key="description" >
</cflock>

<!--- ACTION=DELETE, #5 (TYPE=CUSTOM) ---->
<cflock name="verity"
      timeout="60">
<cfquery name="book"
      datasource="book">
      select * from book where bookid='bookid1'
</cfquery>
<cfindex
      collection="custom_book"
      action="delete"
      type="custom"
      query="book"
      key="bookid" >
</cflock>

<!--- for ACTION=PURGE------------------------------------------------>
<cflock name="verity"
      timeout="60">
<cfindex
      action="purge"
      collection="snippets">
</cflock>
```

cfinput

Description
Used within the cfform tag, to place radio buttons, check boxes, or text boxes on a form. Provides input validation for the specified control type.

Category
Forms tags

Syntax

```
<cfinput
   type = "input_type"
   name = "name"
   value = "initial_value"
   required = "Yes" or "No"
   range = "min_value, max_value"
   validate = "data_type"
   onValidate = "javascript_function"
   pattern = "regexp"
   message = "validation_msg"
   onError = "text"
   size = "integer"
   maxLength = "integer"
   checked
   passThrough = "HTML_attributes">
```

See also
cfapplet, cfform, cfgrid, cfselect, cfslider, cftextinput, cftree

History
New in ColdFusion MX: if the cfform tag preserveData attribute is set to True, ColdFusion checks radio and checkbox values only if their value matches the posted value for the control. (In earlier releases, if the posted value did not match any of the cfinput checkboxes or radio buttons for the control, the checked attribute was used.

Attributes

Attribute	Req/Opt	Default	Description
type	Optional	text	• text: creates a text entry box control • radio: creates a radio button control • checkbox: creates a checkbox control • password: creates a password entry control
name	Required		Name for form input element.
value	Optional		Initial value for form input element.
required	Optional	No	• Yes • No
range	Optional		Minimum and maximum value range, separated by a comma. If type = "text" or "password", this applies only to numeric data.

Attribute	Req/Opt	Default	Description
validate	Optional		Verifies a value's format: • date: US date `mm/dd/yyyy` • eurodate: European date `dd/mm/yyyy` • time: time `hh:mm:ss` • float: floating point entry • integer: integer entry • telephone: telephone ###-###-####. Separator: hyphen or blank. Area code and exchange must begin with a digit 1–9. • zipcode: (U.S. formats only) 5-digit ##### or 9-digit #####-####. Separator: hyphen or blank. • creditcard: strips blanks and dashes; uses the mod10 algorithm. • social_security_number: ###-##-####. Separator: hyphen or blank. • regular_expression: matches input against regular expression specified by the `pattern` attribute.
onValidate	Optional		Custom JavaScript function to validate user input. The form object, input object, and input object values are passed to the routine, which should return True if validation succeeds, and False otherwise. If used, the `validate` attribute is ignored.
pattern	Required if `validate = "regular_expression"`		JavaScript regular expression pattern to validate input. Omit leading and trailing slashes. For examples and syntax, see the "Building Dynamic Forms" chapter in Developing ColdFusion Applications.
message	Optional		Message text to display if validation fails.
onError	Optional		Custom JavaScript function to execute if validation fails.
size	Optional		Size of input control. Ignored, if `type = "radio"` or `"checkbox"`.
maxLength	Optional		Maximum length of text entered, if `type = "Text"` or `"password"`.
checked	Optional		Selects a control. No value is required. Applies if `type = "radio"` or `"checkbox"`. Optional: you can enter the following values: • true (equivalent to `checked`) • false (equivalent to omitting the attribute)
passThrough	Optional		HTML attributes that are not supported by `cfinput`. If you specify an attribute and value, they are passed to the HTML code generated for the tag.

Usage

If the `cfform preserveData` attribute is true and the form posts back to the same page, the posted value of the cfinput control is used, instead of its Value or Checked attribute.

If `cfinput` checkbox or radio type values match the submitted value for the control, ColdFusion checks their values. If no value matches, nothing is checked.

To add other HTML <input> tag attributes and values to this tag, use the `passThrough` attribute. They are passed through ColdFusion to the browser when creating a form. The supported HTML attributes are: CLASS, ID, MAXLENGTH, MESSAGE, ONBLUR, ONCHANGE, ONCLICK, ONDBLCLICK, ONFOCUS, SIZE, STYLE, and TABINDEX.

If you specify a value in quotation marks, you must escape them; for example,

```
passThrough = "readonly = " "YES " " "
```

For more information, see cfform on page 107. For information on using JavaScript regular expressions with this tag, see *Developing ColdFusion MX Applications with CFML*.

Example

```
<!--- this example shows the use of cfinput within a cfform to ensure simple
         validation of text items --->
<cfform action = "cfinput.cfm">
<!--- phone number validation --->
Phone Number Validation (enter a properly formatted phone number): <br>
<cfinput
   type = "Text" name = "MyPhone"
   message = "Enter telephone number,formatted xxx-xxx-xxxx (e.g. 617-761-2000)"
   validate = "telephone" required = "Yes">
   <font size = -1 color = red>Required</font>
<!--- zip code validation --->
<p>Zip Code Validation (enter a properly formatted zip code):<br>
<cfinput
   type = "Text" name = "MyZip"
   message = "Enter zip code, formatted xxxxx or xxxxx-xxxx"
   validate = "zipcode" required = "Yes">
   <font size = -1 color = red>Required</font>
<!--- range validation --->
<p>Range Validation (enter an integer from 1 to 5): <br>
<cfinput
   type = "Text" name = "MyRange" range = "1,5"
   message = "You must enter an integer from 1 to 5"
   validate = "integer" required = "No">
<!--- date validation --->
<p>Date Validation (enter a properly formatted date):<br>
<cfinput
   type = "Text" name = "MyDate"
   message = "Enter a correctly formatted date (dd/mm/yy)"
   validate = "date" required = "No">
<input
   type = "Submit" name = ""
   value = "send my information">
</cfform>
```

cfinsert

Description Inserts records in data sources from data in a ColdFusion form or form Scope.

Category Database manipulation tags

Syntax
```
<cfinsert
   dataSource = "ds_name"
   tableName = "tbl_name"
   tableOwner = "owner"
   tableQualifier = "tbl_qualifier"
   username = "username"
   password = "password"
   formFields = "formfield1, formfield2, ...">
```

See also cfprocparam, cfprocresult, cfquery, cfqueryparam, cfstoredproc, cftransaction, cfupdate

History New in ColdFusion MX: the connectString, dbName, dbServer, dbtype, provider and providerDSN attributes are deprecated. Do not use them. They do not work, and might cause an error, in releases later than ColdFusion 5.

Attributes

Attribute	Req/Opt	Default	Description
dataSource	Required		Data source; contains table.
tableName	Required		Table in which to insert form fields. ORACLE drivers: must be uppercase. Sybase driver:case-sensitive. Must be the same case used when table was created
tableOwner	Optional		For data sources that support table ownership (such as SQL Server, Oracle, and Sybase SQL Anywhere), use this field to specify the owner of the table.
tableQualifier	Optional		For data sources that support table qualifiers, use this field to specify qualifier for table. The purpose of table qualifiers varies among drivers. For SQL Server and Oracle, qualifier refers to name of database that contains table. For Intersolv dBASE driver, qualifier refers to directory where DBF files are located.
username	Optional		Overrides username specified in ODBC setup.
password	Optional		Overrides password specified in ODBC setup.
formFields	Optional	(all on form, except keys)	Comma-delimited list of form fields to insert. If not specified, all fields in the form are included. If a form field is not matched by a column name in the database, ColdFusion throws an error. The database table key field must be present in the form. It may be hidden.

Example

```
<!--- This shows how to use cfinsert instead of cfquery to put
         data in a datasource. --->
<!--- if form.POSTED exists, we insert new record, so begin cfinsert tag --->
<cfif IsDefined ("form.posted")>
   <cfinsert dataSource = "cfsnippets"
      tableName = "Comments"
      formFields = "Email,FromUser,Subject,MessText,Posted">
   <h3><I>Your record was added to the database.</i></h3>
</cfif>

<cfif IsDefined ("form.posted")>
   <cfif Server.OS.Name IS "Windows NT">
      <cfinsert datasource="cfsnippets" tablename="Comments"
        formfields="EMail,FromUser,Subject,MessText,Posted">
   <cfelse>
      <cfinsert datasource="cfsnippets" tablename="Comments"
        formfields="CommentID,EMail,FromUser,Subject,MessText,Posted">
   </cfif>
   <h3><i>Your record was added to the database.</i></h3> </cfif>

<!--- use a query to show the existing state of the database --->
<cfquery name = "GetComments" dataSource = "cfsnippets">
   SELECT
      CommentID, EMail, FromUser, Subject, CommtType, MessText, Posted, Processed
   FROM
      Comments
</cfquery>

<html>
<head></head>
<h3>cfinsert Example</h3>
<p>First, show a list of the comments in the cfsnippets datasource.
<!--- show all the comments in the db --->
<table>
   <tr>
      <td>From User</td><td>Subject</td><td>Comment Type</td>
      <td>Message</td><td>Date Posted</td>
   </tr>
   <cfoutput query = "GetComments">
      <tr>
         <td valign = top><a href = "mailto:#Email#">#FromUser#</A></td>
         <td valign = top>#Subject#</td>
         <td valign = top>#CommtType#</td>
         <td valign = top><font size = "-2">#Left(MessText, 125)#</font></td>
         <td valign = top>#Posted#</td>
      </tr>
   </cfoutput>
</table>
<p>Next, we'll offer the opportunity to enter a comment:
<!--- make a form for input --->
<form action = "cfinsert.cfm" method = "post">
   <pre>
   Email: <input type = "Text" name = "email">
   From: <input type = "Text" name = "fromUser">
   Subject:<input type = "Text" name = "subject">
```

```
   Message:<textarea name = "MessText" COLS = "40" ROWS = "6"></textarea>
   Date Posted: <cfoutput>#DateFormat(Now())#</cfoutput>
   <!--- dynamically determine today's date --->
   <input type = "hidden"
      name = "posted" value = "<cfoutput>#Now()#</cfoutput>">
   </pre>
   <input type = "Submit"
      name = "" value = "insert my comment">
</form>
```

cfinvoke

Description

Invokes component methods from within a ColdFusion page or component. You use this tag to reference a WSDL file and consume a web service from within a block of CFML code.

This tag works as follows:

- Instantiates a component or web service and invokes a method on it
- Invokes a method on an instantiated component or web service

This tag can pass parameters to a method in the following ways:

- With the `cfinvokeargument` tag
- As named attribute-value pairs, one attribute per parameter
- As a structure, in the `argumentCollection` attribute

Category

Extensibility tags

Syntax

```
<!--- Syntax 1 - this syntax invokes a method of a component --->
<cfinvoke
   component = "component name or reference"
   method = "method name"
   returnVariable = "variable name"
   argumentCollection = "argument collection"
   ...>
OR
<!--- Syntax 2 - this syntax can invoke a method of a component only
   from within the component. --->
<cfinvoke
   method = "method name"
   returnVariable = "variable name"
   argumentCollection = "argument collection"
   ...
>
OR
<!--- Syntax 3 - this syntax invokes a web service --->
<cfinvoke
   webservice = "URLtoWSDL_location"
   method = "operation_name"
   username = user name"
   password = "password"
   inputParam1 = "value1"
   inputParam2 = "value2"
   ...
   returnVariable = "var_name"
   ...>
OR
<!--- Syntax 4A - this syntax invokes a component.
This syntax shows instantiation with the cfobject tag.
This cfinvoke syntax applies to instantiating a component
with the cfobject tag and to instantiating a component
with the createobject function. --->
<cfobject
   component = "component name"
   name = "mystringname for instantiated object">
```

```
<cfinvoke
   <!--- value is object name, within pound signs --->
   component = "#mystringname for instantiated component#">
OR
<!--- Syntax 4B - this syntax invokes a web service.
This syntax shows instantiation with the cfobject tag.
This cfinvoke syntax applies to instantiating a web service
with the cfobject tag and to instantiating a web service
with the createobject function. --->
<cfobject
   webservice = "web service name"
   name = "mystringname for instantiated object"
   method = "operation_name">
<cfinvoke
   <!--- value is object name, within pound signs --->
   webservice = "#my stringname for instantiated web service#" >
```

See also `cfargument`, `cfcomponent`, `cffunction`, `cfinvokeargument`, `cfobject`, `cfproperty`, `cfreturn`

History New in ColdFusion MX: This tag is new.

Attributes

Attribute	Req/Opt	Default	Description
component	See Usage section		String or component object; a reference to a component, or component to instantiate.
method	See Usage section		Name of a method. For a web service, the name of an operation.
returnVariable	Optional		Name of a variable for the invocation result.
argumentCollection	Optional		Name of a structure; associative array of arguments to pass to the method.
username	Optional		Overrides username specified in Administrator > Web Services.
password	Optional		Overrides password specified in Administrator > Web Services.
webservice	Required		The URL of the WSDL file for the web service.
input_params ...			Input parameters.

Usage The following table shows which attributes of this tag are required:

Specifying this attribute is required, optional or unnecessary (blank):	For this `cfinvoke` tag syntax:				
	`Syntax 1`	`Syntax 2`	`Syntax 3`	`Syntax 4A`	`Syntax 4B`
component	Required	Optional		Required	
method	Required	Required	Required		Required
returnVariable	Optional	Optional	Optional		Optional
argumentCollection	Optional	Optional	Optional		Optional
username			Optional		Optional
password			Optional		Optional
webservice			Required		Required
input_params ...	Optional	Optional	Optional		Optional

If the `component` attribute specifies a component name, the component with the corresponding name is instantiated, the requested method is invoked, and then the component instance is immediately destroyed. If the attribute contains a reference to an instantiated component object, no instantiation or destruction of the component occurs.

Method arguments can be passed in one of the following ways (and, if an argument is passed in more than one way with the same name, this order of precedence applies):

1 Using the `cfinvokeargument` tag

2 Passing directly as attributes of the `cfinvoke` tag (they cannot have the same name as a registered `cfinvoke` attribute: `method`, `component`, `webservice`, `returnVariable`)

3 Passing as struct keys, using the `argumentCollection` attribute

For example, the `params` struct contains three keys: `a=1`, `b=1`, `c=1`. The following call is evaluated as if the arguments were passed to the method in the order `a=3`, `b=2`, `c=1`:

```
<cfinvoke ... a=2 b=2 argumentCollection=params>
     <cfinvokeargument name="a" value="3">
  </cfinvoke>
```

Note: The following `cfinvoke` tag attribute names are reserved; they cannot be used for parameter names: `component`, `method`, `argumentCollection`, and `result`.

Example 1 This example uses Syntax 1.

```
<!--- immediate instantiation and destruction --->
<cfinvoke
   component="nasdaq.quote"
   method="getLastTradePrice"
   returnVariable="res">
   <cfinvokeargument
      name="symbol"
      value="macr">
</cfinvoke>
<cfoutput>#res#</cfoutput>
```

Example 2 This example uses Syntax 1.

```
<!--- passing the arguments using argumentCollection --->
<cfset args = StructNew()>
<cfset args.symbol = "macr">
<cfinvoke
   component="nasdaq.quote"
   method="getLastTradePrice"
   argumentCollection="#args#"
   returnVariable="res">
<cfoutput>#res#</cfoutput>
```

Example 3 This example uses Syntax 2.

```
<!--- called only from within a component, MyComponent--->
<cfinvoke
   method = "a method name of MyComponent"
   returnVariable = "variable name">
```

Example 4 This example uses Syntax 3.

```
<!--- using cfinvoke to consume a web service using a ColdFusion component --->
<!--- put the following code in a ColdFusion page named wscfml.cfm:--->
<cfinvoke
   webservice='http://www.xmethods.net/sd/2001/BabelFishService.wsdl'
   method='BabelFish'
   translationmode="en_es"
   sourcedata="Hello world, friend"
   returnVariable='foo'>
<cfoutput>#foo#</cfoutput>
```

For more information on the BabelFish web service example, see *Developing ColdFusion MX Applications with CFML*.

Example 5 This example uses Syntax 4A.

```
<!--- separate instantiation and method invocation; useful for
   multiple invocations using different methods or values--->
<cfobject
   name="quoteService"
   component="nasdaq.quote">
<cfinvoke
   component="#quoteService#"
   method="getLastTradePrice"
   symbol="macr"
   returnVariable="res_macr">
<cfoutput>#res#</cfoutput>
<cfinvoke
   component="#quoteService#"
   method="getLastTradePrice"
   symbol="mot"
   returnVariable="res_mot">
<cfoutput>#res#</cfoutput>
```

cfinvokeargument

Description Passes the name and value of a parameter to a component method or a web service. This tag is used within the cfinvoke tag.

Category Extensibility tags

Syntax
```
<cfinvokeargument
   name="argument name"
   value="argument value">
```

See also cfargument, cfcomponent, cffunction, cfinvoke, cfobject, cfproperty, cfreturn

History New in ColdFusion MX: This tag is new.

Attributes

Attribute	Req/Opt	Default	Description
name	Required		Argument name
value	Required		Argument value

Usage You can code multiple name/value pairs within this tag.

Example 1
```
<cfinvoke
   component="nasdaq.quote"
   method="getLastTradePrice"
   returnVariable="res">
   <cfinvokeargument
      name="symbol" value="mot">
   <cfinvokeargument
      name="symbol" value="macr">
</cfinvoke>

<cfoutput>#res#</cfoutput>
```

Example 2
```
<cfinvoke
   webservice ="http://www.xmethods.net/sd/2001/BabelFishService.wsdl"
   method ="BabelFish"
   returnVariable = "varName"
   >
   <cfinvokeargument
      name="translationmode" value="en_es">
   <cfinvokeargument
      name="sourcedata" value="Hello world, friend">
</cfinvoke>
<cfoutput>#varName#</cfoutput>
```

cfldap

Description Provides an interface to a Lightweight Directory Access Protocol (LDAP) directory server, such as the Netscape Directory Server.

Category Forms tags, Internet Protocol tags

Syntax
```
<cfldap
   server = "server_name"
   port = "port_number"
   username = "name"
   password = "password"
   action = "action"
   name = "name"
   timeout = "seconds"
   maxRows = "number"
   start = "distinguished_name"
   scope = "scope"
   attributes = "attribute, attribute"
   filter = "filter"
   sort = "attribute[, attribute]..."
   sortControl = "nocase" and/or "desc" or "asc"
   dn = "distinguished_name"
   startRow = "row_number"
   modifyType = "replace" or "add" or "delete"
   rebind = "Yes" or "No"
   referral = "number_of_allowed_hops"
   secure = "multi_field_security_string"
   separator = "separator_character"
   delimiter = "delimiter_character">
```

See also `cfftp`, `cfhttp`, `cfmail`, `cfmailparam`, `cfpop`

History New in ColdFusion MX:

- This tag validates the query name in the `name` attribute.
- This tag does not support client-side sorting of query results. (It supports server-side sorting; use the `sort` and `sortcontrol` attributes.)
- Server-side sorting results might be sorted slightly differently than in ColdFusion 5. If you attempt a sort against a server that does not support it, ColdFusion MX throws an error.
- The `filterconfig` and `filterfile` attributes are deprecated. Do not use them in new applications. They might not work, and might cause an error, in later releases. If they are used, this tag throws an exception.

Attributes

Attribute	Req/Opt	Default	Description
server	Required		Host name or IP address of LDAP server.
port	Optional	389	Port
username	Required if `secure = "CFSSL_BASIC"`	(anonymous)	User ID

Attribute	Req/Opt	Default	Description
password	Required if `secure = "CFSSL_BASIC"`		Password that corresponds to user name. If `secure = "CFSSL_BASIC"`, V2 encrypts the password before transmission.
action	Optional	query	• `query`: returns LDAP entry information only. Requires `name`, `start`, and `attributes` attributes. • `add`: adds LDAP entries to LDAP server. Requires `attributes` attribute. • `modify`: modifies LDAP entries, except distinguished name `dn` attribute, on LDAP server. Requires `dn`. See `modifyType` attribute. • `modifyDN`: modifies distinguished name attribute for LDAP entries on LDAP server. Requires `dn`. • `delete`: deletes LDAP entries on an LDAP server. Requires `dn`.
name	Required if action = "Query"		Name of LDAP query. The tag validates the value.
timeout	Optional	60	Maximum length of time, in seconds, to wait for LDAP processing.
maxRows	Optional		Maximum number of entries for LDAP queries.
start	Required if action = "Query"		Distinguished name of entry to be used to start a search.
scope	Optional	oneLevel	Scope of search, from entry specified in `start` attribute for `action = "Query"`. • oneLevel: entries one level below entry. • base: only the entry. • subtree: entry and all levels below it.
attributes	Required if `action = "Query"`, `"Add"`, `"ModifyDN"`, or `"Modify"`		For queries: comma-delimited list of attributes to return. For queries, to get all attributes, specify `"*"`. If `action = "add"` or `"modify"`, you can specify a list of update columns. Separate attributes with a semicolon. If `action = "ModifyDN"`, ColdFusion passes attributes to the LDAP server without syntax checking.
filter	Optional	"objectclass = *"	Search criteria for `action = "query"`. List attributes in the form: `"(attribute operator value)"` Example: `"(sn = Smith)"`
sort	Optional		Attribute(s) by which to sort query results. Use a comma delimiter.

Attribute	Req/Opt	Default	Description
sortControl	Optional	asc	• nocase: case-insensitive sort • asc: ascending (a to z) case-sensitive sort • desc: descending (z to a) case-sensitive sort You can enter a combination of sort types; for example, `sortControl = "nocase, asc"`.
dn	Required if `action = "Add", "Modify", "ModifyDN", or "delete"`		Distinguished name, for `update` action. Example: `"cn = Bob Jensen, o = Ace Industry, c = US"`
startRow	Optional	1	Used with `action = "query"`. First row of LDAP query to insert into a ColdFusion query.
modifyType	Optional	replace	How to process an attribute in a multi-value list. • add: appends it to any attributes • delete: deletes it from the set of attributes • replace: replaces it with specified attributes You cannot add an attribute that is already present or that is empty.
rebind	Optional	No	• Yes: attempt to rebind referral callback and reissue query by referred address using original credentials. • No: referred connections are anonymous
referral	Optional		Integer. Number of hops allowed in a referral. A value of 0 disables referred addresses for LDAP; no data is returned.
secure	Optional		Security to employ, and required information. One option: • CFSSL_BASIC; certificate_db "CFSSL_BASIC" provides V2 SSL encryption and server authentication -certificate_db: certificate database file (Netscape cert7.db format). Absolute path or simple filename. See the Usage section.

Attribute	Req/Opt	Default	Description
separator	Optional	, [comma]	Delimiter to separate attribute values of multi-value attributes. Used by `query`, `add`, and `modify` actions, and by `cfldap` to output multi-value attributes. For example, if `$` (dollar sign), the `attributes` attribute could be `"objectclass = top$person"`, where the first value of objectclass is `"top"`, and the second value is `"person"`. This avoids confusion if values include commas.
delimiter	Optional		Separator for attribute name-value pairs, if: • the `attributes` attribute specifies more than one item • an attribute has the delimiter semicolon. For example: `mgrpmsgrejecttext;lang-en` Used by `query`, `add`, and `modify` actions, and by `cfldap` to output multi-value attributes. For example, if `$` (dollar sign), you could specify this list of pairs with attributes: `"cn = Double Tree Inn$street = 1111 Elm;Suite 100`

Usage

If you use the `query` action, `cfldap` creates a query object, allowing access to information in the query variables, as follows:

Variable name	Description
queryname.recordCount	Number of records returned by query
queryname.currentRow	Current row of query that `cfoutput` is processing
queryname.columnList	Column names in query

To use the `security = "CFSSL_BASIC"` option, you must copy the cert7.db and/or key3.db key files to the default directory location of a user-installed LDAP directory; on Windows, the directory is C:\cfusion\ldap.

The security certificate encrypts conversation. The server always sends a digital certificate to confirm the server.

Characters that are illegal in ColdFusion can be used in LDAP attribute names. As a result, the `cfldap` tag could create columns in the query result set whose names contain illegal characters and are, therefore, inaccessible in CFML. In ColdFusion, illegal characters are automatically mapped to the underscore character; therefore, column names in the query result set might not exactly match the names of the LDAP attributes.

For usage examples, see *Developing ColdFusion MX Applications with CFML*.

Example

```
<h3>cfldap Example</h3>
<p>Provides an interface to LDAP directory servers like BigFoot
        (<a href = "http://www.bigfoot.com">http://www.bigfoot.com</A>).
<p>Enter a name (try your own name) and search a public LDAP resource.
<!--- If the server has been defined, run the query --->
<cfif IsDefined("form.server")>
  <!--- check to see that there is a name listed --->
  <cfif form.name is not "">
    <!--- make the LDAP query --->
    <cfldap  server = "ldap.bigfoot.com"
        action = "query"
        name = "results"
        start = "cn = #name#,c = US"
        filter = "(cn = #name#)"
        attributes = "cn,o,l,st,c,mail,telephonenumber"
        sort = "cn ASC">
    <!--- Display results --->
    <center>
    <table border = 0 cellspacing = 2 cellpadding = 2>
      <tr>
        <th colspan = 5>
          <cfoutput>#results.recordCount# matches found
          </cfoutput></TH>
      </tr>
      <tr>
        <th><font size = "-2">Name</font></TH>
        <th><font size = "-2">Organization</font></TH>
        <th><font size = "-2">Location</font></TH>
        <th><font size = "-2">E-Mail</font></TH>
        <th><font size = "-2">Phone</font></TH>
      </tr>
      <cfoutput query = "results">
        <tr>
          <td><font size = "-2">#cn#</font></td>
          <td><font size = "-2">#o#</font></td>
          <td><font size = "-2">#l#, #st#, #c#</font></td>
          <td><font size = "-2">
            <A href = "mailto:#mail#">#mail#</A></font></td>
          <td><font size = "-2">#telephonenumber#</font></td>
        </tr>
      </cfoutput>
      </table>
      </center>
    </cfif>
</cfif>
```

cflocation

Description: Stops page execution and opens a ColdFusion page or HTML file.

Category: Flow-control tags, Page processing tags

Syntax:

```
<cflocation
   url = "url"
   addToken = "Yes" or "No">
```

See also: cfabort, cfbreak, cfexecute, cfexit, cfif, cfloop, cfswitch, cfthrow, cftry

Attributes

Attribute	Req/Opt	Default	Description
url	Required		URL of HTML file or CFML page to open.
addToken	Optional		clientManagement must be enabled (see cfapplication). • Yes: appends client variable information to URL • No

Usage: You might write a standard message or response in a file, and call it from several applications. You could use this tag to redirect the user's browser to the standard file.

If you use this tag after the cfflush tag on a page, an error is thrown.

Warning: Do not set a cookie variable on a page on which you use this tag. If you do, the cookie is not saved on the browser. Similarly, if you use a cookie to store a client variable, the client variable is not set.

Example:

```
<!--- This view-only example shows the use of cflocation --->
<h3>cflocation Example</h3>
<p>This tag redirects the browser to a web resource; normally, you would
         use this tag to go to a CF page or an HTML file on the same server.
         The addToken attribute lets you send client information to the
         target page.
<p>This code redirects you to CFDOCS home page (if you remove the comment
         marks, this information displays in frame):
<!--- <cflocation url = "../cfdocs/index.htm" addToken = "No"> --->
```

cflock

Description

Ensures the integrity of shared data. Instantiates the following kinds of locks:

- **Exclusive**: allows single-thread access to the CFML constructs in its body. The tag body can be executed by one request at a time. No other requests can start executing within the tag while a request has an exclusive lock. ColdFusion issues exclusive locks on a first-come, first-served basis.
- **Read-only**: allows multiple requests to access CFML constructs within the tag body concurrently. Use a read-only lock only when shared data is read and not modified. If another request has an exclusive lock on shared data, the new request waits for the exclusive lock to be released.

Category

Application framework tags

Syntax

```
<cflock
   timeout = "timeout in seconds "
   scope = "Application" or "Server" or "Session"
   name = "lockname"
   throwOnTimeout = "Yes" or "No"
   type = "readOnly" or "exclusive ">
   <!--- CFML to be synchronized --->
</cflock>
```

See also

`cfapplication`, `cfassociate`, `cfmodule`

Attributes

Attribute	Req/Opt	Default	Description
timeout	Required		Maximum length of time, in seconds, to wait to obtain a lock. If lock obtained, tag execution continues. Otherwise, behavior depends on `throwOnTimeout` attribute value.
scope	Optional		Mutually exclusive with the `name` attribute. • Application • Server • Session
name	Optional		Lock name. Only one request can execute within this tag with a given name. Permits synchronizing access to resources from different parts of an application. Lock names are global to a ColdFusion server. They are shared among applications and user sessions, but not clustered servers. Mutually exclusive with the `scope` attribute. Cannot be an empty string.
throwOnTimeout	Optional	Yes	How timeout conditions are handled. • Yes: exception is generated for the timeout. • No: execution continues past this tag.
type	Optional	Exclusive	• read-only: lets more than one request read shared data. • exclusive: lets one request read or write shared data.

Note: Limit the scope of code that updates shared data structures, files, and CFXs. Exclusive locks are required to ensure the integrity of updates, but read-only locks are faster. In a performance-sensitive application, substitute read-only locks for exclusive locks where possible; for example, when reading shared data.

Usage

ColdFusion Server is a multithreaded server; it can process multiple page requests at a time. Use the `cflock` tag for these purposes:

- To ensure that modifications to shared data and objects made in concurrently executing requests occur sequentially.
- Around file manipulation constructs, to ensure that file updates do not fail because files are open for writing by other applications or tags.
- Around CFX invocations, to ensure that ColdFusion can safely invoke CFXs that are not implemented in a thread-safe manner. (This applies only to CFXs developed in C++ using the CFAPI.)

To work safely with ColdFusion, a C++ CFX that maintains and manipulates shared (global) data structures must be made thread-safe; however, this requires advanced knowledge. You can use a CFML custom tag wrapper around a CFX to make its invocation thread-safe.

Scope

When you display, set, or update variables in a shared scope, use the `scope` attribute to identify the scope as Server, Application or Session.

In the ColdFusion Administrator, Server section, the Locking page sets locking options according to scope. The following features are available for Server, Application, and Session scope:

	Availability		
Feature	**Server**	**Application**	**Session**
No automatic checking or locking	Yes	Yes	Yes
Full checking	Yes	Yes	Yes
Automatic read locking	Yes	Yes	Yes
Single threaded sessions			Yes

These feature have the following tradeoffs:

- **No automatic checking or locking**: Reads and writes are not locked nor checked for correct protection. Select this only if you have run with full checking and know that there are no errors to handle, and that all locking is handled programmatically. This provides the fastest performance.
- **Full checking**: All unlocked accesses are detected. Select this when in debug mode. This slows performance.
- **Automatic read locking**: Reads are locked. Unlocked writes cause an error. This feature slows performance considerably.
- **Single-threaded sessions**: The request must finish before another request for the same session is processed. This feature might slow performance, depending on the request pattern. For example, the total response time might increase if an application has multiple frames that can be refreshed at once, causing multiple requests to queue.

For more information, see *Administering ColdFusion MX*.

If you create a lock with the `name` attribute, and enable full lock checking in the ColdFusion Administrator, ColdFusion returns an error.

Do not specify full checking for a lock in its own scope; for example, if the lock is in the Application scope, do not specify full checking for the Application scope.

Deadlocks

A deadlock is a state in which no request can execute the locked section of a page. Once a deadlock occurs, neither user can break it, because all requests to the protected section of the page are blocked until the deadlock can be resolved by a lock timeout.

The `cflock` tag uses kernel level synchronization objects that are released automatically upon timeout and/or the abnormal termination of the thread that owns them. Therefore, while processing a `cflock` tag, ColdFusion never deadlocks for an infinite period of time. However, very large timeouts can block request threads for long periods, and radically decrease throughput. To prevent this, always use the minimum timeout value.

Another cause of blocked request threads is inconsistent nesting of `cflock` tags and inconsistent naming of locks. If you nest locks, everyone accessing the locked variables must consistently nest `cflock` tags in the same order. Otherwise, a deadlock can occur.

These examples show situations that cause deadlocks:

Example deadlock with two users	
User 1	**User 2**
Locks the session scope.	Locks the Application scope.
Deadlock: Tries to lock the Application scope, but it is already locked by User 2.	Deadlock: Tries to lock the Session scope, but it is already locked by User 1.

The following deadlock could occur if you tried to nest a write lock after a read lock:

Example deadlock with one user
User 1
Locks the Session scope with a read lock.
Attempts to lock the Session scope with an exclusive lock.
Deadlock: Attempts to lock the Session scope with an exclusive lock, but cannot because the scope is already locked for reading.

The following code shows this scenario:

```
<cflock timeout = "60" scope = "SESSION" type = "readOnly">
   ..............
   <cflock timeout = "60" scope = "SESSION" type = "Exclusive">
   .........
   </cflock>
</cflock>
```

To avoid a deadlock, everyone who nests locks must do so in a well-specified order and name the locks consistently. If you must lock access to the Server, Application, and Session scopes, you must do so in this order:

1 Lock the Session scope. In the `cflock` tag, specify `scope = "session"`.

2 Lock the Application scope. In the `cflock` tag, specify `scope = "Application"`.

3 Lock the Server scope. In the `cflock` tag, specify `scope = "server"`.

4 Unlock the Server scope.

5 Unlock the Application scope.

6 Unlock the Session scope.

Note: If you do not have to lock a scope, you can skip any pair of these lock/unlock steps. For example, if you do not have to lock the Server scope, you can skip Steps 3 and 4. Similar rules apply for named locks.

For more information, see the following:

- *Developing ColdFusion MX Applications with CFML*
- Article #20370, *ColdFusion Locking Best Practices,* on the Macromedia website at http://www.macromedia.com/support/service/

Example

```
<!--- This example shows how cflock can guarantee consistency of data updates
         to variables in the Application, Server, and Session scopes. --->
<!----  Copy the following code into an Application.cfm file in the application
         root directory. ----->
<!------------------------------   beginning of Application.cfm code
   <h3>cfapplication Example</h3>
   <p>cfapplication defines scoping for a ColdFusion application and
         enables or disables storing of application and session variables.
         Put this tag in a special file called Application.cfm. It is run before
         any other CF page in its directory.</p>
   <cfapplication name = "ETurtle"
         sessionTimeout = #CreateTimeSpan(0,0, 0, 60)#
         sessionManagement = "Yes">
<!--- Initialize session and application variables used by E-Turtleneck.
         Use session scope for the session variables. --->
   <cflock scope = "Session"
      timeout = "30" type = "Exclusive">
      <cfif NOT IsDefined("session.size")>
        <cfset session.size = "">
      </cfif>
      <cfif NOT IsDefined("session.color")>
        <cfset session.color = "">
      </cfif>
   </cflock>
<!--- Use application lock for application variable. Variable keeps track
         of number of turtlenecks sold. The application lock should have
         the same name as specified in the cfapplication tag. ---->
   <cflock scope = "Application" timeout = "30" type = "Exclusive">
      <cfif NOT IsDefined("application.number")>
        <cfset application.number = 1>
      </cfif>
   </cflock>
```

```
   <cflock scope = "Application" timeout = "30" type = "readOnly">
      <cfoutput>
        E-Turtleneck has sold #application.number# turtlenecks to date.
      </cfoutput>
  </cflock>
-------------------- End of Application.cfm ---------------------->
<h3>cflock Example</h3>
<cfif IsDefined("form.submit")>
   <cfoutput>
      Thanks for shopping E-Turtleneck. You chose size <b>#form.size#</b>,
          color <b>#form.color#</b>. </cfoutput>
   <!--- Lock session variables to assign form values to them. ---->
   <cflock scope = "Session" timeout = "30" type = "Exclusive">
      <cfparam name = session.size Default = #form.size#>
      <cfparam name = session.color Default = #form.color#>
   </cflock>
   <!---- Lock variable application.number to find total turtlenecks sold. --->
   <cflock scope = "Application" timeout = "30" type = "Exclusive">
      <cfset application.number = application.number + 1>
   </cflock>
<cfelse><!--- Show the form only if it has not been submitted. --->
   <form action = "cflock.cfm">
   <p>Congratulations! You selected the most comfortable turtleneck in the world.
         Please select color and size.</p>
   <table cellspacing = "2" cellpadding = "2" border = "0">
      <tr>
        <td>Select a color.</td>
        <td><select type = "Text" name = "color">
             <option>red
             <option>white
             <option>blue
             <option>turquoise
             <option>black
             <option>forest green
           </select>
        </td>
      </tr>
      <tr>
        <td>Select a size.</td>
        <td><select type = "Text" name = "size" >
             <option>XXsmall
             <option>Xsmall
             <option>small
             <option>medium
             <option>large
             <option>Xlarge
           </select>
        </td>
      </tr>
      <tr>
        <td>Press Submit when you are finished making your selection.</td>
        <td><input type = "Submit" name = "submit" value = "Submit"> </td>
      </tr>
   </table>
   </form>
</cfif>
```

cflog

Description Writes a message to a log file.

Category Data output tags

Syntax
```
<cflog
   text = "text"
   log = "log type"
   file = "filename"
   type = "message type"
   application = "application name yes or no">
```

See also cfcol, cfcontent, cfoutput, cftable

History New in ColdFusion MX: The thread, date, and time attributes are deprecated. Do not use them in new applications. They might not work, and might cause an error, in later releases. (In earlier releases, these attributes determined whether the respective data items were output to the log. In ColdFusion MX, this data is always output.)

Attributes

Attribute	Req/Opt	Default	Description
text	Required		Message text to log.
log	Optional		If you omit the file attribute, writes messages to standard log file. Ignored, if you specify file attribute. • Application: writes to Application.log, normally used for application-specific messages. • Scheduler: writes to Scheduler.log, normally used to log the execution of scheduled tasks.
file	Optional		Message file. Specify only the main part of the filename. For example, to log to the Testing.log file, specify "Testing". The file must be located in the default log directory. You cannot specify a directory path. If the file does not exist, it is created automatically, with the suffix .log.
type	Optional	Information	Type (severity) of the message: • Information • Warning • Error • Fatal Information
application	Optional	Yes	• Yes: log application name, if it is specified in a cfapplication tag. • No

Usage

This tag logs custom messages to standard or custom log files. You can specify a file for the log message or send messages to the default application or scheduler log. The log message can include ColdFusion expressions. Log files must have the suffix .log and must be located in the ColdFusion log directory.

Log entries are written as comma-delimited lists with these fields:

- type
- thread
- date
- time
- application
- text

Values are enclosed in double quotation marks. If you specify `No` for the `application` attribute, the corresponding entry in the list is empty.

You can disable `cflog` tag execution. For more information, see the ColdFusion Administrator, Basic Security page.

The following example logs the name of a user that logs on an application. The message is logged to the file myAppLog.log in the ColdFusion log directory. It includes the date, time, and thread ID, but not the application name.

```
<Cflog file="myAppLog" application="No"
   text="User #Form.username# logged on.">
```

For example, if a user enters "Sang Thornfield" in a form's username field, this entry is added to the myApplog.log file entry:

"Information","153","02/28/01","14:53:40",,"User Sang Thornfield logged on."

cflogin

Description A container for user login and authentication code. ColdFusion checks the user-provided ID and password against a data source, LDAP directory, or other repository of login identification. Used with `cfloginuser` tag.

Category Extensibility tags

Syntax
```
<cflogin
   idletimeout = "value"
   applicationToken = "token"
   cookieDomain = "domain"
   ...
   <cfloginuser
      name = "name"
      password = "password-string"
      roles = "roles">
   ...>
</cflogin>
```

See also `cfloginuser`, `cflogout`

History New in ColdFusion MX: This tag is new.

Attributes

Attribute	Req/Opt	Default	Description
idletimout	Optional	1800	Time interval with no keyboard activity after which ColdFusion logs the user off. Seconds.
applicationtoken	Optional	the application name	Unique application identifier. Limits the login scope to an application context, so that logins cannot be created illegally.
cookiedomain	Optional		Domain for which the security cookie is valid.

Usage The body of this tag executes only if there is no logged-in user. When using application-based security, you put code in the body of the `cflogin` tag to check the user-provided ID and password against a data source, LDAP directory, or other repository of login identification. The body must include a `cfloginuser` tag to establish the authenticated user's identity in ColdFusion.

The following example shows a simple authentication. This code is typically in the application.cfm page.

Example
```
<cflogin>
   <cfloginuser
      name  = "foo"
      password ="bar"
      roles = "admin">
</cflogin>
<cfoutput>Authorized user: #getAuthUser()#</cfoutput>
<cflogout>
<cfoutput>Authorized user: #getAuthUser()#</cfoutput>
```

cfloginuser

Description Identifies an authenticated user to ColdFusion. Specifies the user ID and roles. Used within a `cflogin` tag.

Category Extensibility tags

Syntax
```
<cfloginuser
   name = "name"
   password = "password-string"
   roles = "roles">
```

See also `cflogin`, `cflogout`

History New in ColdFusion MX: This tag is new.

Attributes

Attribute	Req/Opt	Default	Description
name	Required		A username.
password	Required		A user password.
roles	Required		A comma-delimited list of role identifiers. ColdFusion processes spaces in a list element as part of the element.

Usage If Basic HTTP Authentication is in use, ColdFusion checks the standard security data in the request header. The Cflogin scope within this tag contains the user name and password in the cflogin.name and cflogin.password variables.

Example See cflogin on page 193.

cflogout

Description: Logs the current user out. Removes knowledge of the user ID and roles from the server. If you do not use this tag, the user is automatically logged out when the session ends.

Category: Extensibility tags

Syntax:
```
<cflogout>
```

See also: cflogin, cfloginuser

History: New in ColdFusion MX: This tag is new.

Example:
```
<cflogin>
   <cfloginuser
      name  = "foo"
      password ="bar"
      roles = "admin">
</cflogin>
<cfoutput>Authorized user: #getAuthUser()#</cfoutput>
<cflogout>
<cfoutput>Authorized user: #getAuthUser()#</cfoutput>
```

cfloop

Description

Looping is a programming technique that repeats a set of instructions or displays output repeatedly until one or more conditions are met. This tag supports the following types of loops:

- cfloop: index loop on page 197
- cfloop: conditional loop on page 199
- cfloop: looping over a query on page 200
- cfloop: looping over a list or file on page 202
- cfloop: looping over a COM collection or structure on page 203

Category

Flow-control tags

cfloop: index loop

Description An index loop repeats for a number of times that is determined by a numeric value. An index loop is also known as a FOR loop.

Syntax
```
<cfloop
   index = "parameter_name"
   from = "beginning_value"
   to = "ending_value"
   step = "increment">
   ... HTML or CFML code ...
</cfloop>
```

See also `cfabort`, `cfbreak`, `cfexecute`, `cfexit`, `cfif`, `cflocation`, `cfrethrow`, `cfswitch`, `cfthrow`, `cftry`

Attributes

Attribute	Req/Opt	Default	Description
index	Required		Index value. ColdFusion sets it to `from` value and increments or decrements by `step` value, until it equals `to` value.
from	Required		Beginning value of index.
to	Required		Ending value of index.
step	Optional	1	Step by which to increment or decrement the index value.

Usage Using anything other than integer values in the `from` and `to` attributes of an index loop can product unexpected results. For example, if you increment through an index loop from 1 to 2, with a step of 0.1, ColdFusion outputs "1,1.1,1.2,...,1.9", but not "2". This is a programming language problem regarding the internal representation of floating point numbers.

Note: The `to` value is evaluated once, when the `cfloop` tag is encountered. Any change to this value within the loop block, or within the expression that evaluates to this value, does not affect the number of times the loop is executed.

Example In this example, the code loops five times, displaying the `index` value each time:

```
<cfloop index = "LoopCount" from = "1" to = "5">
   The loop index is <cfoutput>#LoopCount#</cfoutput>.<br>
</cfloop>
```

The output of this loop is as follows:

```
The loop index is 1.
The loop index is 2.
The loop index is 3.
The loop index is 4.
The loop index is 5.
```

In this example, the code loops four times, displaying the `index` value each time. The value of `j` is decreased by one for each iteration. This does not affect the value of `to`, because it is a copy of `j` that is made before entering the loop.

```
<cfset j = 4>
<cfloop index = "LoopCount" from = "1" to = #j#>
   <cfoutput>The loop index is #LoopCount#</cfoutput>.<br>
   <cfset j = j - 1>
</cfloop>
```

The output of this loop is as follows:

```
The loop index is 1.
The loop index is 2.
The loop index is 3.
The loop index is 4.
```

As before, the value of `j` is decremented by one for each iteration, but this does not affect the value of `to`, because its value is a copy of `j` that is made before the loop is entered.

In this example, `step` has the default value, 1. The code decrements the index:

```
<cfloop index = "LoopCount"
   from = "5"
   to = "1"
   step = "-1">
The loop index is <cfoutput>#LoopCount#</cfoutput>.<br>
</cfloop>
```

The output of this loop is as follows:

```
The loop index is 5.
The loop index is 4.
The loop index is 3.
The loop index is 2.
The loop index is 1.
```

cfloop: conditional loop

Description

A conditional loop iterates over a set of instructions as long as a condition is True. To use this type of loop correctly, the instructions must change the condition every time the loop iterates, until the condition is False. Conditional loops are known as WHILE loops, as in, "loop WHILE this condition is true."

Syntax

```
<cfloop
   condition = "expression">
   ...
</cfloop>
```

See also

cfabort, cfbreak, cfexecute, cfexit, cfif, cflocation, cfswitch, cfthrow, cftry

Attributes

Attribute	Req/Opt	Default	Description
condition	Required		Condition that controls the loop.

Example

The following example increments `CountVar` from 1 to 5.

```
<!—- Set the variable CountVar to 0 —->
<cfset CountVar = 0>
<!—- Loop until CountVar = 5 —->
<cfloop condition = "CountVar LESS THAN OR EQUAL TO 5">
   <cfset CountVar = CountVar + 1>
   The loop index is <cfoutput>#CountVar#</cfoutput>.<br>
</cfloop>
```

The output of this loop is as follows:

```
The loop index is 1.
The loop index is 2.
The loop index is 3.
The loop index is 4.
The loop index is 5.
```

cfloop: looping over a query

Description

A loop over a query executes for each record in a query record set. The results are similar to those of the `cfoutput` tag. During each iteration, the columns of the current row are available for output. The `cfloop` tag loops over tags that cannot be used within a `cfoutput` tag.

Syntax

```
<cfloop
   query = "query_name"
   startRow = "row_num"
   endRow = "row_num">
</cfloop>
```

See also

`cfabort`, `cfbreak`, `cfexecute`, `cfexit`, `cfif`, `cflocation`, `cfoutput`, `cfswitch`, `cfthrow`, `cftry`

History

New in ColdFusion MX: On Windows, if the `cfdirectory` tag `action = "list"`, the `cfdirectory` tag does not return the directory entries `"."` (dot) or `".."` (double dot), which represent "the current directory" and "the parent directory." (In earlier releases, ColdFusion returned all the entries.)

CFML code such as the following, which was acceptable in ColdFusion 5, may cause incorrect output in ColdFusion MX:

```
<cfdirectory action = "list" directory ="c:\" name="somename">
Files in c:\<br>
<cfloop query = "somename" startrow = 3>
   #name#<br> >
```

For more information, see cfdirectory on page 73.

Attributes

Attribute	Req/Opt	Default	Description
query	Required		Query that controls the loop.
startRow	Optional		First row of query that is included in the loop.
endRow	Optional		Last row of query that is included in the loop.

Example

This example shows a `cfloop` looping over a query the same way as a `cfoutput` tag that uses the `query` attribute:

```
<cfquery name = "MessageRecords"
   dataSource = "cfsnippets">
      SELECT * FROM Messages
</cfquery>
<cfloop query = "MessageRecords">
   <cfoutput>#Message_ID#</cfoutput><br>
</cfloop>
```

The cfloop tag also iterates over a record set with dynamic start and stop points. This gets the next *n* sets of records from a query. This example loops from the 10th through the 20th record returned by MyQuery:

```
<cfset Start = 10>
<cfset End = 20>
<cfloop query = "MyQuery"
   startRow = "#Start#"
   endRow = "#End#">
   <cfoutput>#MyQuery.MyColName#</cfoutput><br>
</cfloop>
```

The loop stops when there are no more records, or when the current record index is greater than the value of the endRow attribute.

The advantage of looping over a query is that you can use CFML tags that are not allowed in a cfoutput tag. The following example combines the pages that are returned by a query of a list of page names into one document, using the cfinclude tag.

```
<cfquery name = "GetTemplate"
   dataSource = "Library"
   maxRows = "5">
   SELECT TemplateName
   FROM Templates
</cfquery>
<cfloop query = "TemplateName">
   <cfinclude template = "#TemplateName#">
</cfloop>
```

cfloop: looping over a list or file

Description

Looping over a list steps through elements contained in any of these entities:

- A variable
- A value that is returned from an expression
- A file

Syntax

```
<cfloop
   index = "index_name"
   list = "list_items"
   delimiters = "item_delimiter">
   ...
</cfloop>
```

See also

cfabort, cfbreak, cfexecute, cfexit, cfif, cflocation, cfswitch, cfthrow, cftry

Attributes

Attribute	Req/Opt	Default	Description
index	Required		In a list loop, the variable to receive next list element.
list	Required		A list, variable, or file name; contains a list
delimiters	Optional		Character(s) that separates items in list

Example

This loop displays four names:

```
<cfloop index = "ListElement"
   list = "John,Paul,George,Ringo">
      <cfoutput>#ListElement#</cfoutput><br>
</cfloop>
```

You can put more than one character in the `delimiters` attribute, in any order. For example, this loop processes commas, colons, and slashes as list delimiters:

```
<cfloop index = "ListElement"
   list = "John/Paul,George::Ringo"
   delimiters = ",:/">
      <cfoutput>#ListElement#</cfoutput><br>
</cfloop>
```

ColdFusion skips the second and subsequent consecutive delimiters between list elements. Thus, in the example, the two colons between "George" and "Ringo" are processed as one delimiter.

To loop over each line of a file, use the tag this way:

```
<cfloop list="#theFile#"
   index="curLine"
   delimiters="#chr(10)##chr(13)#">
   ...
</cfloop>
```

cfloop: looping over a COM collection or structure

Description

The `cfloop collection` attribute loops over every object within a COM/DCOM collection object, or every element in a structure:

- A COM/DCOM collection object is a set of similar items referenced as a group. For example, the group of open documents in an application is a collection.
- A structure contains a related set of items, or it can be used as an associative array. Looping is particularly useful when using a structure as an associative array.

In the loop, each item is referenced by the variable name in the `item` attribute. The loop executes until all items have been accessed.

The `collection` attribute is used with the `item` attribute. In the example that follows, `item` is assigned a variable called `file2`, so that with each cycle in the `cfloop`, each item in the collection is referenced. In the `cfoutput` section, the `name` property of the `file2` item is referenced for display.

Example

This example uses a COM object to output a list of files. In this example, `FFunc` is a collection of `file2` objects.

```
<cfobject
   class = FileFunctions.files
   name = FFunc
   action = Create>
<cfset FFunc.Path = "c:\">
<cfset FFunc.Mask = "*.*" >
<cfset FFunc.attributes = 16 >
<cfset x = FFunc.GetFileList()>
<cfloop collection = #FFUNC# item = "file2">
   <cfoutput>#file2.name# <br> </cfoutput>
</cfloop>
<!---Loop through a structure that is used as an associative array: --->
...<!--- Create a structure and loop through its contents --->
<cfset Departments = StructNew()>
<cfset val = StructInsert(Departments, "John ", "Sales ")>
<cfset val = StructInsert(Departments, "Tom ", "Finance ")>
<cfset val = StructInsert(Departments, "Mike ", "Education ")>
<!--- Build a table to display the contents --->
<cfoutput>
<table cellpadding = "2 " cellspacing = "2 ">
   <tr>
      <td><b>Employee</b></td>
      <td><b>Dept.</b></td>
   </tr>
<!--- Use item to create the variable person to hold value of key as loop runs --->
<cfloop collection = #Departments# item = "person ">
   <tr>
      <td>#person#</td>
      <td>#StructFind(Departments, person)#</td>
   </tr>
</cfloop>
</table>
</cfoutput>
```

cfmail

Description Sends a e-mail message that contains query output, using an SMTP server.

Category Forms tags, Internet Protocol tags

Syntax
```
<cfmail
   to = "recipient"
   from = "sender"
   cc = "copy_to"
   bcc = "blind_copy_to"
   subject = "msg_subject"
   type = "msg_type"
   maxrows = "max_msgs"
   mimeattach = "path"
   query = "query_name"
   group = "query_column"
   groupcasesensitive = "yes" or "no"
   startrow = "query_row"
   server = "servername"
   port = "port_id"
   mailerid = "headerid"
   timeout = "seconds"
   spoolenable = "yes" or "no">
```

See also cfftp, cfhttp, cfldap, cfmailparam, cfpop

History New in ColdFusion MX: the `SpoolEnable` attribute is new.

Attributes

Attribute	Req/Opt	Default	Description
to	Required		Message recipient name(s). • Static address. For example, `"support@macromedia.com"` • Variable that contains an address. For example, `"#Form.Email#"`. • Name of a query column that contains an address. For example, `"#EMail#"`. An e-mail message is sent for each returned row.
from	Required		E-mail message sender: • A static string; for example, `"support@mex.com"` • A variable; for example, `"#GetUser.EMailAddress#"`
cc	Optional		Address(es) to which to copy the message.
bcc	Optional		Address(es) to which to copy the message, without listing them in the message header.
subject	Required		Message subject. Can be dynamically generated. For example, to send messages that give customers status updates, `"Status of Order Number #Order_ID#"`.

Attribute	Req/Opt	Default	Description
type	Optional		Extended type attributes for message. Informs receiving e-mail client that message has embedded HTML tags to process. Used only by mail clients that support HTML. • HTML
maxRows	Optional		Maximum number of messages to send.
MIMEAttach	Optional		Path of file to attach to message. Attached file is MIME-encoded.
query	Optional		Name of `cfquery` from which to draw data for message(s). Use this attribute to send more than one message, or to send query results within a message.
group	Optional	CurrentRow	Query column to use when you group sets of records to send as a message. For example, to send a set of billing statements to a customer, group on "`Customer_ID`." Case-sensitive. Eliminates adjacent duplicates when data is sorted by the specified field.
groupCaseSensitive	Optional	Yes	Boolean. Whether to consider case when grouping. If the `query` attribute specifies a query object that was generated by a case-insensitive SQL query, set this attribute to `No`, to keep the record set intact.
startRow	Optional	1	Row in a query to start from.
server	Optional		SMTP server address to use for sending messages. Server must be specified here or in Administrator. A value here overrides the Administrator.
port		-1	TCP/IP port on which SMTP server listens for requests. This is normally 25.
mailerID	Optional	ColdFusion Application Server	Mailer ID to be passed in X-Mailer SMTP header, which identifies the mailer application.
timeout	Optional	-1	Number of seconds to wait before timing out connection to SMTP server.
spoolEnable	Optional	Yes	• Yes: Saves a copy of the message until the sending operation is complete. May be slower than the `No` option. • No: Queues the message for sending, without storing a copy until the operation is complete.

Example

```
<h3>cfmail Example</h3>
<p>This view-only example shows the use of cfmail.
<!---
<cfif IsDefined("form.mailto")>
   <cfif form.mailto is not ""
           AND form.mailfrom is not ""
           AND form.Subject is not "">
      <cfmail to = "#form.mailto#"
          from = "#form.mailFrom#"
          subject = "#form.subject#">
           This message was sent by an automatic mailer built with cfmail:
           = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
           #form.body#
      </cfmail>
      <h3>Thank you</h3>
      <p>Thank you, <cfoutput>#mailfrom#: your message, #subject#, has
           been sent to #mailto#</cfoutput>.
   </cfif>
</cfif>
<p>
<form action = "cfmail.cfm">
   <pre>
   TO:   <input type = "Text" name = "MailTo">
   FROM: <input type = "Text" name = "MailFrom">
   SUBJECT: <input type = "Text" name = "Subject">
   <hr>
   MESSAGE BODY:
   <textarea name ="body" cols="40" rows="5" wrap="virtual"></textarea>
   </pre>
   <!--- establish required fields --->
   <input type = "hidden" name = "MailTo_required" value = "You must enter
          a recipient">
   <input type = "hidden" name = "MailFrom_required" value = "You must
          enter a sender">
   <input type = "hidden" name = "Subject_required" value = "You must enter
          a subject">
   <input type = "hidden" name = "Body_required" value = "You must enter
          some text">
   <p><input type = "Submit" name = "">
</form> --->
```

cfmailparam

Description Attaches a file or adds a header to an e-mail message. Used within the `cfmail` tag. You can use more than one `cfmailparam` tag within a `cfmail` tag.

Category Forms tags, Internet Protocol tags

Syntax

```
<cfmail
   to = "recipient"
   subject = "msg_subject"
   from = "sender"
   ...more attributes... >

   <cfmailparam
      file = "file-name" >
   or
   <cfmailparam
      name = "header-name"
      value = "header-value" >
   ...
</cfmail>
```

See also `cfftp`, `cfhttp`, `cfldap`, `cfmail`, `cfpop`

Attributes

Attribute	Req/Opt	Default	Description
file	Required if you do not specify `name` attribute		Attaches file to a message. Mutually exclusive with `name` attribute.
name	Required if you do not specify `file` attribute		Name of header. Case-insensitive. Mutually exclusive with `file` attribute.
value	Optional		Value of header.

Example

```
<h3>cfmailparam Example</h3>
<p>This view-only example uses cfmailparam to attach files and add header to
   a message.</p>
<cfmail from = "peter@domain.com" To = "paul@domain.com"
         Subject = "See Important Attachments and Reply">
   <cfmailparam name = "Reply-To" value = "peter@domain.com">
   Please review the new logo. Tell us what you think.
   <cfmailparam file = "c:\work\readme.txt">
   <cfmailparam file = "c:\work\logo.gif">
</cfmail>
```

cfmodule

Description

Invokes a custom tag for use in ColdFusion application pages. This tag processes custom tag name conflicts.

For more information, see the "Creating and Using Custom CFML Tags" chapter, in *Developing ColdFusion MX Applications with CFML*.

Category

Application framework tags

Syntax

```
<cfmodule
   template = "path"
   name = "tag_name"
   attributeCollection = "collection_structure"
   attribute_name1 = "valuea"
   attribute_name2 = "valueb"
   ...>
```

See also

`cfapplication`, `cfassociate`, `cflock`

History

New in ColdFusion MX: When using this tag within a custom tag, if the `attribute_name` parameter is the same as a key element within the `attributeCollection` parameter, ColdFusion uses the name value that is within the `attributeCollection` parameter. (Earlier releases did not process this consistently.)

Attributes

Attribute	Req/Opt	Default	Description
template	Required unless `name` attribute is used		Mutually exclusive with the `name` attribute. A path to the page that implements the tag. • Relative path: expanded from the current page • Absolute path: expanded using ColdFusion mapping A physical path is not valid.
name	Required unless `template` attribute is used		Mutually exclusive with the `template` attribute. A custom tag name, in the form `"Name.Name.Name..."` Identifies subdirectory, under the ColdFusion tag root directory, that contains custom tag page. For example (Windows format): `<cfmodule name = "macromedia.Forums40.GetUserOptions">` This identifies the page GetUserOptions.cfm in the directory CustomTags\macromedia\Forums40 under the ColdFusion root directory.
attributeCollection	Optional		Structure. A collection of key-value pairs that represent attribute names and values. You can specify multiple key-value pairs. You can specify this attribute only once.

Attribute	Req/Opt	Default	Description
attribute_name	Optional		Attribute for a custom tag. You can include multiple instances of this attribute to specify the parameters of a custom tag.

Usage

To name a ColdFusion page that contains the custom tag definition, including its path, use the `template` attribute. To refer to the custom tag in the ColdFusion installation directory, using dot notation to indicate its location, use the `name` attribute.

You can use `attributeCollection` and `attribute_name` in the same call.

Within the custom tag code, the attributes passed with `attributeCollection` are saved as independent attribute values, with no indication that they are grouped into a structure by the custom tag's caller.

Similarly, if the custom tag uses a `cfassociate` tag to save its attributes, the attributes passed with `attributeCollection` are saved as independent attribute values, with no indication that they are grouped into a structure by the custom tag's caller.

Example

```
<h3>cfmodule Example</h3>
<p>This view-only example shows use of cfmodule to call a custom tag inline.</p>
<p>This example uses a sample custom tag that is saved in myTag.cfm in
         the snippets directory. You can also save ColdFusion custom tags in the
         Cfusion\CustomTags directory.
<cfset attrCollection1 = StructNew()>
<cfparam name="attrCollection1.value1" default="22">
<cfparam name="attrCollection1.value2" default="45">
<cfparam name="attrcollection1.value3" default="88">
<!--- Call the tag with CFMODULE with Name--->
<cfmodule
   Template="myTag.cfm"
   X="3"
   attributeCollection=#attrCollection1#
   Y="4">
<!--- show the code --->
<HR size="2" color="#0000A0">
<P>Here is one way in which to invoke the custom tag,
using the TEMPLATE attribute.</P>
<cfoutput>#HTMLCodeFormat(" <CFMODULE
   Template=""myTag.cfm""
   X=3
   attributeCollection=##attrCollection1##
   Y=4>")#
</cfoutput>
<P>The result: <cfoutput>#result#</cfoutput>
<!--- Call the tag with CFMODULE with Name--->
<!---
   <CFMODULE
      Name="myTag"
      X="3"
      attributeCollection=#attrCollection1#
      Y="4">
--->
```

```
<!--- show the code --->
<HR size="2" color="#0000A0">
<P>Here is another way to invoke the custom tag,
using the NAME attribute.</P>
<cfoutput>#HTMLCodeFormat(" <CFMODULE
      NAME='myTag'
      X=3
      attributeCollection=##attrCollection1##
      Y=4>")#
</cfoutput>
<P>The result: <cfoutput>#result#</cfoutput>
<!--- Call the tag using the short cut notation --->
<!---
<CF_myTag
   X="3"
   attributeCollection=#attrCollection1#
   Y="4">
--->


<!--- show the code --->
<p>Here is the short cut to invoking the same tag.</p>
<cfoutput>#HTMLCodeFormat("<cf_mytag
   x = 3
   attributeCollection = ##attrcollection1##
   y = 4>")#
</cfoutput>
<p>The result: <cfoutput>#result#</cfoutput></p>
```

cfobject

Description

Creates a ColdFusion object, of a specified type.

Note: You can enable and disable this tag in the ColdFusion Administrator page, under ColdFusion Basic Security, Tag Restrictions.

Category

Extensibility tags

Syntax

The tag syntax depends on the object type. Some types use the `type` attribute; others do not. See the following sections:

- cfobject: COM object on page 212
- cfobject: component object on page 214
- cfobject: CORBA object on page 215
- cfobject: Java or EJB object on page 217
- cfobject: web service object on page 219

Note: On UNIX, this tag does not support COM objects.

See also

`cfargument`, `cfcomponent`, `cffunction`, `cfinvoke`, `cfinvokeargument`, `cfproperty`, `cfreturn`

History

New in ColdFusion MX: This tag, and the `CreateObject` function, can instantiate ColdFusion components (CFCs); you can use them within the `cfscript` tag.

cfobject: COM object

Description

Creates and manipulates a Component Object Model (COM) object. Invokes a registered automation server object type.

For information on OLEView, and about COM and DCOM, see the Microsoft OLE Development website: http://www.microsoft.com.

To use this tag, you must provide the object's program ID or filename, the methods and properties available through the IDispatch interface, and the arguments and return types of the object's methods. For most COM objects, you can get this information with the OLEView utility.

Note: On UNIX, this tag does not support COM objects.

Syntax

```
<cfobject
   type = "com"
   action = "action"
   class = "program_ID"
   name = "text"
   context = "context"
   server = "server_name">
```

See also

`cfcollection`, `cfexecute`, `cfindex`, `cfreport`, `cfsearch`, `cfwddx`

Attributes

Attribute	Req/Opt	Default	Description
type	Optional		Object type: • com • corba • java (The other object types do not take the `type` attribute.)
action	Required		• create: instantiates a COM object (typically, a DLL) before invoking methods or properties. • connect: connects to a COM object (typically, an EXE) running on server.
class	Required		Component ProgID for the object to invoke.
name	Required		String; name for the instantiated component
context	Optional		• inproc • local • remote On Windows: If not specified, uses Registry setting.
server	Required if `context = "Remote"`		Server name, using Universal Naming Convention (UNC) or Domain Name Serve (DNS) convention, in one of these forms: • \\lanserver • lanserver • http://www.servername.com • www.servername.com • 127.0.0.1

Example

```
<h3>cfobject (COM) Example</h3>
<!--- Create a COM object as an inproc server (DLL). (class = prog-id)--->
<cfobject action = "Create"
   type = "COM"
   class = Allaire.DocEx1.1
   name = "obj">

<!--- Call a method. Methods that expect no arguments should be called
         using empty parentheses. --->
<cfset obj.Init()>

<!--- This is a collection object. It should support, at a minimum:
   Property : Count
   Method : Item(inarg, outarg)
   and a special property called _NewEnum
--->
<cfoutput>
   This object has #obj.Count# items.
   <br> <HR>
</cfoutput>

<!--- Get the 3rd object in the collection. --->
<cfset emp = obj.Item(3)>
<cfoutput>
   The last name in the third item is #emp.lastname#.
   <br> <HR>
</cfoutput>
<!---Loop over all the objects in the collection.--->
<p>Looping through all items in the collection:
<br>
<cfloop
   collection = #obj#
   item = file2>
   <cfoutput>Last name: #file2.lastname# <br></cfoutput>
</cfloop>
```

cfobject: component object

Description The `cfobject` tag can create a ColdFusion component (CFC) object.

Syntax
```
<cfobject
   name = "variable name"
   component = "component name">
```

See also `cfcollection`, `cfcomponent`, `cfexecute`, `cfindex`, `cfreport`, `cfsearch`, `cfwddx`

Attributes

Attribute	Req/Opt	Default	Description
name	Required		String; name for the instantiated component. The name must not have a period as the first or last character.
component	Required		Name of component to instantiate

Usage Executing operations on a CFC object executes CFML code that implements the CFC's method in the CFC file. Using this tag to instantiate a component returns a ComponentProxy object that can delegate control to a CFC page.

Example
```
<!--- separate instantiation and method invocation; permits multiple
          invocations --->
<cfobject
          name="quoteService"
          component="nasdaq.quote">
<cfinvoke
          component="#quoteService#"
          method="getLastTradePrice"
          symbol="macr"
          returnVariable="res">
<cfoutput>#res#</cfoutput><br>

<cfinvoke
          component="#quoteService#"
          method="getLastTradePrice"
          symbol="mot"
          returnVariable="res">
<cfoutput>#res#</cfoutput>
```

cfobject: CORBA object

Description Calls methods on a registered CORBA object.

Syntax
```
<cfobject
   type = "corba"
   context = "context"
   class = "file or naming service"
   name = "text"
   locale = "type-value arguments">
```

See also `cfcollection`, `cfexecute`, `cfindex`, `cfreport`, `cfsearch`, `cfwddx`

History New in ColdFusion MX: the Naming Service separator format for addresses has changed from a dot to a forward slash. For example, if "`context=NameService`", for a class, use either of the following formats for the `class` parameter:

- "`Macromedia/Eng/CF`"
- "`Macromedia.current/Eng.current/CF`"

(In earlier releases, the format was "`Macromedia.Eng.CF`".)

New in ColdFusion MX: the `locale` attribute specifies the Java config that contains the properties file.

Attributes

Attribute	Req/Opt	Default	Description
type	Optional		Object type: • com • corba • java (The other object types do not take the `type` attribute.)
context	Required		• ior: ColdFusion uses Interoperable Object Reference (IOR) to access CORBA server. • nameservice: ColdFusion uses naming service to access server. This option is valid only with the InitialContext of a VisiBroker Orb.
class	Required		• If `context = "ior"`: absolute path of file that contains string version of the Interoperable Object Reference (IOR). ColdFusion must be able to read file; it should be local to ColdFusion server or accessible on network. • If `context = "nameservice`:" forward slash-delimited naming context for naming service. For example: `Allaire//Doc/empobject`
name	Required		String; name for the instantiated component. An application uses it to reference the CORBA object's methods and attributes.
locale	Optional		Sets arguments for a call to `init_orb`. Use of this attribute is specific to VisiBroker ORBs. It is available on C++, Version 3.2. The value must be in the form: `locale = " -ORBagentAddr 199.99.129.33 -ORBagentPort 19000"` Each type-value pair must start with a hyphen.

Usage ColdFusion Enterprise version 4.0 and later supports CORBA through the Dynamic Invocation Interface (DII). To use `cfobject` with CORBA objects, you must provide the name of the file that contains a string-formatted version of the IOR, or the object's naming context in the naming service; and the object's attributes, method names, and method signatures.

User-defined types (for example, structures) are not supported.

Example
```
<cfobject type = "corba"
  context = "ior"
  class = "c:\\myobject.ior"
  name = "GetName">
```

cfobject: Java or EJB object

Description Creates and manipulates a Java and Enterprise Java Bean (EJB) object.

Syntax
```
<cfobject
   type = "Java"
   action = "Create"
   class = "Java class"
   name = "object name">
```

See also cfcollection, cfexecute, cfindex, cfreport, cfsearch, cfwddx

Attributes

Attribute	Req/Opt	Default	Description
type	Optional		Object type: • com • corba • java (The other object types do not take the `type` attribute.)
action	Required		Create: Creates a Java or WebLogic Environment object.
class	Required		Java class.
name	Required		String; name for the instantiated component.

Usage To call Java CFXs or Java objects, ColdFusion uses a Java Virtual Machine(JVM) that is embedded in the process. You can configure JVM loading, location and settings in the ColdFusion Administrator.

Any Java class available in the class path that is specified in the ColdFusion Administrator can be loaded and used from ColdFusion, using the `cfobject` tag.

To access Java methods and fields, do the following steps:

1 Call the `cfobject` tag, to load the class. See the example code.

2 Use the `init` method with appropriate arguments, to call a constructor. For example:
```
<cfset ret = myObj.init(arg1, arg2)>
```

Calling a public method on the object without first calling the `init` method results in an implicit call to the default constructor. Arguments and return values can be any Java type (simple, array, object). ColdFusion makes the conversions if strings are passed as arguments, but not if they are received as return values.

Overloaded methods are supported if the number of arguments is different.

Calling EJBs

To create and call EJB objects, use the `cfobject` tag. In the second example below, the WebLogic JNDI is used to register and find EJBHome instances.

Example

```
<!--- Example of a Java Object his cfobject call loads the class MyClass
         but does not create an instance object. Static methods and fields
         are accessible after a call to cfobject. --->
<cfobject
   action = "create"
   type = "java"
   class = "myclass"
   name = "myobj">


<!---- Example of an EJB - The cfobject tag creates the Weblogic Environment
         object, which is used to get InitialContext. The context object is
         used to look up the EJBHome interface. The call to create() results
         in getting an instance of stateless session EJB. --->

<cfobject
   action = "create"
   type = "java"
   class = "weblogic/jndi/Environment"
   name = "wlEnv">

<cfset ctx = wlEnv.getInitialContext()>
<cfset ejbHome = ctx.lookup("statelessSession.TraderHome")>
<cfset trader = ejbHome.Create()>
<cfset value = trader.shareValue(20, 55.45)>
<cfoutput>
    Share value = #value#
</cfoutput>
<cfset value = trader.remove()>
```

cfobject: web service object

Description Creates a web service proxy object.

Syntax
```
<cfobject
   webservice= "http://....?wsdl" or "name set in Administrator"
   name = "myobjectname">
```

See also cfcollection, cfexecute, cfindex, cfreport, cfsearch, cfwddx

Attributes

Attribute	Req/Opt	Default	Description
webservice	Required		URL to web service WSDL file. • Absolute URL of web service • Name (string) assigned in the Administrator to the web service
name	Required		Local name for the web service. String.

Usage Instantiates a proxy object for a web service. You can enter the absolute URL in this tag, or refer to a web service that is entered in the ColdFusion Administrator. To minimize potential code maintenance, enter the web service in the Administrator, then refer to that name in this tag.

cfobjectcache

Description: Flushes the query cache.

Category: Database manipulation tags

Syntax:

```
<cfobjectcache
   action = "clear">
```

See also: cfobject

History: This tag was added in ColdFusion 5.

Attributes

Attribute	Req/Opt	Default	Description
action	Required		clear: Clears queries from the cache in the Application scope

cfoutput

Description Displays the results of a database query or other operation.

Category Data output tags

Syntax
```
<cfoutput
   query = "query_name"
   group = "query_column"
   groupCaseSensitive = "Yes" or "No"
   startRow = "start_row"
   maxRows = "max_rows_output">
</cfoutput>
```

See also cfcol, cfcontent, cfdirectory, cftable

History New in ColdFusion MX: On Windows, if the cfdirectory tag action = "list", the tag does not return the directory entries "." (dot) or ".." (double dot), which represent "the current directory" and "the parent directory." (In earlier releases, it returned all the entries.)

CFML code such as the following, which was acceptable in ColdFusion 5, might cause incorrect output in ColdFusion MX:

```
<cfdirectory action = "list" directory="c:\" name="foo">
Files in c:\<br>
<cfoutput query="foo" startrow=3>
   #name#<br>
</cfoutput>
```

CFML code such as the following, which was acceptable in ColdFusion 5, is acceptable in ColdFusion MX, although it is unnecessary:

```
<cfdirectory directory="c:\" name="foo">
Files in c:\<br>
<cfoutput query="foo"
   <cfif NOT foo.name is "." AND NOT foo.name is "..">
      #name#<br>
   </cfif>
</cfoutput>
```

Attributes

Attribute	Req/Opt	Default	Description
query	Optional		Name of cfquery from which to draw data for output section.
group	Optional		Query column to use when you group sets of records. Use if you retrieved a record set ordered on a query column. For example, if a record set is ordered on "Customer_ID" in the cfquery tag, you can group the output on "Customer_ID." Case-sensitive. Eliminates adjacent duplicates when data is sorted.

Attribute	Req/Opt	Default	Description
groupCaseSensitive	Optional	Yes	Boolean. Whether to group by case. If the query attribute specifies a query object that was generated by a case-insensitive SQL query. To keep record set intact, set to "No".
startRow	Optional	1	Row from which to start output.
maxRows	Optional		Maximum number of rows to display.

Usage

To nest `cfoutput` blocks, you must specify the `group` and `query` attributes at the top-most level, and the `group` attribute for each inner block except the innermost `cfoutput` block.

This tag requires an end tag.

Example

```
<!--- This example shows how cfoutput operates --->
<!--- run a sample query --->
<cfquery name = "GetCourses" dataSource = "cfsnippets">
   SELECT Dept_ID, CorName, CorLevel
   FROM courseList
   ORDER by Dept_ID, CorLevel, CorName
</cfquery>
<h3>cfoutput Example</h3>
<p>cfoutput tells ColdFusion Server to begin processing, and then
         to hand back control of page rendering to the web server.
<p>For example, to show today's date, you could write #DateFormat("#Now()#").
         If you enclosed that expression in cfoutput, the result would be
         <cfoutput>#DateFormat(Now())#</cfoutput>.

<p>In addition, cfoutput may be used to show the results of a query
         operation, or only a partial result, as shown:

<p>There are <cfoutput>#getCourses.recordCount#</cfoutput> total records
         in our query. Using the maxRows parameter, we are limiting our
         display to 4 rows.
<p><cfoutput query = "GetCourses" maxRows = 4>
   <PRE>#Dept_ID##CorName##CorLevel#</PRE>
   </cfoutput>

<p>cfoutput can also show the results of a more complex expression,
such as getting the day of the week from today's date. We first
extract the integer representing the Day of the Week from
the server function Now() and then apply the result to
the DayofWeekAsString function:

<br>Today is #DayofWeekAsString(DayofWeek(Now()))#
<br>Today is <cfoutput>#DayofWeekAsString(DayofWeek(Now()))#</cfoutput>
```

cfparam

Description Tests for a parameter's existence, tests its data type, and, if a default value is not assigned, provides one.

Category Variable manipulation tags

Syntax
```
<cfparam
   name = "param_name"
   type = "data_type"
   default = "value">
```

See also `cfcookie`, `cfregistry`, `cfsavecontent`, `cfschedule`, `cfset`

Attributes

Attribute	Req/Opt	Default	Description
name	Required		Name of parameter to test (such as "Client.Email " or "Cookie.BackgroundColor "). If omitted, and if the parameter does not exist, an error is thrown.
type	Optional		• any: any value • array: any array value • binary: a binary value • boolean: a Boolean value • date: a date-time value • numeric: a numeric value • query: a query object • string: a string value or single character • struct: a structure • UUID: a Universally Unique Identifier, formatted 'XXXXXXXX-XXXX-XXXX-XXXXXXXXXXXXXXXX', where 'X' is a hex digit. See CreateUUID on page 396. • variableName: a variable name
default	Optional		Value to set parameter to if it does not exist.

Usage You can use this tag to make the following tests:

- To test whether a required variable exists, use this tag with only the `name` attribute. If it does not exist, ColdFusion Server stops processing the page and returns an error.
- To test whether a required variable exists, and that it is of the specified type, use this tag with the `name` and `type` attributes. If the variable of the specified type does not exist, ColdFusion returns an error.
- To test whether an optional variable exists, use this tag with the `name` and `default` attributes. If it does not exist, it is created and set to the value of the `default` attribute. If it exists, processing continues; the value is not changed.

Example

```
<!--- This example shows how to use CFPARAM to define default values for page
   variables -------->
<cfparam name = "storeTempVar" default = "my default value">
<cfparam name = "tempVar" default = "my default value">

<!--- check if form.tempVar was passed --->
<cfif IsDefined("form.tempVar") is "True">
   <!--- check if form.tempVar is not blank --->
   <cfif form.tempVar is not "">
      <!--- if not, set tempVar to value of form.tempVar --->
      <cfset tempVar = form.tempVar>
   </cfif>
</cfif>

<body>
<h3>cfparam Example</h3>
<p>cfparam is used to set default values so thata developer does not have to
check for the existence of a variable using a function like IsDefined.

<p>The default value of our tempVar is
   "<cfoutput>#StoreTempVar# </cfoutput>"

<!--- check if tempVar is still the same as StoreTempVar
and that tempVar is not blank --->
<cfif tempVar is not #StoreTempVar#
   and tempVar is not "">
      <h3>The value of tempVar has changed: the new value is
      <cfoutput>#tempVar#</cfoutput></h3>
</cfif>

<p>
<form action = "cfparam.cfm" method = "post">
   Type in a new value for tempVar, and hit submit:<br>
   <input type = "Text" name = "tempVar">
   <input type = "Submit" name = "" value = "submit">
</form>
```

cfpop

Description Retrieves and deletes e-mail messages from a POP mail server.

Category Forms tags, Internet Protocol tags

Syntax
```
<cfpop
   server = "servername"
   port = "port_number"
   username = "username"
   password = "password"
   action = "action"
   name = "queryname"
   messageNumber = "number"
   uid = "number"
   attachmentPath = "path"
   timeout = "seconds"
   maxRows = "number"
   startRow = "number"
   generateUniqueFilenames = "boolean">
```

See also `cfftp`, `cfhttp`, `cfldap`, `cfmail`, `cfmailparam`, `SetLocale`

Attributes

Attribute	Req/Opt	Default	Description
server	Required		POP server identifier: • A host name; for example, `"biff.upperlip.com"` • An IP address; for example, `"192.1.2.225"`
port	Optional	110	POP port
username	Optional	Anonymous	• A user name • `"Anonymous"`
password	Optional		Password that corresponds to `username`.
action	Optional	getHeaderOnly	• getHeaderOnly: returns message header information only • getAll: returns message header information, message text, and attachments if attachmentPath is specified • delete: deletes messages on POP server
name	Required if `action = "getAll"` or `"getHeaderOnly"`		Name for index query.

Attribute	Req/Opt	Default	Description
message Number	If `action = "delete"`, this attribute, or `uid`, is required		Message number or comma-delimited list of message numbers to get. Applies to `action ="getAll"` and `"getHeaderOnly"`. For these actions, if it is omitted, all messages are returned. Invalid message numbers are ignored.
uid	If `action = "delete"`, this attribute, or `messageNumber`, is required		UID or a comma-delimited list of UIDs to get. Applies to `action = "getHeaderOnly"` and `action = "getAll"`. For these actions, if it is omitted, all messages are returned. Invalid UIDs are ignored.
attachment Path	Optional		If `action = "getAll"`, allows attachments to be written to directory. If this value is invalid, no attachment files are written to server.
timeout	Optional	60	Maximum time, in seconds, to wait for mail processing.
maxRows	Optional	999999	Number of messages to return, starting with the number in `startRow`. If `messageNumber` is specified, this attribute is ignored.
startRow	Optional	1	First row number to get. If `messageNumber` is specified, this attribute is ignored.
generate Unique Filenames	Optional	No	• Yes: Generate unique filenames for files attached to an e-mail message, to avoid naming conflicts when files are saved • No

Usage

Note: To optimize performance, two retrieve options are available. Message header information is typically short, and therefore quick to transfer. Message text and attachments can be very long, and therefore take longer to process.

cfpop query variables

The following table describes the query variables that are returned by `cfpop`:

Variable names	Description
queryname.recordCount	Number of records returned by query
queryname.currentRow	Current row that `cfoutput` is processing
queryname.columnList	List of column names in query
queryname.UID	Unique identifier for the email message file

Message header and body columns

The following table lists the message header and body columns that are returned if `action = "getHeaderOnly"` or `"getAll"`:

Column name	getHeaderOnly returns	getAll returns
queryname.date	yes	yes
queryname.from	yes	yes
queryname.messagenumber	yes	yes
queryname.replyto	yes	yes
queryname.subject	yes	yes
queryname.cc	yes	yes
queryname.to	yes	yes
queryname.body	not available	yes
queryname.header	not available	yes
queryname.attachments	not available	yes
queryname.attachmentfiles	not available	yes

To create a ColdFusion date/time object from the date-time string that is extracted from a mail message in the `queryname.date` column, use the following table:

Locale	How to create a ColdFusion date/time object from queryname.date
English (US)	Use the `ParseDateTime` function, which converts a date-time value to UTC
Other	Extract the date part of string; pass it to the `LSParseDateTime` function

Note: To set the default display format of date, time, number, and currency values, use the `SetLocale` function.

For more information on `cfpop`, see *Developing ColdFusion MX Applications with CFML*.

Example

```
<!--- This view-only example shows the use of cfpop --->
<h3>cfpop Example</h3>
<p>cfpop lets you retrieve and manipulate mail in a POP3 mailbox.
         This view-only example shows how to create one feature of
         a mail client, to display the mail headers in a POP3 mailbox.
<p>To execute this, un-comment this code and run with a mail-enabled CF Server.
<!---
<cfif IsDefined("form.server ")>
   <!--- make sure server, username are not empty --->
   <cfif form.server is not "" and form.username is not "">
      <cfpop server = "#server# " username = #UserName# password = #pwd#
      action = "GETHEADERONLY " name = "GetHeaders ">
      <h3>Message Headers in Your Inbox</h3>
      <p>Number of Records:
      <cfoutput>#GetHeaders.recordCount#</cfoutput></p>
```

```
      <ul>
         <cfoutput query = "GetHeaders">
         <li>Row: #currentRow#: From: #From# -- Subject: #Subject#
         </cfoutput>
      </ul>
   </cfif>
</cfif>

<form action = "cfpop.cfm " method = "post">
   <p>Enter your mail server:
   <p><input type = "Text" name = "server">
   <p>Enter your username:
   <p><input type = "Text" name = "username">
   <p>Enter your password:
   <p><input type = "password" name = "pwd">
   <input type = "Submit" name = "get message headers">
</form>
--->
```

cfprocessingdirective

Description
Determines whether to suppress the output of extra white space and other CFML output, within the Tag scope. Used with applications that depend on the whitespace characteristics of their output stream.

Category
Data output tags

Syntax
```
<cfprocessingdirective
   pageencoding = "page-encoding literal string">
or
<cfprocessingdirective
   suppressWhiteSpace = "Yes" or "No"
   pageEncoding = "page-encoding literal string">
   CFML tags
</cfprocessingdirective>
```

See also
cfcol, cfcontent, cfoutput, cfsetting, cftable

History
New in ColdFusion MX: You can specify the suppresswhitespace attribute value as a literal string variable. (ColdFusion 5 supported setting it only as a constant.)

New in ColdFusion MX: the pageEncoding attribute is new.

Attributes

Attribute	Req/Opt	Default	Description
suppressWhiteSpace	Optional		Boolean. Whether to suppress white space and other output that is generated by CFML tags within a cfprocessingdirective block.
pageEncoding	Optional		A string literal; the character encoding to use to read the page. The value may be enclosed in single or double quotation marks, or none.

Usage
If you use the pageEncoding attribute, the following rules apply:

- You must put the tag within the first 4096 bytes of a page. It can be positioned after a cfsetting or cfsilent tag.
- If there are multiple occurrences of the tag in one page with the pageEncoding attribute, the attribute must specify the same value; if not, ColdFusion throws an error.
- You cannot embed the tag within conditional logic, because the tag is evaluated when ColdFusion compiles a page (not when it executes the page). For example, the following code has no effect at execution time, because the cfprocessingdirective tag has already been evaluated:

```
<cfif dynEncoding is not "dynamic encoding is not possible">
   <cfprocessingdirective pageencoding=#dynEncoding#>
</cfif>
```

You can specify the suppresswhitespace attribute value as a constant or a variable. To use a variable: define the variable (for example, whitespaceSetting), assign it the value "Yes" or "No", and code a statement such as the following:

```
<!--- ColdFusion allows suppression option to be set at runtime --->
<cfprocessingdirective suppresswhitespace=#whitespaceSetting#>
    code to whose output the setting is applied
</cfprocessingdirective>
```

This tag's options do not apply to pages that are included by cfinclude, cfmodule, custom tag invocation, and so on.

Macromedia recommends that you include either the pageencoding or suppresswhitespace attribute; not both.

If you specify only the pageencoding attribute, you must code it within the tag (do not use a separate end tag).

If you specify the suppresswhitespace attribute, you must code it within the start tag, and use and end tag (</cfprocessingdirective).

When the ColdFusion compiler begins processing a page, it searches for a byte order mark (BOM). Processing is as follows:

- If a BOM is present, ColdFusion uses the encoding that it specifies to read and parse the page.
- If no BOM is present, ColdFusion uses the system default encoding to read and parse the page.

ColdFusion accepts character encoding names that are defined by the Java platform. If an invalid name is specified, ColdFusion throws an InvalidEncodingSpecification exception.

The following example shows the use of a nested cfprocessingdirective tag. The outer tag suppresses unnecessary whitespace during computation of a large table; the inner tag retains whitespace, to output a preformatted table.

Example

```
<cfprocessingdirective suppressWhiteSpace = "Yes">
  <!--- CFML code --->
  <cfprocessingdirective suppressWhiteSpace = "No">
    <cfoutput>#table_data#
    </cfoutput>
  </cfprocessingdirective>
</cfprocessingdirective>
```

The following example shows the use of the pageencoding attribute:

```
<cfprocessingdirective pageencoding = "shift-jis">
```

cfprocparam

Description Parameter information. This tag is nested within a `cfstoredproc` tag.

Category Database manipulation tags

Syntax

```
<cfprocparam
   type = "in" or "out" or "inout"
   variable = "variable name"
   dbVarName = "DB variable name"
   value = "parameter value"
   CFSQLType = "parameter datatype"
   maxLength = "length"
   scale = "decimal places"
   null = "Yes" or "No">
```

See also `cfinsert`, `cfprocresult`, `cfquery`, `cfqueryparam`, `cfstoredproc`, `cftransaction`, `cfupdate`

Attributes

Attribute	Req/Opt	Default	Description
type	Optional	in	• in: passes the parameter by value. • out: passes parameter as bound variable • inout: passes parameter as a bound variable
variable	Required if `type = "OUT" or "INOUT"`		ColdFusion variable name; references the value that the output parameter has after the stored procedure is called.
dbVarName	Required for named notation		Parameter name that corresponds to the name of the parameter in the stored procedure.
value	Required if `type = "OUT" or "INOUT"`		Value that corresponds to the value that ColdFusion passes to the stored procedure.

Attribute	Req/Opt	Default	Description
CFSQLType	Required		• SQL type to which the parameter (any type) is bound: • CF_SQL_BIGINT • CF_SQL_BIT • CF_SQL_BLOB • CF_SQL_CHAR • CF_SQL_CLOB • CF_SQL_DATE • CF_SQL_DECIMAL • CF_SQL_DOUBLE • CF_SQL_FLOAT • CF_SQL_IDSTAMP • CF_SQL_INTEGER • CF_SQL_LONGVARCHAR • CF_SQL_MONEY • CF_SQL_MONEY4 • CF_SQL_NUMERIC • CF_SQL_REAL • CF_SQL_REFCURSOR • CF_SQL_SMALLINT • CF_SQL_TIME • CF_SQL_TIMESTAMP • CF_SQL_TINYINT • CF_SQL_VARCHAR
maxLength	Optional	0	Maximum length of parameter.
scale	Optional	0	Number of decimal places in parameter.
null	Optional	No	Whether parameter is passed as a null value. • Yes: tag ignores the `value` attribute • No

Usage

Use this tag to identify stored procedure parameters and their data types. Code one `cfprocparam` tag for each parameter. The parameters that you code vary based on parameter type and DBMS. The order in which you code `cfprocparam` tags depends on whether the stored procedure uses positional or named notation:

- Positional notation: ColdFusion passes parameters to the stored procedure in the order in which they are defined
- Named notation: The `dbVarName` for the parameter must correspond to the variable name in the stored procedure on the server

Output variables are scoped with the name of the `variable` attribute passed to the tag.

CFML supports Oracle 8 Reference Cursor type, which passes a parameter by reference. Parameters that are passed this way can be allocated and deallocated from memory within the execution of one application.

To use reference cursors in packages or stored procedures, use the cfprocresult tag. This causes Datadirect JDBC to put Oracle reference cursors into a result set. (You cannot use this method with Oracle's ThinClient JDBC drivers.)

Example The following example shows how to invoke an Oracle 8 PL/SQL stored procedure. It makes use of Oracle 8 support of the Reference Cursor type.

The following package, `Foo_Data`, houses a procedure `refcurproc` that declares output parameters as Reference Cursor:

- Parameter `pParam1` returns the rows in the EMP table
- Parameter `pParam2` returns the rows in the DEPT table

The procedure declares one input parameter as an integer, and one output parameter as a two-byte char varying type. Before the `cfstoredproc` tag can call this procedure, it must be created, compiled, and bound in the RDBMS environment.

```
CREATE OR REPLACE PACKAGE Foo_Data AS
   TYPE EmpTyp IS REF CURSOR RETURN Emp%ROWTYPE;
   TYPE DeptTyp IS REF CURSOR RETURN Dept%ROWTYPE;
 PROCEDURE refcurproc(pParam1 in out EmpTyp, pParam2 in out DeptTyp,
          pParam3 in integer, pParam4 out varchar2);
END foo_data;

CREATE OR REPLACE PACKAGE BODY Foo_Data AS
   PROCEDURE RefCurProc(pParam1 in out EmpTyp,
         pParam2 in out DeptTyp,
         pParam3 in integer,
         pParam4 out varchar2) IS
   BEGIN
      OPEN pParam1 FOR select * from emp;
      OPEN pParam2 FOR select * from dept;
      IF pParam3 = 1
      THEN
         pParam4 : = 'hello';
      ELSE
         pParam4 : = 'goodbye';
      END IF;
   END RefCurProc;
END Foo_Data;
```

The following CFML example shows how to invoke the `RefCurProc` procedure using `cfstoredproc`, `cfprocparam`, and `cfprocresult`:

```
<cfstoredproc procedure = "foo_data.refcurproc"
   dataSource = "oracle8i"
   username = "scott"
   password = "tiger"
   returnCode = "No">

   <cfprocparam type = "Out" CFSQLType = "CF_SQL_REFCURSOR"
      variable = "param1">
   <cfprocparam type = "Out" CFSQLType = "CF_SQL_REFCURSOR"
      variable = "param2">
   <cfprocparam type = "IN" CFSQLType = "CF_SQL_INTEGER" value = "1">

   <cfprocparam type = "OUT" CFSQLType = "CF_SQL_VARCHAR"
      variable = "FOO">
```

```
   <cfprocresult name = "rs1">
   <cfprocresult name = "rs2" resultSet = "2">
</cfstoredproc>

<b>The first result set:</b><br>
<hr>
<cftable query = "rs1" colHeaders HTMLTable border = "1">
   <cfcol header = "EMPNO" text = "#EMPNO#">
   <cfcol header = "EMPLOYEE name" text = "#ENAME#">
   <cfcol header = "JOB" text = "#JOB#">
   <cfcol header = "SALARY" text = "#SAL#">
   <cfcol header = "DEPT NUMBER" text = "#DEPTNO#">
</cftable>

<hr>
<b>The second result set:</b><br>

<cftable query = "rs2" colHeaders HTMLTable border = "1">
   <cfcol header = "DEPT name" text = "#DNAME#">
   <cfcol header = "DEPT NUMBER" text = "#DEPTNO#">
</cftable>
<hr>
<cfoutput>
   <b>The output parameter is:</b>'#FOO#'
</cfoutput>
```

cfprocresult

Description Result set name that other ColdFusion tags, such as `cfoutput` and `cftable`, use to access a result set. Identifies which of the stored procedure's result sets to return. This tag is nested within a `cfstoredproc` tag.

Category Database manipulation tags

Syntax
```
<cfprocresult
   name = "query_name"
   resultSet = "1-n"
   maxRows = "maxrows">
```

See also `cfinsert`, `cfprocparam`, `cfquery`, `cfqueryparam`, `cfstoredproc`, `cftransaction`, `cfupdate`

Attributes

Attribute	Req/Opt	Default	Description
name	Required		Name for the query result set.
resultSet	Optional	1	Names one result set, if stored procedure returns more than one.
maxRows	Optional	-1 (All)	Maximum number of rows returned in result set.

Usage To enable access to data returned by the stored procedure, specify one or more `cfprocresult` tags.

The `resultSet` attribute must be unique within the scope of the `cfstoredproc` tag. If you specify a result set twice, the second occurrence overwrites the first.

Example
```
<!--- This example executes a Sybase stored procedure that returns three
         result sets, two of which we want. The stored procedure returns
         status code and one output parameter, which we display. We use
         named notation for parameters. --->
<!--- cfstoredproc tag --->
<cfstoredproc procedure = "foo_proc"
     dataSource = "MY_SYBASE_TEST" username = "sa"
     password = "" dbServer = "scup" dbName = "pubs2"
     returnCode = "Yes" debug = "Yes">
   <!--- cfprocresult tags --->
   <cfprocresult name = RS1>
   <cfprocresult name = RS3 resultSet = 3>
   <!--- cfprocparam tags --->
   <cfprocparam type = "IN"
      CFSQLType = CF_SQL_INTEGER
         value = "1" dbVarName = @param1>

   <cfprocparam type = "OUT" CFSQLType = CF_SQL_DATE
      variable = FOO dbVarName = @param2>
   <!--- Close the cfstoredproc tag --->
</cfstoredproc>
<cfoutput>
   The output param value: '#foo#'<br>
</cfoutput>
```

```
<h3>The Results Information</h3>
<cfoutput query = RS1>#name#,#DATE_COL#<br>
</cfoutput>
<p>
<cfoutput>
   <hr>
   <p>Record Count: #RS1.recordCount# <p>Columns: #RS1.columnList#
   <hr>
</cfoutput>
<cfoutput query = RS3>#col1#,#col2#,#col3#<br>
</cfoutput>
<p>
<cfoutput>
   <hr>
   <p>Record Count: #RS3.recordCount# <p>Columns: #RS3.columnList#
   <hr>
   The return code for the stored procedure is:
          '#cfstoredproc.statusCode#'<br>
</cfoutput>
...
```

cfproperty

Description Defines components as complex types that are used for web services authoring. The attributes of this tag are exposed as component metadata and are subject to inheritance rules.

Category Extensibility tags

Syntax
```
<cfproperty
   name="name"
   type="type"
   ...>
```

See also `cfargument`, `cfcomponent`, `cffunction`, `cfinvoke`, `cfinvokeargument`, `cfobject`, `cfreturn`

History New in ColdFusion MX: This tag is new.

Attributes

Attribute	Req/Opt	Default	Description
name	Required		A string; a property name. Must be a static value.
type	Optional		A string; a property type name; data type. • any • array • binary • boolean • date • guid • numeric • query • string • struct • uuid • variableName • a component name If the value is not a recognized type, ColdFusion processes it as a component name

Usage You must position `cfproperty` tags at the beginning of a component, above executable code and function definitions.

If at least one `cfproperty` tag that has a property within it is present within a component, the property metadata is present in the component metadata.

cfquery

Description

Passes queries or SQL statements to a data source.

Macromedia recommends that you use the `cfqueryparam` tag within every `cfquery` tag, to help secure your databases from unauthorized users. For more information, see:

- Security Bulletin ASB99-04, "Multiple SQL Statements in Dynamic Queries," at http://www.allaire.com/handlers/index.cfm?ID=8728&Method=Full
- *Developing ColdFusion MX Applications with CFML*

Category

Database manipulation tags

Syntax

```
<cfquery
   name = "query_name"
   dataSource = "ds_name"
   dbtype = "query"
   username = "username"
   password = "password"
   maxRows = "number"
   blockFactor = "blocksize"
   timeout = "seconds"
   cachedAfter = "date"
   cachedWithin = "timespan"

   Either of the following:
      debug = "Yes" or "No"
   or:
      debug

   SQL statement(s) >
</cfquery>
```

See also

`cfinsert`, `cfprocparam`, `cfprocresult`, `cfqueryparam`, `cfstoredproc`, `cftransaction`, `cfupdate`

History

New in ColdFusion MX: ColdFusion can index query results and delete index keys.

New in ColdFusion MX: Query of Queries supports a subset of standard SQL. For more information, see *Developing ColdFusion MX Applications with CFML*.

New in ColdFusion MX: ColdFusion supports dot notation within a record set name. ColdFusion interprets such a name as a structure. For more information, see *Developing ColdFusion MX Applications with CFML*.

New in ColdFusion MX: The `connectString`, `dbName`, `dbServer`, `provider`, `providerDSN`, and `sql` attributes, and all values of the `dbtype` attribute except `query`, are deprecated. Do not use them. They do not work, and might cause an error, in releases later than ColdFusion 5.

Attributes

Attribute	Req/Opt	Default	Description
name	Required		Name of query. Used in page to reference query record set. Must begin with a letter. Can include letters, numbers, and underscores.
dataSource	Required		Name of data source from which query gets data.
dbtype	Optional	query	query. Use this value to specify the results of a query as input.
username	Optional		Overrides username in data source setup.
password	Optional		Overrides password in data source setup.
maxRows	Optional	-1 (All)	Maximum number of rows to return in record set.
blockFactor	Optional	1	Maximum rows to get at a time from server. Range: 1 - 100. Applies to ORACLE native database drivers.
timeout			Maximum number of seconds that each action of a query is permitted to execute before returning an error. The cumulative time may exceed this value. For JDBC statements, ColdFusion sets this attribute. For other drivers, check driver documentation.
cachedAfter	Optional		Date value (for example, April 16, 1999, 4-16-99). If date of original query is after this date, ColdFusion uses cached query data. Takes effect only if query caching is enabled in Administrator. To use cached data, current query must use same SQL statement, data source, query name, user name, password. A date/time object is in the range 100 AD–9999 AD. See "How ColdFusion processes two-digit year values" on page 377. When specifying a date value as a string, you must enclose it in quotation marks.
cachedWithin	Optional		Timespan, using the `CreateTimeSpan` function. If original query date falls within the time span, cached query data is used. `CreateTimeSpan` defines a period from the present, back. Takes effect only if query caching is enabled in the Administrator. To use cached data, the current query must use the same SQL statement, data source, query name, user name, and password.
debug	Optional; value and equals sign may be omitted		• Yes, or if omitted: If debugging is enabled, but the Administrator Database Activity option is not enabled, displays SQL submitted to datasource and number of records returned by query • No: If the Administrator Database Activity option is enabled, suppresses display

Usage

Because the `timeout` parameter only the maximum time for each sub-operation of a query, the cumulative time may exceed its value. To set a timeout for a page that might get a very large result set, set the Administrator > Server Settings > Timeout Requests option to an appropriate value.

This tag returns data and query information from a ColdFusion data source. The cumulative query execution time, in seconds, is returned in the variable `cfquery.ExecutionTime`.

This tag creates a query object, providing this information in query variables:

Variable name	Description
`query_name.currentRow`	Current row of query that `cfoutput` is processing
`query_name.columnList`	Comma-delimited list of the query columns
`query_name.RecordCount`	Number of records (rows) returned from the query
`cfquery.ExecutionTime`	Cumulative time required to process the query

You can cache query results and execute stored procedures. For information about this and about displaying `cfquery` output, see *Developing ColdFusion MX Applications with CFML*.

You cannot use SQL reserved words as ColdFusion variable or query names.

Database query results for date and time values can vary in sequence and formatting, unless you use functions to format the results. To ensure that customers using your ColdFusion application are not confused by the display, Macromedia recommends that you use the `DateFormat` and `TimeFormat` functions to format values from queries. For more information and examples, see TechNote 22183, "*ColdFusion Server (5 and 4.5.x) with Oracle: Formatting Date and Time Query Results*," at http://www.coldfusion.com/Support/KnowledgeBase/SearchForm.cfm.

Example

```
<!--- This example shows the use of CreateTimeSpan with CFQUERY ------>
<!--- define startrow and maxrows to facilitate 'next N' style browsing ---->
<cfparam name="MaxRows" default="10">
<cfparam name="StartRow" default="1">
<!-----------------------------------------------------------------------
Query database for information if cached database information has
not been updated in the last six hours; otherwise, use cached data.
------------------------------------------------------------------------>
<cfquery
  name="GetParks" datasource="cfsnippets"
  cachedwithin="#CreateTimeSpan(0, 0, 6, 0)#">
  SELECT PARKNAME, REGION, STATE
  FROM Parks
  ORDER BY ParkName, State
</cfquery>
<!---- build HTML table to display query ------------------------->
<table cellpadding="1" cellspacing="1">
  <tr>
    <td colspan="2" bgcolor="f0f0f0">
      <b><i>Park Name</i></b>
    </td>
```

```
    <td bgcolor="f0f0f0">
      <b><i>Region</i></b>
    </td>
    <td bgcolor="f0f0f0">
      <b><i>State</i></b>
    </td>
  </tr>
  <!---- Output the query and define the startrow and maxrows parameters.
    Use the query variable CurrentCount to keep track of the row you
    are displaying. ------>
  <cfoutput
    query="GetParks" startrow="#StartRow#" maxrows="#MaxRows#">
  <tr>
    <td valign="top" bgcolor="ffffed">
      <b>#GetParks.CurrentRow#</b>
    </td>
    <td valign="top">
      <font size="-1">#ParkName#</font>
    </td>
    <td valign="top">
      <font size="-1">#Region#</font>
    </td>
    <td valign="top">
      <font size="-1">#State#</font>
    </td>
  </tr>
  </cfoutput>
  <!----- If the total number of records is less than or equal
  to the total number of rows, then offer a link to
  the same page, with the startrow value incremented by
  maxrows (in the case of this example, incremented by 10) --------->
  <tr>
    <td colspan="4">
    <cfif (StartRow + MaxRows) LTE GetParks.RecordCount>
      <a href="index.cfm?startrow=
        <cfoutput>   #Evaluate(StartRow + MaxRows)#</cfoutput>
        ">See next <cfoutput>#MaxRows#</cfoutput> rows</a>
    </cfif>
    </td>
  </tr>
</table>
```

cfqueryparam

Description

Checks the data type of a query parameter. This tag is nested within a `cfquery` tag, embedded in a query SQL statement. If you specify optional parameters, this tag performs data validation.

Macromedia recommends that you use the `cfqueryparam` tag within every `cfquery` tag, to help secure your databases from unauthorized users. For more information, see:

- Security Bulletin ASB99-04, "*Multiple SQL Statements in Dynamic Queries*," at http://www.allaire.com/handlers/index.cfm?ID=8728&Method=Full
- *Developing ColdFusion MX Applications with CFML*

Category

Database manipulation tags

Syntax

```
<cfquery
   name = "query_name"
   dataSource = "ds_name"
   ...other attributes...
   SELECT STATEMENT WHERE column_name =
   <cfqueryparam value = "parameter value"
      CFSQLType = "parameter type"
      maxLength = "maximum parameter length"
      scale = "number of decimal places"
      null = "Yes" or "No"
      list = "Yes" or "No"
      separator = "separator character">
   AND/OR ...additional criteria of the WHERE clause...
   </cfquery>
```

See also

`cfinsert`, `cfprocparam`, `cfprocresult`, `cfquery`, `cfstoredproc`, `cftransaction`, `cfupdate`

Attributes

Attribute	Req/Opt	Default	Description
value	Required		Value that ColdFusion passes to the right of the comparison operator in a `where` clause. If `CFSQLType` is a date or time option: • With a native database driver, you cannot specify the `#now()#` function in this attribute. (Use the `DateFormat` or `TimeFormat` function.) • With an ODBC driver, you can specify the `#now()#` function in this attribute.

Attribute	Req/Opt	Default	Description
CFSQLType	Optional	CF_SQL_CHAR	SQL type that parameter (any type) is bound to. • CF_SQL_BIGINT • CF_SQL_BIT • CF_SQL_CHAR • CF_SQL_BLOB • CF_SQL_CLOB • CF_SQL_DATE • CF_SQL_DECIMAL • CF_SQL_DOUBLE • CF_SQL_FLOAT • CF_SQL_IDSTAMP • CF_SQL_INTEGER • CF_SQL_LONGVARCHAR • CF_SQL_MONEY • CF_SQL_MONEY4 • CF_SQL_NUMERIC • CF_SQL_REAL • CF_SQL_REFCURSOR • CF_SQL_SMALLINT • CF_SQL_TIME • CF_SQL_TIMESTAMP • CF_SQL_TINYINT • CF_SQL_VARCHAR
maxLength	Optional	Length of string in value attribute	Maximum length of parameter.
scale	Optional	0	Number of decimal places in parameter. Applies to `CF_SQL_NUMERIC` and `CF_SQL_DECIMAL`.
null	Optional	No	Whether parameter is passed as a null value. • Yes: tag ignores the `value` attribute • No
list	Optional	No	• Yes: The `value` attribute value is a delimited list • No
separator	Required, if you specify a list in value attribute	, (comma)	Character that separates values in list, in `value` attribute.

Usage For data, to ensure that validation is enforced, you must specify the `maxlength` attribute.

This tag does the following:

- Allows the use of SQL bind parameters
- Allows long text fields to be updated from an SQL statement
- Improves performance

The ColdFusion ODBC, DB2, Informix, Oracle 7 and Oracle 8 drivers support SQL bind parameters. The ColdFusion Sybase 11 driver and Sybase native driver do not support SQL bind parameters.

If a database does not support bind parameters, ColdFusion validates and substitutes the validated parameter value back into the string. If validation fails, it returns an error message.

The validation rules are as follows:

- For these types, a data value can be converted to a numeric value: CF_SQL_SMALLINT, CF_SQL_INTEGER, CF_SQL_REAL, CF_SQL_FLOAT, CF_SQL_DOUBLE, CF_SQL_TINYINT, CF_SQL_MONEY, CF_SQL_MONEY4, CF_SQL_DECIMAL, CF_SQL_NUMERIC, and CF_SQL_BIGINT
- For these types, a data value can be converted to a date supported by the target data source: CF_SQL_DATE, CF_SQL_TIME, CF_SQL_TIMESTAMP
- For all other types, if the `maxLength` attribute is used, a data value cannot exceed the maximum length specified.

The SQL syntax that the ColdFusion server generates depends on the target database. For an ODBC, DB2, or Informix data source, the syntax is as follows:

```
SELECT *
   FROM courses
   WHERE col1 = ?
```

For an Oracle 7 or Oracle 8 data source, the syntax is as follows:

```
SELECT *
   FROM courses
   WHERE col1 = :1
```

For a Sybase11 data source, the syntax is as follows:

```
SELECT *
   FROM courses
   WHERE col1 = 10
```

Example

```
<!--- This example shows cfqueryparam with VALID input in Course_ID. --->
<h3>cfqueryparam Example</h3>
<cfset Course_ID = 12>
<cfquery name = "getFirst" dataSource = "cfsnippets">
   SELECT *
   FROM courses
   WHERE Course_ID = <cfqueryPARAM value = "#Course_ID#"
   CFSQLType = "CF_SQL_INTEGER">
</cfquery>
<cfoutput query = "getFirst">
   <p>Course Number: #Course_ID#<br> Description: #descript#</p>
</cfoutput>

<!--- This example shows the use of CFQUERYPARAM when INVALID string data is
   in Course_ID. ---->
<p>This example throws an error because the value passed in the CFQUERYPARAM
tag exceeds the MAXLENGTH attribute</p>

<cfset LastName="Peterson; DELETE employees WHERE LastName='Peterson'">
```

```
<!------- Note that for string input you must specify the MAXLENGTH attribute
   for validation. ------------------------------------------------->
<cfquery
   name="getFirst" datasource="cfsnippets">
   SELECT *
   FROM employees
   WHERE LastName=<cfqueryparam
                     value="#LastName#"
                     cfsqltype="CF_SQL_VARCHAR"
                     maxlength="17">
</cfquery>
<cfoutput
   query="getFirst"> <p>
      Course Number: #FirstName# #LastName#
      Description: #Department# </p>
</cfoutput>
```

cfregistry

Description

This tag is deprecated for the UNIX platform.

Reads, writes, and deletes keys and values in the system registry. Provides persistent storage of client variables.

Note: For this tag execute, it must be enabled in the ColdFusion Administrator. For more information, see *Administering ColdFusion MX*.

Category

Other tags, Variable manipulation tags

Syntax

The tag syntax depends on the `action` attribute value. See the following sections.

- cfregistry action = "getAll" on page 247
- cfregistry action = "get" on page 248
- cfregistry action = "set" on page 249
- cfregistry action = "delete" on page 250

See also

`cfcookie`, `cfparam`, `cfsavecontent`, `cfschedule`, `cfset`

History

New in ColdFusion MX: This tag is deprecated on the UNIX platform. Do not use it in new applications in which ColdFusion Server runs on UNIX. It might not work, and might cause an error, in later releases.

New in ColdFusion MX: The functionality of this tag is available through the web services architecture. ColdFusion stores most persistent data outside the system registry, in XML files.

New in ColdFusion MX: ColdFusion stores most persistent data outside the system registry, in XML files.

cfregistry action = "getAll"

Description Returns all registry keys and values defined in a branch. You can access the values as you would any record set.

Syntax

```
<cfregistry
   action = "getAll"
   branch = "branch"
   type = "data type"
   name = "query name"
   sort = "criteria">
```

Attributes

Attribute	Req/Opt	Default	Description
action	Required		`getall`
branch	Required		Name of a registry branch.
type	Optional	String	• string: return string values • dWord: return DWord values • key: return keys • any: return keys and values
name	Required		Name of record set to contain returned keys and values.
sort	Optional	ASC	Sorts query column data (case-insensitive). Sorts on Entry, Type, and Value columns as text. Specify a combination of columns from query output, in a comma-delimited list. For example: `sort = "value desc, entry asc"` • asc: ascending (a to z) sort order • desc: descending (z to a) sort order

Usage This tag returns `#entry#`, `#type#`, and `#value#` in a record set that you can access through tags such as `cfoutput`. To fully qualify these variables, use the record set name, as specified in the `name` attribute.

If `#type#` is a key, `#value#` is an empty string.

If you specify `type= "any"`, `getAll` also returns binary registry values. For binary values, the `#type#` variable contains UNSUPPORTED and `#value#` is blank.

Example

```
<!--- This example uses cfregistry with the getAll Action --->
<cfregistry action = "getAll"
   branch = "HKEY_LOCAL_MACHINE\Software\Microsoft\Java VM"
   type = "Any" name = "RegQuery">
<p><h1>cfregistry action = "getAll"</h1>
<cftable query = "RegQuery" colHeaders HTMLTable border = "Yes">
<cfcol header = "<b>Entry</b>" width = "35" text = "#RegQuery.Entry#">
<cfcol header = "<b>Type</b>" width = "10" text = "#RegQuery.type#">
<cfcol header = "<b>Value</b>" width = "35" text = "#RegQuery.Value#">
</cftable>
```

cfregistry action = "get"

Description Accesses a registry value and stores it in a ColdFusion variable.

Syntax

```
<cfregistry
   action = "get"
   branch = "branch"
   entry = "key or value"
   variable = "variable"
   type = "data type">
```

Attributes

Attribute	Req/Opt	Default	Description
action	Required		get
branch	Required		Name of a registry branch.
entry	Required		Registry value to access.
variable	Required		Variable into which to put value.
type	Optional	string	• string: return string value • dWord: return DWord value • key: return key's default value

Usage If the value does not exist, `cfregistry` does not create an entry.

Example

```
<!--- This example uses cfregistry with the Get Action --->
<cfregistry action = "get"
 branch = "HKEY_LOCAL_MACHINE\Software\Microsoft\Java VM"
 entry = "ClassPath" type = "String" variable = "RegValue">
<h1>cfregistry action = "get"</h1>
<cfoutput>
   <p>Java ClassPath value is #RegValue#
</cfoutput>
```

cfregistry action = "set"

Description Adds a registry key, adds a value, or updates a value.

Syntax
```
<cfregistry
   action = "set"
   branch = "branch"
   entry = "key or value"
   type = "value type"
   value = "data">
```

Attributes

Attribute	Req/Opt	Default	Description
action	Required		set
branch	Required		Name of a registry branch.
entry	Required		Key or value to set.
type	Optional		• string: set a string value (default). • dWord: set a DWord value. • key: create a key.
value	Optional		Value data to set. If you omit this attribute, `cfregistry` creates default value, as follows: • string: creates an empty string: `""` • dWord: creates a value of `0` (zero)

Usage If it does not exist, `cfregistry` creates the key or value.

Example
```
<!--- This example uses cfregistry Set Action to modify registry value data --->
<!--- Normally you pass in a file name instead of setting one here. --->
<cfset FileName = "dummy.cfm">
<cfregistry action = "set"
 branch = "HKEY_LOCAL_MACHINE\Software\cflangref"
 entry = "LastCFMO1" type = "String" value = "#FileName#">
<h1>cfregistry action = "set"</h1>
```

cfregistry action = "delete"

Description Deletes a registry key or value.

Syntax
```
<cfregistry
   action = "delete"
   branch = "branch"
   entry = "keyorvalue">
```

Attributes

Attribute	Req/Opt	Default	Description
action	Required		delete
branch	Required		• For key deletion: name of registry key to delete. Do not specify the entry. • For value deletion: name of registry branch that contains value to delete. You must specify entry.
entry	Required for value deletion		Value to delete

Usage If you delete a key, `cfregistry` also deletes values and subkeys defined beneath it.

Example
```
<cfregistry action = "delete"
 branch = "HKEY_LOCAL_MACHINE\Software\cflangref\tempkey"
 entry = "LastCFM01">
<h1>cfregistry action = "delete"</h1>
```

cfreport

Description Runs a predefined Crystal Reports report. Applies only to Windows systems. Uses the CFCRYSTAL.exe file to generate reports. Sets parameters in the Crystal Reports engine according to its attribute values.

Category Extensibility tags

Syntax
```
<cfreport
   report = "report_path"
   dataSource = "ds_name"
   type = "type"
   timeout = "number of seconds"
   orderBy = "result_order"
   username = "username"
   password = "password"
   formula = "formula">
</cfreport>
```

See also cfcollection, cfexecute, cfindex, cfobject, cfsearch, cfwddx

History New in ColdFusion MX: Crystal Reports establishes an independent connection to the data source. The connection is not subject to any ColdFusion data source-specific restrictions. For example, the Crystal Reports server can access a data source regardless of whether it is disabled in the ColdFusion Administrator.

Attributes

Attribute	Req/Opt	Default	Description
datasource	Optional		Name of registered or native data source
type	Optional	standard	• standard (not valid for Crystal Reports 8.0) • netscape • microsoft
timeout	Optional		Maximum time, in seconds, in which a connection must be made to a Crystal Report
report	Required		Report path. Store Crystal Reports files in the same directories as ColdFusion page files.
orderBy	Optional		Orders results according to your specifications.
username	Optional		Username required for entry into database from which report is created. Overrides default settings for data source in ColdFusion Administrator.
password	Optional		Password that corresponds to username required for database access. Overrides default settings for data source in ColdFusion Administrator.

Attribute	Req/Opt	Default	Description

formula	Optional		One or more named formulas. Terminate each formula with a semicolon. Use the format:
			`formula = "formulaname1 = 'formula1';formulaname2 = 'formula2';"`
			If you use a semicolon in a formula, you must escape it by typing it twice (;;). For example:
			`formula = "Name1 = 'Val_1a;;Val_1b';Name2 = 'Val2';"`

Usage This tag requires an end tag.

Example
```
<!--- This view-only example shows the use of cfreport --->
<h3>cfreport Tag<h3>
<p>cfreport lets reports from the Crystal Reports Professional report writer
         display through a ColdFusion interface. To run, the tag requires the
         name of the report. cfreport can also pass information to the report
         file displayed, to change the output conditions.
<p>This example would run a report called "monthlysales.rpt " and pass it an
         optional filter condition to show only the information for a subset
         of the report.

<cfreport report = '/reports/monthlysales.rpt'>
     {Departments.Department} = 'International'
</cfreport>

<p>Substitute your report files and filters for this code. cfreport can put
         Crystal Reports into web pages.
```

cfrethrow

Description: Rethrows the currently active exception. Preserves the exception's `cfcatch.type` and `cfcatch.tagContext` variable values.

Category: Exception handling tags, Extensibility tags

Syntax: `<cfrethrow>`

See also: `cferror`, `cfthrow`, `cftry`

Usage: Use this tag within a `cfcatch` block. This tag is useful in error handling code, if the error handler cannot handle an error that it catches. For example, if `cfcatch type = "any"` gets a DATABASE exception, and the code is designed to handle only CFX exceptions, the handler raises the exceptions again, with details intact, so that a higher-level handler can process the error information. If you used the `cfthrow` tag, the type and details of the original exception would be lost.

Example:

```
<h3>cfrethrow Example</h3>
<!--- Rethrow a DATABASE exception. --->
<cftry>
   <cftry>
      <cfquery name = "GetMessages" dataSource = "cfsnippets">
         SELECT  *
         FROM   Messages
      </cfquery>
   <cfcatch type = "DATABASE">
      <!---- If database signalled a 50555 error, ignore; otherwise rethrow
          exception. ---->
      <cfif cfcatch.sqlstate neq 50555>
         <cfrethrow>
      </cfif>
   </cfcatch>
   </cftry>
<cfcatch>
   <h3>Sorry, this request can't be completed</h3>
   <h4>Catch variables</h4>
   <cfoutput>
      <cfloop collection = #cfcatch# item = "c">
         <br>
         <cfif IsSimpleValue(cfcatch[c])>#c# = #cfcatch[c]#
         </cfif>
      </cfloop>
   </cfoutput>
</cfcatch>
</cftry>
```

cfreturn

Description: Returns result values from a component method. Contains an expression returned as result of the function.

Return value: An expression; the result of the function from which this tag is called.

Category: Extensibility tags

Syntax:

```
<cfreturn
   expr>
```

See also: cfargument, cfcomponent, cffunction, cfinvoke, cfinvokeargument, cfobject, cfproperty

History: New in ColdFusion MX: This tag is new.

Attributes

Attribute	Req/Opt	Default	Description
expr	Required		Function result; value of any type

Usage: This tag is equivalent to a `return` statement within a `cfscript` tag. It accepts one return variable argument. To return more than one value, populate a structure with name-value-pairs, and return the structure with this tag.

To access the result value from this tag, you use the variable scope that is the value of the `cfinvoke` tag `returnVariable` attribute.

You can code a maximum of one `cfreturn` tag within a function.

For example code, see the Building and Using ColdFusion Components chapter of *Developing ColdFusion MX Applications with CFML*.

Example:

```
<cfcomponent>
   <cffunction name="getEmp">
      <cfquery name="empQuery" datasource="ExampleApps" >
        SELECT FIRSTNAME, LASTNAME, EMAIL
        FROM tblEmployees
      </cfquery>
      <cfreturn empQuery>
   </cffunction>
   <cffunction name="getDept">
     <cfquery name="deptQuery" datasource="ExampleApps" >
        SELECT *
        FROM tblDepartments
      </cfquery>
      <cfreturn deptQuery>
   </cffunction>
</cfcomponent>
```

cfsavecontent

Description — Saves the generated content of the `cfsavecontent` tag, including the results of evaluating expressions and executing custom tags, in the specified variable.

Category — Variable manipulation tags

Syntax

```
<cfsavecontent
   variable = "variable name">
   the content
</cfsavecontent>
```

See also — `cfcookie`, `cfparam`, `cfregistry`, `cfschedule`, `cfset`

Attributes

Attribute	Req/Opt	Default	Description
variable	Required		Name ofthe variable in which to save the generated content of the tag.

Usage — This tag requires an end tag.

You cannot use this tag to suppress output from a tag library.

Example — The following example uses a custom tag to generate a report and saves the report in the variable CONTENT. It replaces all instances of the word "report" with the phrase "MyCompany Quarterly Report" and outputs the result.

```
<cfsavecontent variable="content">
   <CF_OutputBigReport>
</cfsavecontent>
<cfoutput>
   #replace(content, "report", "MyCompany Quarterly Report", "all")#
</cfoutput>
```

cfschedule

Description

Provides a programmatic interface to the ColdFusion scheduling engine. You can run a page at scheduled intervals, with the option to write out static HTML pages. This lets users access pages that publish data, such as reports, without waiting while a database transaction is performed to populate the page.

You register ColdFusion scheduled events in the ColdFusion Administrator. You can disable `cfschedule` execution in the Administrator.

Information that the user supplies includes the scheduled ColdFusion page to execute, the time and frequency of execution, and whether to publish the task output. If the output is published, a path and file are specified.

The event submission and its success or failure status is written to the \cfusion\log\schedule.log file.

Category

Variable manipulation tags

Syntax

```
<cfschedule
   action = "update"
   task = "taskname"
   operation = "HTTPRequest"
   file = "filename"
   path = "path_to_file"
   startDate = "date"
   startTime = "time"
   url = "URL"
   publish = "Yes" or "No"
   endDate = "date"
   endTime = "time"
   interval = "seconds"
   requestTimeOut = "seconds"
   username = "username"
   password = "password"
   resolveURL = "Yes" or "No"
   proxyServer = "hostname"
   port = "port_number"
   proxyPort = "port_number">

<cfschedule
   action = "delete"
   task = "TaskName">

<cfschedule
   action = "run"
   task = "TaskName">
```

See also

cfcookie, cfparam, cfregistry, cfsavecontent, cfset

Attributes

Attribute	Req/Opt	Default	Description
action	Required		• delete: deletes task • update: creates task, if one does not exist • run: executes task
task	Required		Name of task.
operation	Required if `action = "update"`		Task that scheduler performs. For static page generation, the only option is `"HTTPRequest"`.
file	Required if `publish = "Yes"`		Filename for the published file.
path	Required if `publish = "Yes"`		Path location for the published file.
startDate	Required if `action = "update"`		Date when task scheduling starts.
startTime	Required if `action = "update"`		Time when scheduling of task starts (seconds).
url	Required if `action = "update"`		URL to execute.
publish	Optional	No	• Yes: save the result to a file • No
endDate	Optional		Date when scheduled task ends.
endTime	Optional		Time when scheduled task ends (seconds).
interval	Required if `action = "update"`	One hour	Interval at which task is scheduled. • number of seconds (minimum is 60) • once • daily • weekly • monthly
requestTimeOut	Optional		Can be used to extend the default timeout period.
username	Optional		Username, if URL is protected.
password	Optional		Password, if URL is protected.
proxyServer	Optional		Host name or IP address of a proxy server.
resolveURL	Optional	No	• Yes: resolve links in result page to absolute references • No

Attribute	Req/Opt	Default	Description
port	Optional	80	Server port number from which the task is scheduled. IF `resolveURL= "yes"`, retrieved document URLs that specify a port number are automatically resolved, to preserve links in the retrieved document.
proxyPort	Optional	80	Port number on proxy server from which task is requested. When used with `resolveURL`, URLs of retrieved documents that specify a port number are automatically resolved, to preserve links in retrieved document.

Usage

You cannot use `cfschedule` and apply Secure Sockets Layer (SSL) to an application.

Example

```
<h3>cfschedule Example</h3>
<p>cfschedule provides a programmatic interface to the ColdFusion scheduling
        engine. You can run a specified page at scheduled intervals with the
        option to write out static HTML pages. This lets you offer users access
        to pages that publish data, such as reports, without forcing users to
        wait while a database transaction is performed to populate the data
        on the page.
<p>View-only-example</p>
<cfschedule action = "update"
  task = "TaskName"
  operation = "HTTPRequest"
  url = "http://127.0.0.1/playpen/history.cfm"
  startDate = "8/7/98"
  startTime = "12:25 PM"
  interval = "3600"
  resolveURL = "Yes"
  publish = "Yes"
  file = "sample.html"
  path = "c:\inetpub\wwwroot\playpen"
  requestTimeOut = "600">
```

cfscript

Description Encloses a code block that contains `cfscript` statements.

Category Application framework tags, Other tags

Syntax
```
<cfscript>
   cfscript code here
</cfscript>
```

See also `cfinvoke`, `cfmodule`, `CreateObject`

History New in ColdFusion MX:

- This tag can invoke component methods, using the `createObject` function
- You cannot use ColdFusion reserved words within this tag
- You can use `cftry` and `cfcatch` tags as `cfscript` constructs

Usage Performs processing in CFScript. This tag uses ColdFusion functions, expressions, and operators. You can read and write ColdFusion variables within this tag.

You can use this tag to enclose a series of assignment statements that would otherwise require `cfset` statements.

Caution: If you code a cftry/cfcatch block within this tag using an exception's Java class name, you must provide the fully-qualified class name.

You cannot use some ColdFusion reserved words in this tag. You cannot put a user-defined function whose name begins with any of these strings within this tag:

- cf
- cf_
- _cf
- coldfusion
- coldfusion_
- _coldfusion

You cannot use the `elseif` construct within a `cfscript` tag. You can use code such as the following:

```
else if ( condition )
{
...
}
```

Exception handling with the cfscript tag

To handle exceptions with this tag, use `try` and `catch` statements, which are equivalent to the `cftry` and `cfcatch` tags. For each `try` statement, you must have a `catch` statement. In the `catch` block, the variable `exceptionVariable` contains the exception type. This variable is the equivalent of the `cfcatch` tag built-in variable `cfcatch.Type`. For more information, see *Developing ColdFusion MX Applications with CFML*.

Invoking ColdFusion components with the cfscript tag

CFScript invokes component methods using the `createObject` function.

The following example shows how to invoke a component object with the `cfscript` tag, using ordered arguments:

```
<cfscript>
quote = createObject( “component”, “nasdaq.quote” ) ;
<!--- invocation using ordered arguments --->
res = quote.getLastTradePrice( “macr” ) ;
</cfscript>
```

The following example shows how to use an attribute collection within the `cfscript` tag to pass parameters when invoking a component object. An attribute collection is a structure in which each key corresponds to a parameter name and each value is the parameter value passed for the corresponding key.

```
<cfscript>
   stArgs = structNew();
   stArgs.translationmode = "en_es";
   stArgs.sourceData= "Hello world, friend";
</cfscript>
...
<cfinvoke
   webservice = "http://www.xmethods.net/sd/2001/BabelFishService.wsdl"
   method     = "BabelFish"
   argumentCollection = "#stArgs#"
   returnVariable = "varName">
<cfoutput>#varName#</cfoutput>
```

In this example, the structure is created in a `cfscript` block, but you can use any ColdFusion method to create the structure.

Consuming web services with the cfscript tag

The following example shows how to consume a web service with the `cfscript` tag. You use the `CreateObject` function to connect to the web service.

```
<cfscript>
   ws = CreateObject("webservice",
           "http://www.xmethods.net/sd/2001/BabelFishService.wsdl");
   xlatstring = ws.BabelFish("en_es", "Hello world, friend");
   writeoutput(xlatstring);
</cfscript>
```

For more information, see *Developing ColdFusion MX Applications with CFML*.

Example

```
<p>This simple example shows variable declaration and manipulation.
<cfif IsDefined("form.myValue")>
   <cfif IsNumeric(form.myValue)>
      <cfset x = form.myValue>
      <cfscript>
        y = x;
        z = 2 * y;
        StringVar = form.myString;
      </cfscript>
```

```
   <cfoutput><p>twice #x# is #z#.
      <p>Your string value was: <b><I>#StringVar#</i></b></cfoutput>
<cfelse>
```

cfsearch

Description

Searches Verity collections using ColdFusion or K2Server, whichever search engine a collection is registered by. (ColdFusion can also search collections that have not been registered, with the `cfcollection` tag.)

A collection must be created and indexed before this tag can return search results.

A collection can be **created** in these ways:

- With the `cfcollection` tag
- In the ColdFusion Administrator, which calls the `cfcollection` tag
- Externally, using a native Verity indexing tool, such as Vspider or MKVDK

A collection can be **registered with ColdFusion** in the following ways:

- With the `cfcollection` tag
- In the ColdFusion Administrator, which calls the `cfcollection` tag

A collection can be **registered with K2Server** by editing the k2server.ini file.

A collection can be **indexed** in the following ways:

- In ColdFusion, with the `cfindex` tag
- In the ColdFusion Administrator, which calls the `cfindex` tag
- Using a native Verity indexing tool, such as Vspider or MKVDK

For more information, see *Developing ColdFusion MX Applications with CFML*.

Category

Extensibility tags

Syntax

```
<cfsearch
   name = "search_name"
   collection = "collection_name"
   type = "criteria"
   criteria = "search_expression"
   maxRows = "number"
   startRow = "row_number"
   language = "language">
```

See also

`cfcollection`, `cfexecute`, `cfindex`, `cfobject`, `cfreport`, `cfwddx`

History

New in ColdFusion MX: the The `external` attribute is deprecated. Do not use it in new applications. It might not work, and might cause an error, in later releases. (ColdFusion stores this information about each collection; it automatically detects whether a collection is internal or external.) This tag supports absolute (also known as fully qualified) collection pathnames and mapped collection names.

New in ColdFusion MX: ColdFusion can create collections, index query results and delete index keys.

New in ColdFusion MX: ColdFusion supports Verity operations on Acrobat PDF files.

New in ColdFusion MX: This tag can search multiple collections. In a multiple collection search, you cannot combine collections that are registered with K2Server and and registered in another way.

New in ColdFusion MX: This tag accepts collection names that include spaces.

New in ColdFusion MX: this tag supports Verity 2.6.1 and the LinguistX and ICU locales.

New in ColdFusion MX: This tag can throw the SEARCHENGINE exception.

Attributes

Attribute	Req/Opt	Default	Description
name	Required		Name of the search query.
collection	Required		One or more path(s) and/or registered collection name(s). For a registered collection, specify the collection name. For an unregistered collection, specify an absolute path. Registered names are listed in the ColdFusion Administrator, Verity Collections and Verity Server pages. To specify multiple collections, use a comma delimiter. For example: "`CFUSER, e:\collections\personnel` If you specify multiple collections, you cannot include a combination of collections that are registered by K2Server and registered by Verity.
type	Optional	simple	• simple: STEM and MANY operators are implicitly used. See Developing ColdFusion MX Applications with CFML. • explicit: operators must be invoked explicitly
criteria	Optional		Search criteria. Follows the syntax rules of the `type` attribute. If you pass a mixed-case entry in this attribute, the search is case-sensitive. If you pass all uppercase or all lowercase, the search is case-insensitive. Follow Verity syntax and delimiter character rules; see *Developing ColdFusion MX Applications with CFML*.
maxRows	Optional	All	Maximum number of rows to return in query results. Use double or single quotation marks.
startRow	Optional	1	First row number to get.
language	Optional	english	For options, see cfcollection on page 56. Requires the ColdFusion International Search Pack.

Usage

To permit application users to search Verity collections for non-standard strings, words or characters (for example, "AB23.45.67" or "--->") that would otherwise cause an error, you can create a text file that lists these elements and defines their formats for Verity. Name the file style.lex and put copies of the file in these directories:

- Windows:
  - *cf_root*\lib\common\style (typically, *cf_root* = c:\cfusionmx)
  - *cf_root*\lib\common\style\custom
  - *cf_root*\lib\common\style\file
- Unix:
  - *cf_root*/lib/common/style (typically, *cf_root* = /opt/coldfusionmx)
  - *cf_root*/lib/common/style/custom
  - *cf_root*/lib/common/style/file

Note: To search for a character such as an angle bracket (< or >), you must use a `criteria` attribute value such as `"&lt:"` or `"&lt:"`. The bracket characters are reserved in Verity, and using a backslash to escape the character (`criteria="\<"`) does not work in this context. For more information, see *Developing ColdFusion MX Applications with CFML*.

Macromedia does not recommend using the `cflock` tag with this tag; Verity provides the locking function. Using the `cflock` tag slows search performance.

This tag returns a record set whose columns you can reference in a `cfoutput` tag. For example, the following code specifies a search for the exact terms "filming" or "filmed":

```
<cfsearch
          name = "mySearch"
          collection = "myCollection"
          criteria = '<WILDCARD>`film{ing,ed}`'
          type="explicit"
          startrow=1>
<cfdump var = "#mySearch#>
```

In this example, the single quotation mark (') and backtick (`) characters are used as delimiters; for more information, see *Developing ColdFusion MX Applications with CFML*.

cfsearch result columns

Variable	Description
`url`	Value of URLpath attribute in the `cfindex` tag used to populate a collection. If `type = "custom"`, the value is always empty when you populate a collection.
`key`	Value of the attribute in the `cfindex` tag used to populate collection
`title`	Value of `title` attribute in `cfindex` operation used to populate the collection, including PDF and Office document titles. If title is not provided, the tag uses the `cfindex title` attribute value for each row.
`score`	Relevancy score of document based on search criteria
`custom1`, `custom2`	Value of custom fields in `cfindex` operation used to populate collection.
`summary`	Contents of automatic summary generated by `cfindex`. Default: best three matching sentences, up to 500 characters.
`recordCount`	Number of records returned in record set
`currentRow`	Current row that `cfoutput` is processing
`columnList`	List of column names within record set
`recordsSearched`	Number of records searched

You can use query result columns in standard CFML expressions, preceding the result column name with the name of the query, as follows:

```
#DocSearch.url#
#DocSearch.key#
#DocSearch.title#
#DocSearch.score#
```

Example

```
<!--- #1 (TYPE=SIMPLE) ------------------------------>
<cfsearch
     name="name"
     collection="snippets,syntax,snippets"
     criteria="example" >
<p>
<cfoutput>Search Result total =  #name.RecordCount# </cfoutput><br>
<cfoutput>
     url=#name.url#<br>
     key=#name.key#<br>
     title=#name.title#<br>
     score=#name.score#<br>
     custom1=#name.custom1#<br>
     custom2=#name.custom2#<br>
     summary=#name.summary#<br>
     recordcount=#name.recordcount#<br>
     currentrow=#name.currentrow#<br>
     columnlist=#name.columnlist#<br>
     recordssearched=#name.recordssearched#<br>
</cfoutput>
<cfdump var = #name#>
<br>

<!--- #2 (TYPE=EXPLICIT) ------------------------------>
<cfsearch
     name = "snippets"
     collection = "snippets"
     criteria = '<wildcard>`film{ing,ed}`'
     type="explicit"
     startrow=1>
<cfoutput
     query="snippets">
     url=#url#<br>
     key=#key#<br>
     title=#title#<br>
     score=#score#<br>
     custom1=#custom1#<br>
     custom2=#custom2#<br>
     summary=#summary#<br>
     recordcount=#recordcount#<br>
     currentrow=#currentrow#<br>
     columnlist=#columnlist#<br>
     recordssearched=#recordssearched#<br>
</cfoutput>
<cfdump var = #snippets#>
<br>
```

```
<!--- #3 (search by CF key) ----------------------------->
<cfsearch
      name = "book"
      collection = "custom_book"
      criteria = "cf_key=bookid2">
<cfoutput>
      url=#book.url#<br>
      key=#book.key#<br>
      title=#book.titleE#<br>
      score=#book.score#<br>
      custom1=#book.custom1#<br>
      custom2=#book.custom2#<br>
      summary=#book.summary#<br>
      recordcount=#book.recordcount#<br>
      currentrow=#book.currentrow#<br>
      columnlist=#book.columnlist#<br>
      recordssearched=#book.recordssearched#<br>
</cfoutput>
<cfdump var = #book#>
<br>
```

cfselect

Description Constructs a drop-down list box form control. Used within a `cfform` tag.

You can populate the list from a query, or by using the HTML `option` tag.

Category Forms tags

Syntax

```
<cfselect
  name = "name"
  required = "Yes" or "No"
  message = "text"
  onError = "text"
  size = "integer"
  multiple = "Yes" or "No"
  query = "queryname"
  selected = "column_value"
  value = "text"
  display = "text"
  passThrough = "HTML_attributes">
</cfselect>
```

See also `cfapplet`, `cfform`, `cfgrid`, `cfinput`, `cfgridcolumn`, `cfgridrow`, `cfgridupdate`, `cfslider`, `cftextinput`, `cftree`, `cftreeitem`

Attributes

Attribute	Req/Opt	Default	Description
name	Required		Name of the select form element
size	Optional		Number of entries in drop-down list.
required	Optional	No	• Yes: a list element must be selected when form is submitted. Minimum size of select box is 2. • No
message	Optional		Message to display if `required = "Yes"` and no selection is made.
onError	Optional		Custom JavaScript function to execute if validation fails
multiple	Optional	No	• Yes: allow selecting multiple elements in drop-down list • No
query	Optional		Name of query to populate drop-down list.
selected	Optional		A list of option values to preselect in the selection list. This attribute applies only if list items are generated from a query. The `cfform preservedata` attribute value can override this value.
value	Optional		Query column to use for the value of each list element. Used with `query` attribute.
display	Optional	Value of `value` attribute	Query column to use for the display label of each list element. Used with `query` attribute.

Attribute	Req/Opt	Default	Description
passThrough	Optional		HTML attribute(s) that are not explicitly supported by `cfselect`. If you specify an attribute and its value, they are passed to HTML code that is generated for cfselect tag. For example: `"style=""font-family:courier"""`

Usage

To ensure that a selected list box item persists across postbacks, use the `cfform preserveData` attribute with a list generated from a query. (This strategy works only with data that is populated from a query.)

If the `cfform preserveData` attribute is true and the form posts back to the same page, and if the control is populated by a query, the posted selection(s) for the `cfselect` control are used instead of the `Selected` attribute. For controls that are populated with regular HTML `option` tags, the developer must dynamically add the `Selected` attribute to the appropriate option tag(s).

For this tag to work properly. the browser must be JavaScript-enabled.

To add other HTML <input> tag attributes and values to this tag, use the `passThrough` attribute. They are passed through to the `select` tag that ColdFusion generates for the `cfselect` control when creating a form. The supported HTML attributes are: CLASS, ID, MAXLENGTH, MESSAGE, ONBLUR, ONCHANGE, ONCLICK, ONDBLCLICK, ONFOCUS, SIZE, STYLE, and TABINDEX.

If you put a value in quotation marks, you must escape them; for example:

```
passThrough = "readonly = " "yes " " " "
```

For more information, see the `cfform` tag entry.

This tag requires an end tag.

Example

```
<!--- This example shows the use of cftree, cfselect and cfgrid in a cfform.
         The query takes a list of employees, and uses cftree and cfselect to
         display results of query. cfgrid is used to show an alternate means
         of displaying the data --->
<!--- set a default for the employeeNames variable --->
<cfparam name = "employeeNames" default = "">
<!--- if an employee name has been passed from the form, set employeeNames
         variable to this value --->
<cfif IsDefined("form.employeeNames") is not "False">
   <cfset employeeNames = form.employeeNames>
</cfif>

<!--- query the datasource to find the employee information--->
<cfquery name = "GetEmployees" dataSource = "cfsnippets">
   SELECT  Emp_ID, FirstName, LastName, EMail, Phone, Department
   FROM   Employees
   WHERE 0=0
      <cfif employeeNames is not "">
        AND LastName = '#employeeNames#'
      </cfif>
</cfquery>
```

```
<h3>cfselect Example</h3>
<!--- Use cfform when using other cfinput tools --->
<cfform action = "cfselect.cfm">

<!--- Use cfselect to present the contents of the query by column --->
<h3>cfselect Presentation of Data</h3>
<h4>Click on an employee's last name and hit "see information for this employee"
   to see expanded information.</h4>
<cfselect name = "EmployeeNames"
         message = "Select an Employee Name"
         size = "#getEmployees.recordcount#"
         query = "GetEmployees" value = "LastName"
         required = "No">
<option value = "">Select All
</cfselect>

<input type="Submit"
   name="" value="see information for this employee">
<!--- showing the use of cftree ----------------------------------------->
<!--- use cftree for an expanded presentation of the data Loop through the
     query to create each branch of the CFTREE ----------------------------->
<h3>cftree Presentation of Data</h3>
<h4>Click on the folders to "drill down" and reveal information.</h4> <p>
   <cftree name="SeeEmployees"
      height="150" width="240" font="Arial Narrow" bold="No"
      italic="No" border="Yes" hscroll="Yes" vscroll="Yes"
      required="No" completepath="No" appendkey="Yes" highlighthref="Yes">
   <cfloop query="GetEmployees">
      <cftreeitem value="#Emp_ID#" parent="SeeEmployees" expand="No">
      <cftreeitem value="#LastName#" display="Name" parent="#Emp_ID#"
         queryasroot="No" expand="No">
      <cftreeitem value="#LastName#, #FirstName#" parent="#LastName#"
         expand="No" queryasroot="No">
      <cftreeitem value="#Department#" display="Department" parent="#Emp_ID#"
         queryasroot="No" expand="No">
      <cftreeitem value="#Department#" parent="#Department#"
         expand="No" queryasroot="No">
      <cftreeitem value="#Phone#" display="Phone" parent="#Emp_ID#"
         queryasroot="No" expand="No">
      <cftreeitem value="#Phone#" parent="#Phone#"
         expand="No" queryasroot="No">
      <cftreeitem value="#Email#" display="Email" parent="#Emp_ID#"
         queryasroot="No" expand="No">
      <cftreeitem value="#Email#" parent="#Email#"
         expand="No" queryasroot="No">
   </cfloop>
</cftree>
<!----- You can also use CFGRID for a more comprehensive, quicker view at
   the data ------------------------------------------------------------------->
<h3>CFGRID Presentation of Data</h3>
<cfgrid name="SampleGrid"
   width="600" query="GetEmployees" insert="No" delete="No"
   sort="No" font="Verdana" bold="No" italic="No" appendkey="No"
   highlighthref="No" griddataalign="LEFT" gridlines="Yes"
   rowheaders="No" rowheaderalign="LEFT" rowheaderitalic="No"
```

```
   rowheaderbold="No" colheaders="Yes" colheaderalign="CENTER"
   colheaderitalic="No" colheaderbold="No" bgcolor="Teal"
   selectmode="BROWSE" picturebar="No">
   <cfgridcolumn name="LastName"
      header="Last Name" headeralign="LEFT" dataalign="LEFT"
      bold="No" italic="No" select="Yes" display="Yes" headerbold="No"
      headeritalic="No">
   <cfgridcolumn name="FirstName"
      header="First Name" headeralign="LEFT" dataalign="LEFT"
      fontsize="2" bold="No" italic="No" select="No" display="Yes"
      headerbold="No" headeritalic="No">
   <cfgridcolumn name="Email"
      header="Email" headeralign="LEFT" dataalign="LEFT" bold="No"
      italic="No" select="No" display="Yes" headerbold="No"
      headeritalic="No">
   <cfgridcolumn name="Phone"
      header="Phone" headeralign="LEFT" dataalign="LEFT" bold="No"
      italic="Yes" select="No" display="Yes" headerbold="No" headeritalic="No">
   <cfgridcolumn name="Department"
      header="Department" headeralign="LEFT" dataalign="LEFT" bold="Yes"
      italic="No" select="No" display="Yes" headerbold="No" headeritalic="No">
   <cfgridcolumn name="Emp_ID" header="ID" headeralign="LEFT" dataalign="LEFT"
      width="40" bold="No" italic="No" select="No" display="Yes" headerbold="No"
      headeritalic="No">
</cfgrid>
</cfform>
```

cfservlet

Description

This tag is deprecated. Do not use it in new applications. Executes a Java servlet on a JRun engine.

To access servlets that run on the same server as ColdFUsion, use code such as the following, in which *path* specifies a servlet, jsp, or anything else:

```
GetPageContext().include(path)
GetPageContext().forward(path)
```

For more information, see the JSP PageContext API or or the Servlet RequestDispatcher API.

History

New in ColdFusion MX: This tag is deprecated. Do not use it in new applications. It might not work, and it might cause an error, in later releases.

cfservletparam

Description

This tag is deprecated. Do not use it in new applications.

A child tag of the `cfservlet` tag. Passes data to a servlet. Each `cfservletparam` tag within the `cfservlet` block passes a separate item of data to the servlet.

To access servlets that run on the same server as ColdFUsion, use code such as the following, in which *path* specifies a servlet, jsp, or anything else:

```
GetPageContext().include(path)
GetPageContext().forward(path)
```

For more information, see the JSP PageContext API or or the Servlet RequestDispatcher API.

History

New in ColdFusion MX: This tag is deprecated. Do not use it in new applications. It might not work, and it might cause an error, in later releases.

cfset

Description Defines a ColdFusion variable. If the variable exists, this tag sets it to the specified value.

Category Variable manipulation tags

Syntax
```
<cfset
   variable_name = expression>
```

See also cfcookie, cfparam, cfregistry, cfsavecontent, cfschedule

Attributes

Attribute	Req/Opt	Default	Description
variable_name	Required		Variable

Usage

Arrays

The following example assigns a new array to the variable `months`:

```
<cfset months = ArrayNew(1)>
```

This example creates a variable `Array_Length` that resolves to the length of the array `Scores`:

```
<cfset Array_Length = ArrayLen(Scores)>
```

This example assigns, to index position two in the array `months`, the value `February`:

```
<cfset months[2] = "February">
```

Dynamic variable names

In this example, the variable name is itself a variable:

```
<cfset myvariable = "current_value">
<cfset "#myvariable#" = 5>
```

COM objects

In this example, a COM object is created. A `cfset` defines a value for each method or property in the COM object interface. The last `cfset` creates a variable to store the return value from the COM object's `SendMail` method.

```
<cfobject action = "Create"
   name = "Mailer"
   class = "SMTPsvg.Mailer">

<cfset MAILER.FromName = form.fromname>
<cfset MAILER.RemoteHost = RemoteHost>
<cfset MAILER.FromAddress = form.fromemail>
<cfset MAILER.AddRecipient("form.fromname", "form.fromemail")>
<cfset MAILER.Subject = "Testing cfobject">
<cfset MAILER.BodyText = "form.msgbody">
<cfset Mailer.SMTPLog = "logfile">
<cfset success = MAILER.SendMail()>
<cfoutput> #success# </cfoutput>
```

Example

```
<!--- This example shows how to use cfset --->
<cfquery name = "GetMessages" dataSource = "cfsnippets">
   SELECT  *
   FROM   Messages
</cfquery>

<h3>cfset Example</h3>
<p>cfset sets and reassigns values to local or global variables within a page.

<cfset NumRecords = GetMessages.recordCount>
<p>For example, the variable NumRecords has been declared on this
          page to hold the number of records returned from query
          (<cfoutput>#NumRecords#</cfoutput>).

<p>In addition, cfset can be used to pass variables from other pages,
          such as this example, which takes the url parameter Test from this link:
          (<a href = "cfset.cfm?test = <cfoutput>
                #URLEncodedFormat("hey, you, get off of my cloud")#</cfoutput>
             ">click here</A>) to display a message:
<p>
<cfif IsDefined ("url.test") is "True">
   <cfoutput><b><I>#url.test#</i></b></cfoutput>
<cfelse>
   <h3>The variable url.test has not been passed from another page.</h3>
</cfif>

<p>cfset can also be used to collect environmental variables, such as the
          time, the IP address of the user, or another function or expression.

<cfset the_date = #DateFormat(Now())# & " " & #TimeFormat(Now())#>
<cfset user_ip = CGI.REMOTE_ADDR>
<cfset complex_expr = (23 MOD 12) * 3>
<cfset str_example = Reverse(Left(GetMessages.body, 35))>

<cfoutput>
   <ul>
      <li>The date: #the_date#
      <li>User IP Address: #user_ip#
      <li>Complex Expression ((23 MOD 12) * 3): #complex_expr#
      <li>String Manipulation (the first 35 characters of
           the body of the first message in our query)
           <br><b>Reversed</b> :#str_example#
           <br><b>Normal</b>: #Reverse(str_example)#
   </ul>
</cfoutput>
```

cfsetting

Description: Controls aspects of page processing, such as the output of HTML code in pages.

Category: Page processing tags, Variable manipulation tags

Syntax:

```
<cfsetting
   enableCFoutputOnly = "Yes" or "No"
   showDebugOutput = "Yes" or "No"
   requestTimeOut = "value in seconds" >
```

See also: cfcache, cfflush, cfheader, cfhtmlhead, cfinclude, cfsilent

History:

New in ColdFusion MX: The requestTimeOut attribute is new.

New in ColdFusion MX: The catchExceptionsByPattern attribute is obsolete. Do not use it. It does not work, and causes an error, in releases later than ColdFusion 5.

New in ColdFusion 5.0: The structured exception manager searches for the best-fit cfcatch handler. (In earlier releases, an exception was handled by the first cfcatch block that could handle an exception of its type.)

Attributes

Attribute	Req/Opt	Default	Description
enableCFoutputOnly	Required		• Yes: blocks output of HTML that is outside cfoutput tags • No
showDebugOutput	Optional	Yes	• Yes • No: suppresses debugging information that would otherwise display at end of generated page.
requestTimeout	Optional		• Integer; number of seconds. Time limit, after which ColdFusion processes the page as an unresponsive thread. Overrides the timeout set in the ColdFusion Administrator.

Usage:

The cfsetting requestTimeout attribute replaces the use of requestTmeOut within a URL. To enforce a page timeout, detect the URL variable and use code such as the following to change the page timeout:

```
<cfsetting RequestTimeout = "#URL.RequestTimeout#">
```

You can use this tag to manage whitespace in ColdFusion output pages.

If you nest cfsetting tags: to make HTML output visible, you must match each enableCFoutputOnly = "Yes" statement with an enableCFoutputOnly = "No" statement. For example, after five enableCFoutputOnly = "Yes" statements, to enable HTML output, you must have five corresponding enableCFoutputOnly = "No" statements.

If HTML output is enabled (no matter how many enableCFoutputOnly = "No" statements have been processed) the first enableCFoutputOnly = "Yes" statement blocks output.

If the debugging service is enabled and `showDebugOutput =" Yes"`, the `IsDebugMode` function returns Yes; otherwise, No.

Example

```
<p>CFSETTING is used to control the output of HTML code in Cold Fusion pages.
   This tag can be used to minimize the amount of generated whitespace.

<cfsetting enableCFoutputOnly = "Yes">
   This text is not shown
<cfsetting enableCFoutputOnly = "No">
   <p>This text is shown
<cfsetting enableCFoutputOnly = "Yes">
   <cfoutput>
      <p>Text within cfoutput is always shown
   </cfoutput>
<cfsetting enableCFoutputOnly = "No">
   <cfoutput>
      <p>Text within cfoutput is always shown
   </cfoutput>
```

cfsilent

Description: Suppresses output produced by CFML within a tag's scope.

Category: Data output tags, Page processing tags

Syntax:

```
<cfsilent>
...
</cfsilent>
```

See also: cfcache, cfflush, cfheader, cfhtmlhead, cfinclude, cfsetting

Usage: This tag requires an end tag.

Example:

```
<h3>cfsilent</h3>

<cfsilent>
   <cfset a = 100>
   <cfset b = 99>
   <cfset c = b-a>
   <cfoutput>#c#</cfoutput>
</cfsilent>

<p>Even information within cfoutput tags does not display, within a
         cfsilent block.<br>
b-c =
<cfoutput>
   #c#
</cfoutput>
</p>
```

cfslider

Description Puts a slider control, for selecting a numeric value from a range, in a ColdFusion form. The slider moves over the slider groove. As the user moves the slider, the current value displays. Used within a `cfform` tag.

Category Forms tags

Syntax
```
<cfslider
name = "name"
   label = "text"
   refreshLabel = "Yes" or "No"
   range = "min_value, max_value"
   scale = "uinteger"
   value = "integer"
   onValidate = "script_name"
   message = "text"
   onError = "text"
   height = "integer"
   width = "integer"
   vSpace = "integer"
   hSpace = "integer"
   align = "alignment"
   tickMarkMajor = "Yes" or "No"
   tickMarkMinor = "Yes" or "No"
   tickMarkImages = "URL1, URL2, URLn"
   tickMarkLabels = "Yes" or "No" or "list"
   lookAndFeel = "motif" or "windows" or "metal"
   vertical = "Yes" or "No"
   bgColor = "color"
   textColor = "color"
   font = "font_name"
   fontSize = "integer"
   italic = "Yes" or "No"
   bold = "Yes" or "No"
   notSupported = "text">
```

See also `cfapplet`, `cfinput`, `cfform`, `cfselect`, `cftextinput`, `cftree`, `cftreeitem`

History New in ColdFusion MX: The `img`, `imgStyle`, and `grooveColor` attributes are deprecated. Do not use them in new applications. They might not work, and might cause an error, in later releases.

Attributes

Attribute	Req/ Opt	Default	Description
name	Required		Name of cfslider control.
label	Optional		Label to display with control; for example, `"Volume"` This displays: `"Volume %value%"` To reference the value, use `"%value%"`. If %% is omitted, slider value displays directly after label.

Attribute	Req/ Opt	Default	Description
refreshLabel	Optional	Yes	• Yes: when user moves slider, label is refreshed • No
range	Optional	"0,100"	Numeric slider range values. Separate values with a comma.
scale	Optional		Unsigned integer. Defines slider scale, within `range`. For example, if `range = "0,1000"` and `scale = "100"`, the display values are: 0, 100, 200, 300, ... Signed and unsigned integers in ColdFusion are in the range -2,147,483,648 to 2,147,483,647.
value	Optional	Minimum in range	Starting slider setting. Must be within the `range` values.
onValidate	Optional		Custom JavaScript function to validate user input; in this case, a change to the default slider value. Specify only the function name.
message	Optional		Message text to appear if validation fails.
onError	Optional		Custom JavaScript function to execute if validation fails. Specify only the function name.
height	Optional	40	Slider control height, in pixels.
width	Optional		Slider control width, in pixels.
vSpace	Optional		Vertical spacing above and below slider, in pixels.
hSpace	Optional		Horizontal spacing to left and right of slider, in pixels.
align	Optional		Alignment of slider • top • left • bottom • baseline • texttop • absbottom • middle • absmiddle • right
tickMarkMajor	Optional	No	• Yes: render major tickmarks in slider scale. • No: no major tickmarks. Major tick marks display at intervals specified by `scale`.
tickMarkMinor	Optional	No	• Yes: render minor tickmarks in slider scale. • No: no minor tickmarks Minor tickmarks display between major tickmarks.

Attribute	Req/ Opt	Default	Description
tickMarkImages	Optional		Comma-delimited list of URLs specifying images in slider tickmark scale. If you do not specify enough values, the last value is repeated for remaining tickmarks. If you specify too many values, extra values are ignored.
tickMarkLabels	Optional	No	• yes: numeric tickmarks based on the value of the `range` and `scale` attributes. • no: prevents label text from displaying • Comma-delimited list of strings for tickmark labels; for example, `"ten, twenty, thirty, forty"` If you do not specify enough values, the last value is repeated for remaining tickmarks. If you specify too many values, extra values are ignored.
lookAndFeel	Optional	Windows	• motif: renders slider using Motif style • windows: renders slider using Windows style • metal: renders slider using Java Swing style If platform does not support choice, the tag defaults to the platform's default style.
vertical	Optional	No	• Yes: renders slider in browser vertically. You must set `width` and `height` attributes; ColdFusion does not automatically swap width and height values. • No: renders slider horizontally.
bgColor	Optional		Background color of slider label. For a hex value, use the form: `bgColor = "##xxxxxx"`, where x = 0-9 or A-F; use two pound signs or none. • Any color, in hex format • black • red • blue • magenta • cyan • orange • darkgray • pink • gray • white • lightgray • yellow
textColor	Optional		Options: same as for `bgcolor` attribute
font	Optional		Font name for label text.
fontSize	Optional		Font size for label text, in points.
italic	Optional	No	• Yes: italics label text • No: normal text

Attribute	Req/ Opt	Default	Description
bold	Optional	No	• Yes: bold label text • No: medium text
notSupported	Optional		Text to display if a page that contains a Java applet-based `cfform` control is opened by a browser that does not support Java or has Java support disabled. For example: `"<b> Browser must support Java to view ColdFusion Java Applets</b>"` Default message: Browser must support Java to view ColdFusion Java Applets!

Usage

This tag requires the client to download a Java applet. Using this tag may be slightly slower than using an HTML form element to display the same information.

For this tag to work properly. the browser must be JavaScript-enabled.

If the following conditions are true, a user's selection from query data that populates this tag's options continues to display after the user submits the form:

- The `cfform preserveData` attribute is set to `"Yes"`
- The `cfform action` attribute posts to the same page as the form itself (this is the default), or the action page has a form that contains controls with the same names as corresponding controls on the user entry form

For more information, see the `cfform` tag entry.

Example

```
<!--- This example shows how to use cfslider within cfform --->
<h3>cfslider Example</h3>
<p>cfslider, used within a cfform, can provide functionality
   to Java-enabled browsers.
<p>Try moving the slider back and forth to see the real-time value change.
   Then, submit the form to show how cfslider passes its value on to a new page.
<cfif isdefined("form.mySlider") is true>
   <h3>You slid to a value of <cfoutput>#mySlider#</cfoutput></h3>
   Try again!
</cfif>
<cfform action = "cfslider.cfm">
   <cfslider name = "mySlider" value = "12"
     label = "Actual Slider Value "
     range = "1,100" align = "BASELINE"
     message = "Slide the bar to get a value between 1 and 100"
     height = "50" width = "150" font = "Verdana"
     bgColor = "Silver" bold = "No"
     italic = "Yes" refreshLabel = "Yes"> 100
<p><input type = "Submit" name = "" value = "Show the Result">
</cfform>
```

cfstoredproc

Description
Executes a stored procedure by a JDBC connection to a server database. It specifies database connection information and identifies the stored procedure.

Category
Database manipulation tags

Syntax
```
<cfstoredproc
   procedure = "procedure name"
   dataSource = "ds_name"
   username = "username"
   password = "password"
   blockFactor = "blocksize"
   debug = "Yes" or "No"
   returnCode = "Yes" or "No">
```

See also
cfinsert, cfqueryparam, cfprocparam, cfprocresult, cftransaction, cfquery, cfupdate

History
New in ColdFusion MX: The connectString, dbName, dbServer, dbtype, provider and providerDSN attributes are deprecated. Do not use them. They do not work, and might cause an error, in releases later than ColdFusion 5. (ColdFusion uses Type 4 JDBC drivers.)

Attributes

Attribute	Req/Opt	Default	Description
procedure	Required		Name of stored procedure on database server.
dataSource	Required		Name of data source that points to database that contains stored procedure.
username	Optional		Overrides username in data source setup.
password	Optional		Overrides password in data source setup.
blockFactor	Optional	1	Maximum number of rows to get at a time from server. Range is 1 to 100.
debug	Optional	No	• Yes: Lists debug information on each statement • No
returnCode	Optional	No	• Yes: Tag populates cfstoredproc.statusCode with status code returned by stored procedure. • No

Usage
Within this tag, you code cfprocresult and cfprocparam tags as necessary.

If you set returnCode = "Yes", this tag sets the variable cfstoredproc.statusCode, which holds the status code for a stored procedure. Status code values vary by DBMS. For the meaning of code values, see your DBMS documentation.

This tag sets the variable cfstoredproc.ExecutionTime, which contains the execution time of the stored procedure, in milliseconds.

Before implementing this tag, ensure that you understand stored procedures and their usage. The following examples use a Sybase stored procedure; for an example of an Oracle 8 stored procedure, see cfprocparam.

Example

```
<!--- This view-only example executes a Sybase stored procedure that
         returns three result sets, two of which we want. The stored
         procedure returns the status code and one output parameter,
         which we display. We use named notation for the parameters. --->
<!---
<cfstoredproc procedure = "foo_proc"
  dataSource = "MY_SYBASE_TEST" username = "sa"
  password = "" dbServer = "scup" dbName = "pubs2"
  returnCode = "Yes" debug = "Yes">
  <cfprocresult name = RS1>
  <cfprocresult name = RS3 resultSet = 3>

  <cfprocparam type = "IN" CFSQLType = CF_SQL_INTEGER
       value = "1" dbVarName = @param1>
  <cfprocparam type = "OUT" CFSQLType = CF_SQL_DATE
    variable = FOO dbVarName = @param2>
</cfstoredproc>
--->
<!---
<cfoutput>The output param value: '#foo#'<br></cfoutput>
<h3>The Results Information</h3>
<cfoutput query = RS1>#name#,#DATE_COL#<br></cfoutput><p>
<cfoutput>
  <hr>
  <p>Record Count: #RS1.recordCount# >p>Columns: #RS1.columnList#<hr>
</cfoutput>
<cfoutput query = RS3>#col1#,#col2#,#col3#<br>
</cfoutput><p>
<cfoutput>
  <hr>
  <p>Record Count: #RS3.recordCount# <p>Columns: #RS3.columnList#<hr>
  The return code for the stored procedure is: '#cfstoredproc.statusCode#'<br>
</cfoutput>
--->
```

cfswitch

Description

Evaluates a passed expression and passes control to the `cfcase` tag that matches the expression result. You can, optionally, code a `cfdefaultcase` tag, which receives control if there is no matching `cfcase` tag value.

Category

Flow-control tags

Syntax

```
<cfswitch
   expression = "expression">
   <cfcase
      value = "value"
      delimiters = "delimiters">
      HTML and CFML tags
   </cfcase>
   additional <cfcase></cfcase> tags
   <cfdefaultcase>
     HTML and CFML tags
   </cfdefaultcase>
</cfswitch>
```

See also

`cfabort`, `cfloop`, `cfbreak`, `cfrethrow`, `cfexecute`, `cfexit`, `cfthrow`, `cfif`, `cftry`, `cflocation`

History

New in ColdFusion MX: you can put the `cfdefaultcase` tag at any position within a `cfswitch` statement; it is not required to be the last item.

Attributes

Attribute	Req/Opt	Default	Description
expression	Required		ColdFusion expression that yields a scalar value. ColdFusion converts integers, real numbers, Booleans, and dates to numeric values. For example, True, 1, and 1.0 are all equal.
value	Required		One or more constant values that `cfswitch` compares to the expression (case-insensitive). If a value matches expression, `cfswitch` executes the code between `cfcase` start and end tags. Duplicate value attributes cause a runtime error.
delimiters	Optional	, (comma)	Character that separates entries in a list of values.

Usage

This tag requires an end tag. All code within this tag must be within a `cfcase` or `cfdefaultcase` tag. Otherwise, ColdFusion throws an error.

Use this tag followed by one or more `cfcase` tags. Optionally, include a `cfdefaultcase` tag. This tag selects the matching alternative from the `cfcase` and `cfdefaultcase` tags, jumps to the matching tag, and executes the code between the `cfcase` start and end tags. You do not have to explicitly break out of the `cfcase` tag, as you do in some languages.

You can specify only one `cfdefaultcase` tag within a `cfswitch` tag. You can put the `cfdefaultcase` tag at any position within a `cfswitch` statement; it is not required to be the last item. For example:

```
<cfset foo=1>
<cfswitch expression="#foo#">
   <cfdefaultcase>
      <cfset fooMore= #foo# + 1>
   </cfdefaultcase>
   <cfcase value="2">
      <cfset fooMore = #foo# + 2>
   </cfcase>
</cfswitch>

<cfdump var=#fooMore#>
```

The `cfswitch` tag provides better performance than a series of `cfif`/`cfelseif` tags, and the code is easier to read.

Example

```
<!--- This example shows the use of cfswitch and cfcase to
         exercise a case statement in CFML --->
<cfquery name = "GetEmployees" dataSource = "cfsnippets">
   SELECT  Emp_ID, FirstName, LastName, EMail, Phone, Department
   FROM   Employees
</cfquery>

<h3>cfswitch Example</h3>
<!--- By outputting the query and using cfswitch, we classify the
         output without using a cfloop construct. --->
<p>Each time the case is fulfilled, the specific information is printed;
if the case is not fulfilled, the default case is output </p>
<cfoutput query="GetEmployees">
<cfswitch expression="#Trim(Department)#">
   <cfcase value="Sales">
      #FirstName# #LastName# is in <b>sales</b><br><br>
   </cfcase>
   <cfcase value="Accounting">
      #FirstName# #LastName# is in <b>accounting</b><br><br>
   </cfcase> <cfcase value="Administration">
      #FirstName# #LastName# is in <b>administration</b><br><br>
   </cfcase>
   <cfdefaultcase>
      #FirstName# #LastName# is not in Sales, Accounting, or
         Administration.<br><br>
   </cfdefaultcase>
</cfswitch>
</cfoutput>
```

cftable

Description

Builds a table in a ColdFusion page. This tag renders data as preformatted text, or, with the `HTMLTable` attribute, in an HTML table. If you don't want to write HTML table tag code, or if your data can be presented as preformatted text, use this tag.

Preformatted text (defined in HTML with the <pre> and </pre> tags) displays text in a fixed-width font. It displays white space and line breaks exactly as they are written within the pre tags. For more information, see an HTML reference guide.

To define table column and row characteristics, use the `cfcol` tag within this tag.

Category

Data output tags

Syntax

```
<cftable
   query = "query_name"
   maxRows = "maxrows_table"
   colSpacing = "number_of_spaces"
   headerLines = "number_of_lines"
   HTMLTable
   border
   colHeaders
   startRow = "row_number">
   ...
</cftable>
```

See also

`cfcol`, `cfoutput`, `cfcontent`, `cfprocessingdirective`, `cflog`, `cftable`

Attributes

Attribute	Req/Opt	Default	Description
query	Required		Name of cfquery from which to draw data.
maxRows	Optional		Maximum number of rows to display in the table.
colSpacing	Optional	2	Number of spaces between columns
headerLines	Optional	2	Number of lines to use for table header (the default leaves one line between header and first row of table).
HTMLTable	Optional		Renders data in an HTML 3.0 table. If you use this attribute (regardless of its value), ColdFusion renders data in an HTML table.
border	Optional		Displays border around table. If you use this attribute (regardless of its value), ColdFusion displays a border around the table. Use this only if you use the `HTMLTable` attribute.
colHeaders	Optional		Displays column heads. If you use this attribute, you must also use the `cfcol` tag `header` attribute to define them. If you use this attribute (regardless of its value), ColdFusion displays column heads.
startRow	Optional	1	The query result row to put in the first table row.

Usage

This tag aligns table data, sets column widths, and defines column heads.

At least one `cfcol` tag is required within this tag. You must put `cfcol` and `cftable` tags adjacent in a page. The only tag that you can nest within this tag is the `cfcol` tag. You cannot nest `cftable` tags.

To display the `cfcol header` text, you must specify the `cfcol header` and the `cftable colHeader` attribute. If you specify either attribute without the other, the header does not display and no error is thrown.

Example

```
<!--- This example shows the use of cfcol and cftable to align information
         returned from a query --->
<!--- This query selects employee information from cfsnippets datasource --->
<cfquery name = "GetEmployees" dataSource = "cfsnippets">
   SELECT Emp_ID, FirstName, LastName, EMail, Phone, Department
   FROM Employees
</cfquery>

<html>
<body>
<h3>cftable Example</h3>

<!--- Note use of HTMLTable attribute to display cftable as an HTML table,
         rather than as PRE formatted information --->
<cftable query = "GetEmployees"
   startRow = "1" colSpacing = "3" HTMLTable>
<!--- each cfcol tag sets width of a column in table, and specifies header
         information and text/CFML with which to fill cell --->
   <cfcol header = "<b>ID</b>"
      align = "Left"
      width = 2
      text  = "#Emp_ID#">

   <cfcol header = "<b>Name/Email</b>"
      align = "Left"
      width = 15
      text  = "<a href = 'mailto:#Email#'>#FirstName# #LastName#</A>">

   <cfcol header = "<b>Phone Number</b>"
      align = "Center"
      width = 15
      text  = "#Phone#">
</cftable>
</body>
</html>
```

cftextinput

Description Puts a single-line text entry box in a `cfform` tag and controls its display characteristics.

Category Forms tags

Syntax

```
<cftextinput
   name = "name"
   value = "text"
   required = "Yes" or "No"
   range = "min_value, max_value"
   validate = "data_type"
   onValidate = "script_name"
   message = "text"
   onError = "text"
   size = "integer"
   font = "font_name"
   fontSize = "integer"
   italic = "Yes" or "No"
   bold = "Yes" or "No"
   height = "integer"
   width = "integer"
   vSpace = "integer"
   hSpace = "integer"
   align = "alignment"
   bgColor = "color"
   textColor = "color"
   maxLength = "integer"
   notSupported = "text">
```

See also `cfapplet`, `cfform`, `cfgrid`, `cfgridcolumn`, `cfgridrow`, `cfgridupdate`, `cfinput`, `cfselect`, `cfslider`, `cftree`, `cftreeitem`

Attributes

Attribute	Req/Opt	Default	Description
name	Required		Name for the `cftextinput` control.
value	Optional		Initial value to display in text control.
required	Optional	No	• Yes: the user must enter or change text • No
range	Optional		Minimum–maximum value range, delimited by a comma. Valid only for numeric data.

Attribute	Req/Opt	Default	Description
validate	Optional		• date: verifies format mm/dd/yy. • eurodate: verifies date format dd/mm/yyyy. • time: verifies time format hh:mm:ss. • float: verifies floating point format. • integer: verifies integer format. • telephone: verifies telephone format ###-###-####. The separator can be a blank. Area code and exchange must begin with digit 1 - 9. • zipcode: verifies, in U.S. formats only, 5- or 9-digit format #####-####. The separator can be a blank. • creditcard: strips blanks and dashes; verifies number using mod10 algorithm. • social_security_number: verifies format ###-##-####. The separator can be a blank. • regular_expression: matches input against `pattern` attribute.
onValidate	Optional		Custom JavaScript function to validate user input. The form object, input object, and input object value are passed to routine, which should return True if validation succeeds, False otherwise. The `validate` attribute is ignored.
pattern	Required if validate = "regular_ expression"		JavaScript regular expression pattern to validate input. Omit leading and trailing slashes.
message	Optional		Message text to display if validation fails.
onError	Optional		Custom JavaScript function to execute if validation fails.
size	Optional		Number of characters displayed before horizontal scroll bar displays.
font	Optional		Font name for text.
fontSize	Optional		Font size for text.
italic	Optional	No	• Yes: italics text • No: normal text
bold	Optional	No	• Yes: bold text • No: medium text
height	Optional	40	Height of the control, in pixels.
width	Optional	200	Width of the control, in pixels.
vSpace	Optional		Vertical spacing of the control, in pixels.
hSpace	Optional		Horizontal spacing of the control, in pixels.

Attribute	Req/Opt	Default	Description
align	Optional		Alignment of text entry box with respect to adjacent HTML content: • top • left • bottom • baseline • texttop • absbottom • middle • absmiddle • right
bgColor	Optional		Background color of control. For a hex value, use the form: `textColor = "##xxxxxx"`, where x = 0-9 or A-F; use two pound signs or none. • any color, in hex format • black • red • blue • magenta • cyan • orange • darkgray • pink • gray • white • lightgray • yellow
textColor	Optional		Text color for control. Options: same as for `bgcolor` attribute.
maxLength	Optional		The maximum length of text entered.
notSupported	Optional		Text to display if a page that contains a Java applet-based `cfform` control is opened by a browser that does not support Java, or has Java support disabled. For example: `notSupported = "<b> Browser must support Java to view ColdFusion Java Applets</b>"` If no message is specified, this message displays: `<b>Browser must support Java to   view ColdFusion Java Applets!</b>`

Usage

This tag requires the client to download a Java applet. Downloading an applet takes time; therefore, using this tag might be slightly slower than using an HTML form element or the `cfinput` tag to get the same information.

For this tag to work properly. the browser must be JavaScript-enabled.

If the following conditions are true, a user's selection from query data that populates this tag's options continues to display after the user submits the form:

- The `cfform preserveData` attribute is set to `"Yes"`
- The `cfform action` attribute posts to the same page as the form itself (this is the default), or the action page has a form that contains controls with the same names as corresponding controls on the user entry form

For more information, see `cfform`.

If the `cfform preserveData` attribute is `"yes"`, and the form posts back to the same page, the posted value (not the value of the `value` attribute) of the `cftextinput` control is used.

Example

```
<h3>cftextinput Example</h3>
cftextinput provides simple validation for text fields in cfform and
         control over font information displayed in cfform input boxes
         for text. For example, the field below must not be blank, and
         provides a client-side message upon erring.
<cfform action = "cftextinput.cfm" method = "post">
<cfif IsDefined("form.myInput")>
   <h3>You entered
      <cfoutput>
        #form.myInput#</cfoutput> into the text box </h3>
</cfif>

<cftextinput name = "myInput"
   font = "Courier" fontSize = 12
   value = "Look, this text is red!"
   textColor = "FF0000"
   message = "This field must not be blank"
   required = "Yes">

<input type = "Submit" name = "" value = "submit">
</cfform>
```

cfthrow

Description

Throws a developer-specified exception, which can be caught with a `cfcatch` tag that has any of the following `type` attribute options:

- `type = "custom_type"`
- `type = "Application"`
- `type = "Any"`

Category

Exception handling tags, Flow-control tags

Syntax 1

```
<cfthrow
   type = "exception_type "
   message = "message"
   detail = "detail_description "
   errorCode = "error_code "
   extendedInfo = "additional_information"
   object = "java_except_object">
```

Syntax 2

```
<cfthrow
   object = #object_name#>
```

See also

`cferror`, `cfrethrow`, `cftry`

History

New in ColdFusion MX: this tag can throw ColdFusion component method exceptions.

Attributes

Attribute	Req/Opt	Default	Description
type	Optional	Application	• A custom type • `Application` Do not enter another predefined type; types are not generated by ColdFusion applications. If you specify Application, you need not specify a type for `cfcatch`.
message	Optional		Message that describes exception event.
detail	Optional		Description of the event. ColdFusion appends error position to description; server uses this parameter if an error is not caught by your code.
errorCode	Optional		A custom error code that you supply.
extendedInfo	Optional		A custom error code that you supply.
object	Optional		Requires the value of the `cfobject` tag `name` attribute. Throws a Java exception from a CFML tag. This attribute is mutually exclusive with all other attributes of this tag.

Usage Use this tag within a `cftry` block, to throw an error. The `cfcatch` block can access accompanying information, as follows:

- Message, with `cfcatch.message`
- Detail, with `cfcatch.detail`
- Error code, with `cfcatch.errorcode`

To get more information, use `cfcatch.tagContext`. This shows where control switches from one page to another in the tag stack (for example, `cfinclude`, `cfmodule`).

To display the information displayed by `tagContext`: in the ColdFusion Administrator, Debugging page, select Enable CFML Stack Trace.

To use this tag with the `object` parameter, you must define an exception using the `cfobject` tag, with code such as the following:

```
<cfobject
      type="java"
      action="create"
      class="coldfusion.tagext.InvalidTagAttributeException"
      name="obj">
<cfset obj.init("Attribute", "value")>
```

With this code, the `cfthrow` statement would be as follows:

```
<cfthrow object=#obj#>
```

The `cfthrow` tag passes exception parameters to the `obj` variable.

Example

```
<h3>cfthrow Example</h3>
<!--- open a cftry block --->
<cftry>
<!--- define a condition upon which to throw the error --->
<cfif NOT IsDefined("URL.myID")>
  <!--- throw the error --->
  <cfthrow message = "ID is not defined">
</cfif>

<!--- perform the error catch --->
<cfcatch type = "application">
<!--- display your message --->
  <h3>You've Thrown an <b>Error</b></h3>
<cfoutput>
  <!--- and the diagnostic feedback from the application server --->
  <p>#cfcatch.message#</p>
  <p>The contents of the tag stack are:</p>
  <cfloop
    index = i
    from = 1 to = #ArrayLen(cfcatch.tagContext)#>
      <cfset sCurrent = #cfcatch.tagContext[i]#>
        <br>#i# #sCurrent["ID"]#
        (#sCurrent["LINE"]#,#sCurrent["COLUMN"]#)
        #sCurrent["TEMPLATE"]#
  </cfloop>
</cfoutput>
</cfcatch>
</cftry>
```

cftrace

Description

Displays and logs debugging data about the state of an application at the time the `cftrace` tag executes. Tracks runtime logic flow, variable values, and execution time. Displays output at the end of the request or in the debugging section at the end of the request; or, in Dreamweaver MX and later, in in the Server Debug tab of the Results window.

ColdFusion logs `cftrace` output to the file logs\cftrace.log, in the ColdFusion installation directory.

Note: To permit this tag to execute, you must enable debugging in the ColdFusion Administrator. Optionally, to report trace summaries, enable the Trace section.

Category

Debugging tags, Variable manipulation tags

Syntax

```
<cftrace
   abort = "Yes or No"
   category = "string"
   inline = "Yes or No"
   text = "string"
   type = "format"
   var = "variable_name"
</cftrace>
```

See also

`cfdump`, `cferror`, `cfrethrow`, `cftry`

History

New in ColdFusion MX: this tag is new.

Attributes

Attribute	Req/Opt	Default	Description
abort	Optional	No	• Yes: calls `cfabort` tag when the tag is executed • No
category	Optional		User-defined string for identifying trace groups
inline	Optional	No	• Yes: displays trace code in addition to the trace summary display. • No
text	Optional		User-defined string or simple variable. Outputs to `cflog text` attribute.
type	Optional	Information	Matches to the `cflog type` attribute; displays an appriate icon. • Information • Warning • Error • Fatal Information
var	Optional		The name of a simple or complex variable to display. Useful for displaying a temporary value, or a value that does not display on any CFM page.

Usage

You cannot put application code within this tag. (This avoids problems that can occur if you disable debugging.)

This tag is useful for debugging CFML code during application development.

You can display `cftrace` tag output in the following ways:

- As a section in the debugging output
- In-line in an application page, and as a section in debugging output. If you specify in-line tracing, ColdFusion flushes all output up to the `cftrace` tag, and displays the trace output when it encounters the tag.

This is an example of a log file entry:

```
"Information","web-4","04/08/02","23:21:30",   ,"[30 ms (1st trace)]
[C:\cfusion\wwwroot\generic.cfm @ line: 9] -
  [thisPage = /generic.cfm]  "
"Information","web-0","04/08/02","23:58:58",   ,"[5187 ms (10)]
[C:\cfusion\wwwroot\generic.cfm @ line: 14] - [category]
  [thisPage = /generic.cfm] [ABORTED] thisPage "
```

For a complex variable, ColdFusion lists the variable name and the number of elements in the object; it does not log the contents of the variable.

The following example traces a FORM variable that is evaluated by a `cfif` block:

Example

```
<cftrace var="FORM.variable"
  text="doing equivalency check for FORM.variable"
  category="form_vars"
  inline="true">
<cfif isDefined("FORM.variable") AND #FORM.variable# EQ 1>
  <h1>Congratulations, you're a winner!</h1>
<cfelse>
  <h1>Sorry, you lost!</h1>
</cfif>
```

cftransaction

Description Groups database queries into a unit. Provides database commit and rollback processing.

Category Database manipulation tags

Syntax
```
<cftransaction
   action = "begin" or "commit" or "rollback"
   isolation = "read_uncommitted" or "read_committed" or
      "repeatable_read" >
</cftransaction>
```

See also cfinsert, cfprocparam, cfprocresult, cfquery, cfqueryparam, cfstoredproc, cfupdate

Attributes

Attribute	Req/Opt	Default	Description
action	Optional	begin	• begin: the start of the block of code to execute. • commit: commits a pending transaction • rollback: rolls back a pending transaction
isolation	Optional		ODBC lock type. • read_uncommitted • read_committed • repeatable_read • serializable

Usage Within a transaction block, you can do the following:

- Commit a database transaction by nesting the `<cftransaction action = "commit"/>` tag within the block
- Roll back a transaction by nesting the `<cftransaction action = "rollback"/>` tag within the block

(In these examples, the slash is alternate syntax that is the equivalent of an end tag.)

Within a transaction block, you can write queries to more than one database, but you must commit or rollback a transaction to one database before writing a query to another. Using CFML error handling, you control whether each transaction is committed, based on the success or failure of the database query. To control how the database engine performs locking during the transaction, use the `isolation` attribute.

Example
```
<p>CFTRANSACTION can be used to group multiple queries that use CFQUERY
into one business event.  Changes to data that is requested by the queries
are not committed to the datasource until all actions within the transaction
block have executed successfully.
<p>This a view-only example.
<!---
<cftransaction>
   <cfquery name='makeNewCourse' datasource='Snippets'>
   INSERT INTO Courses
      (Number, Descript)
   VALUES
      ('#myNumber#', '#myDescription#')
```

```
   </cfquery>

   <cfquery name='insertNewCourseToList' datasource='Snippets'>
   INSERT INTO CourseList
      (CorNumber, CorDesc, Dept_ID,
      CorName, CorLevel, LastUpdate)
   VALUES
      ('#myNumber#', '#myDescription#', '#myDepartment#',
      '#myDescription#', '#myCorLevel#', #Now()#)
   </cfquery>
</cftransaction>
--->
```

cftree

Description Inserts a tree control in a form. Validates user selections. Used within a `cftree` tag block. You can use a ColdFusion query to supply data to the tree.

Category Forms tags

Syntax

```
<cftree name = "name"
   required = "Yes" or "No"
   delimiter = "delimiter"
   completePath = "Yes" or "No"
   appendKey = "Yes" or "No"
   highlightHref = "Yes" or "No"
   onValidate = "script_name"
   message = "text"
   onError = "text"
   lookAndFeel = "motif" or "windows" or "metal"
   font = "font"
   fontSize = "size"
   italic = "Yes" or "No"
   bold = "Yes" or "No"
   height = "integer"
   width = "integer"
   vSpace = "integer"
   hSpace = "integer"
   align = "alignment"
   border = "Yes" or "No"
   hScroll = "Yes" or "No"
   vScroll = "Yes" or "No"
   notSupported = "text">

</cftree>
```

See also `cfapplet`, `cfform`, `cfgrid`, `cfgridcolumn`, `cfgridrow`, `cfgridupdate`, `cfinput`, `cfselect`, `cfslider`, `cftextinput`, `cftree`, `cftreeitem`

History New in ColdFusion MX: ColdFusion renders a tree control regardless of whether there are any `treeitems` within it.

Attributes

Attribute	Req/Opt	Default	Description
name	Required		Name for tree control.
required	Optional	No	• Yes: user must select an item in tree control • No
delimiter	Optional	\\	Character to separate elements in form variable path.

Attribute	Req/Opt	Default	Description
completePath	Optional	No; tree name is returned as root	• Yes: passes the root part of `treename.path` form variable when `cftree` is submitted • No, or omitted: root level of form variable is not passed; path value starts with the first node For the `preserveData` attribute of `cfform` to work with the tree, you must set this attribute to Yes. If you specify a root name for a tree item with `cftreeitem queryasroot`, that value is returned. If you do not specify a root name, ColdFusion returns the query name as the root.
appendKey	Optional	Yes	• Yes: when used with href, passes CFTREEITEMKEY variable with the value of the selected tree item in URL to the application page specified in the `cfform action` attribute • No
highlightHref	Optional	Yes	• Yes: highlights links that are associated with a `cftreeitem` with a `URL` attribute value • No: disables highlight
onValidate	Optional		JavaScript function to validate user input. The form object, input object, and input object value are passed to the specified routine, which should return True if validation succeeds; False, otherwise.
message	Optional		Message to display if validation fails.
onError	Optional		JavaScript function to execute if validation fails.
lookAndFeel	Optional	windows	• motif: renders slider in Motif style • windows: renders slider in Windows style • metal: renders slider in Java Swing style If platform does not support style option, tag defaults to platform default style.
font	Optional		Font name for data in tree control.
fontSize	Optional		Font size for text in tree control, in points.
italic	Optional	No	• Yes: displays tree control text in italics • No
bold	Optional	No	• Yes: displays tree control text in bold • No
height	Optional	320	Tree control height, in pixels.
width	Optional	200	Tree control width, in pixels.
vSpace	Optional		Vertical margin above and below tree control, in pixels.
hSpace	Optional		Horizontal spacing to left and right of tree control, in pixels.

Attribute	Req/Opt	Default	Description
align	Optional		• top • left • bottom • baseline • texttop • absbottom • middle • absmiddle • right
border	Optional	Yes	• Yes • No
hScroll	Optional	Yes	• Yes: permits horizontal scrolling • No
vScroll	Optional	Yes	• Yes: permits vertical scrolling • No
notSupported	Optional		Message to display if page that contains Java applet-based form control is opened by browser that does not support Java, or has Java support disabled. For example: `notSupported = "<b> Browser must support Java to view ColdFusion Java Applets</b>"` If no message is specified, this message displays: `<b>Browser must support Java to   view ColdFusion Java Applets!</b>`

Usage

This tag requires the client to download a Java applet. Downloading an applet takes time; therefore, using this tag might be slightly slower than using an HTML form element or the `cfinput` tag to get the same information.

For this tag to work properly. the browser must be JavaScript-enabled.

If the following conditions are true, a user's selection from query data that populates this tag's options continues to display after the user submits the form:

- The `cfform preserveData` attribute is set to `"Yes"`
- The `cfform action` attribute posts to the same page as the form itself (this is the default), or the action page has a form that contains controls with the same names as corresponding controls on the user entry form

For more information, see the `cfform` tag entry.

Example

```
<!--- This example shows the use of cftree in a cfform. The query takes a list of
         employees, and uses cftree and cfselect to display the results. cfgrid is
         used to show an alternate means of displaying the same data --->
<!--- set a default for the employeeNames variable --->
<cfparam name = "employeeNames" default = "">
<!--- if an employee name has been passed from the form, set employeeNames
         variable to this value --->
<cfif IsDefined("form.employeeNames")>
   <cfset employeeNames = form.employeeNames>
```

```
</cfif>
<!--- query the datasource to find the employee information--->
<cfquery name = "GetEmployees" dataSource = "cfsnippets">
   SELECT  Emp_ID, FirstName, LastName, EMail, Phone, Department
   FROM   Employees where lastname
     <cfif #employeeNames# is not ""> = '#employeeNames#'</cfif>
</cfquery>
<html>
<body>
<h3>cftree Example</h3>
<!--- Use cfform when using other cfinput tools --->
<cfform action = "cftree.cfm">
<!--- Use cfselect to present the contents of the query by column --->
<h3>cfselect Presentation of Data</h3>
<h4>Click an employee's last name and "see information for this employee",
         to see expanded information.</h4>
<cfselect name = "EmployeeNames" message = "Select an Employee Name"
   size = "#getEmployees.recordcount#" query = "GetEmployees"
  value = "LastName" required = "No">
   <option value = "">Select All
</cfselect>
<input type = "Submit" name = ""
   value = "see information for this employee">
<!--- showing the use of cftree --->
<!--- Use cftree for an expanded presentation of the data --->
<!--- Loop through the query to create each branch of the cftree --->
<h3>cftree Presentation of Data</h3>
<h4>Click on the folders to "drill down" and reveal information.</h4>
<p>cftreeitem is used to create the "branches" of the tree.
<p>
<cftree name = "SeeEmployees" height = "150" width = "240"
   font = "Arial Narrow" bold = "No"
   italic = "No" border = "Yes"
   hScroll = "Yes" vScroll = "Yes"
   required = "No" completePath = "No"
   appendKey = "Yes" highlightHref = "Yes">
<cfloop query = "GetEmployees">
   <cftreeitem value = "#Emp_ID#" parent = "SeeEmployees" expand = "No">
   <cftreeitem value = "#LastName#" display = "Name"
      parent = "#Emp_ID#" queryAsRoot = "No"
      expand = "No">
   <cftreeitem value = "#LastName#, #FirstName#"
      parent = "#LastName#" expand = "No"
      queryAsRoot = "No">
   <cftreeitem value = "#Department#" display = "Department"
      parent = "#Emp_ID#" queryAsRoot = "No"
      expand = "No">
   <cftreeitem value = "#Department#" parent = "#Department#"
      expand = "No" queryAsRoot = "No">
   <cftreeitem value = "#Phone#" display = "Phone"
      parent = "#Emp_ID#" queryAsRoot = "No"
      expand = "No">
   <cftreeitem value = "#Phone#" parent = "#Phone#"
      expand = "No" queryAsRoot = "No">
   <cftreeitem value = "#Email#" display = "Email" parent = "#Emp_ID#"
      queryAsRoot = "No" expand = "No">
```

```
   <cftreeitem value = "#Email#" parent = "#Email#" expand = "No"
      queryAsRoot = "No">
</cfloop>
</cftree>
...
```

cftreeitem

Description Populates a form tree control, created with the `cftree` tag, with elements. To display icons, you can use the `img` values that ColdFusion provides, or reference your own icons.

Category Forms tags

Syntax

```
<cftreeitem
   value = "text"
   display = "text"
   parent = "parent_name"
   img = "filename"
   imgopen = "filename"
   href = "URL"
   target = "URL_target"
   query = "queryname"
   queryAsRoot = "Yes" or "No"
   expand = "Yes" or "No">
```

See also `cfapplet`, `cfform`, `cfgrid`, `cfgridcolumn`, `cfgridrow`, `cfgridupdate`, `cfinput`, `cfselect`, `cfslider`, `cftextinput`, `cftree`

Attributes

Attribute	Req/Opt	Default	Description
value	Required		Value passed when `cfform` is submitted. When populating a tree with data from a cfquery, specify columns in a delimited list. Example: `value = "dept_id,emp_id"`
display	Optional	value	Tree item label. When populating a tree with data from a query, specify names in a delimited list. Example: `display = "dept_name,emp_name"`
parent	Optional		Value for tree item parent.

img	Optional	folder	Image name, filename, or file URL for tree item icon.
			The following values are provided:
			• cd
			• computer
			• document
			• element
			• folder
			• floppy
			• fixed
			• remote
			You can specify a custom image. To do so, include path and file extension; for example:
			`img = "../images/page1.gif"`
			To specify more than one image in a tree, or an image at the second or subsequent level, use commas to separate names, corresponding to level; for example:
			`img = "folder,document"`
			`img = ",document"` (example of second level)

Attribute	Req/Opt	Default	Description
imgopen	Optional		Icon displayed with open tree item. You can specify icon filename with a relative path. You can use a ColdFusion image.
href	Optional		URL to associate with tree item or query column for a tree that is populated from a query. If `href` is a query column, its value is the value populated by query. If `href` is not recognized as a query column, it is assumed that its text is an HTML href. When populating a tree with data from a query, HREFs can be specified in delimited list; for example: `href = "http://dept_svr,http://emp_svr"`
target	Optional		Target attribute of href URL. When populating a tree with data from a query, specify target in delimited list: `target = "FRAME_BODY,_blank"`
query	Optional		Query name to generate data for the treeitem.
queryAsRoot	Optional		Defines query as the root level. This avoids having to create another parent cftreeitem. • Yes • No • String to use as the root name If you do not specify a root name, ColdFusion returns the query name as the root.
expand	Optional	Yes	• Yes: expands tree to show tree item children • No: keeps tree item collapsed

Usage

This tag requires the client to download a Java applet. Downloading an applet takes time; therefore, using this tag might be slightly slower than using an HTML form element or the `cfinput` tag to get the same information.

For this tag to work properly. the browser must be JavaScript-enabled.

Example

```
<!--- This example shows the use of cftreeitem in cfform. Query takes
         employee list, and uses cftree and cfselect to display query results.
         Shows an alternate means of displaying the data --->
<!--- set a default for the employeeNames variable --->
<cfparam name = "employeeNames" default = "">
<!--- if an employee name has been passed from the form, set employeeNames
         variable to this value ---Auto>
<cfif IsDefined("form.employeeNames")>
   <cfset employeeNames = form.employeeNames>
</cfif>
<!--- query the datasource to find the employee information--->
<cfquery name = "GetEmployees" dataSource = "cfsnippets">
SELECT  Emp_ID, FirstName, LastName, EMail, Phone, Department
FROM   Employees
   WHERE 0=0
   <cfif employeeNames is not "">
        AND LastName = '#employeeNames#'
   </cfif>
```

```
</cfquery>

<!--- Use cfform when using other cfinput tools --->
<cfform action = "cfindex.cfm">
<!--- Use cfselect to present the contents of the query by column --->
<h3>cfselect Presentation of Data</h3>
<h4>Click an employee's last name and click "see information for this employee"
   to see expanded information.</h4>

<cfselect name = "EmployeeNames" message = "Select an Employee Name"
   size = "#getEmployees.recordcount#" query = "GetEmployees"
  value = "LastName" required = "No">
   <option value = "">Select All
</cfselect>

<input type = "Submit" name = ""
   value = "see information for this employee">
<!--- showing use of cftree for expanded presentation of data --->
<!--- Loop through the query to create each branch of the cftree --->
<h3>cftree Presentation of Data</h3>
<h4>Click the folders to "drill down" and reveal information.</h4>
<p>cftreeitem is used to create the branches of the cftree.

<p><cftree name = "SeeEmployees" height = "150" width = "240"
   font = "Arial Narrow" bold = "No" italic = "No" border = "Yes"
   hScroll = "Yes" vScroll = "Yes" required = "No" completePath = "No"
   appendKey = "Yes" highlightHref = "Yes">
<cfloop query = "GetEmployees">
   <cftreeitem value = "#Emp_ID#" parent = "SeeEmployees" expand = "No">
   <cftreeitem value = "#LastName#" display = "Name"
      parent = "#Emp_ID#" queryAsRoot = "No" expand = "No">
   <cftreeitem value = "#LastName#, #FirstName#"
      parent = "#LastName#" expand = "No" queryAsRoot = "No">
   <cftreeitem value = "#Department#" display = "Department"
      parent = "#Emp_ID#" queryAsRoot = "No" expand = "No">
   <cftreeitem value = "#Department#"
      parent = "#Department#" expand = "No" queryAsRoot = "No">
   <cftreeitem value = "#Phone#" display = "Phone"
      parent = "#Emp_ID#" queryAsRoot = "No" expand = "No">
   <cftreeitem value = "#Phone#"
      parent = "#Phone#" expand = "No" queryAsRoot = "No">
   <cftreeitem value = "#Email#" display = "Email"
      parent = "#Emp_ID#" queryAsRoot = "No" expand = "No">
   <cftreeitem value = "#Email#"
      parent = "#Email#" expand = "No" queryAsRoot = "No">
</cfloop>
</cftree>

<!--------For a more comprehensive, quicker view, you can use CFGRID ------->
<h3>cfgrid Presentation of Data</h3>
<cfgrid name="SampleGrid" width="600" query="GetEmployees" insert="No"
   delete="No" sort="No" font="Verdana" bold="No" italic="No" appendkey="No"
   highlighthref="No" griddataalign="LEFT" gridlines="Yes" rowheaders="No"
   rowheaderalign="LEFT" rowheaderitalic="No" rowheaderbold="No" colheaders="Yes"
   colheaderalign="CENTER" colheaderitalic="No" colheaderbold="No"
   bgcolor="Teal" selectmode="BROWSE" picturebar="No">
```

```
<cfgridcolumn name="LastName" header="Last Name" headeralign="LEFT"
   dataalign="LEFT" bold="No" italic="No" select="Yes" display="Yes"
   headerbold="No" headeritalic="No"> <cfgridcolumn name="FirstName"
   header="First Name" headeralign="LEFT" dataalign="LEFT" fontsize="2"
   bold="No" italic="No" select="No" display="Yes" headerbold="No"
   headeritalic="No">
<cfgridcolumn name="Email" header="Email" headeralign="LEFT" dataalign="LEFT"
   bold="No" italic="No" select="No" display="Yes" headerbold="No"
   headeritalic="No">
<cfgridcolumn name="Phone" header="Phone" headeralign="LEFT" dataalign="LEFT"
   bold="No" italic="Yes" select="No" display="Yes" headerbold="No"
   headeritalic="No"> <cfgridcolumn name="Department" header="Department"
   headeralign="LEFT" dataalign="LEFT" bold="Yes" italic="No" select="No"
   display="Yes" headerbold="No" headeritalic="No">
<cfgridcolumn name="Emp_ID" header="ID" headeralign="LEFT" dataalign="LEFT"
   width="40" bold="No" italic="No" select="No" display="Yes" headerbold="No"
   headeritalic="No">
</cfgrid>
</cfform>
```

cftry

Description

Used with one or more cfcatch tags. Together, they catch and process exceptions in ColdFusion pages. **Exceptions** are events that disrupt the normal flow of instructions in a ColdFusion page, such as failed database operations, missing include files, and developer-specified events.

Category

Exception handling tags

Syntax

```
<cftry>
   code here
<cfcatch type = "exceptiontype">
   Exception processing code here
</cfcatch>
   Optional: More cfcatch blocks here
</cftry>
```

See also

cfcatch, cferror, cfrethrow, cfthrow

History

New in ColdFusion MX: The SQLSTATE return value in a cfcatch tag depends on the database driver type:

- Type 1 (JDBC-ODBC bridge): the value is the same as in ColdFusion 5
- Type 4 (100% Java, no native methods): the value might be different

If your application depends on SQLSTATE values for flow control, the application might produce unexpected behavior with ColdFusion MX.

New in ColdFusion MX: it is not necessary, when you include a cfcatch tag with type="any", to do so in the last cfcatch tag in the block, to ensure that all other tests are executed before it. ColdFusion finds the best-match cfcatch block.

New in ColdFusion MX: the cftry and cfcatch tags can be used as cfscript constructs, and for handling component method invocation errors.

New in ColdFusion MX: you cannot modify the structure returned by cfcatch.

New in ColdFusion MX: The cfcollection, cfindex, and cfsearch tags can throw the SEARCHENGINE exception. In earlier releases, an error in processing these tags threw only an UNKNOWN exception.

Attributes

Attribute	Req/Opt	Default	Description
type	Optional	Any	• application: catches application exceptions • database: catches database exceptions • template: catches ColdFusion page exceptions • security: catches security exceptions • object: catches object exceptions • missinginclude: catches missing include file exceptions • expression: catches expression exceptions • lock: catches lock exceptions • custom_type: catches developer-defined exceptions, defined in the cfthrow tag • searchengine: catches Verity search engine exceptions • any: catches all exception types

Usage

Within a cftry block, you must code at least one cfcatch tag. Put cfcatch tags at the end of a cftry block. ColdFusion tests cfcatch tags in the order in which they appear.

If type="any", ColdFusion Server catches exceptions from any CFML tag, data source, or external object. To get the exception type use code such as the following:

```
#cfcatch.type#
```

Applications can use the cfthrow tag to throw developer-defined exceptions. Catch these exceptions with any of these type options:

- "*custom_type*"
- "Application"
- "Any"

The custom_type type is a developer-defined type specified in a cfthrow tag. If you define a custom type as a series of strings concatenated by periods (for example, "MyApp.BusinessRuleException.InvalidAccount"), the cfcatch tag can catch the custom type by its character pattern. The tag searches for a matching exception type, starting with the most specific (the entire string), and ending with the least specific.

For example, you could define a type as follows:

```
<cfthrow type = "MyApp.BusinessRuleException.InvalidAccount">
```

The cfcatch tag first searches the cfthrow tag for the entire type string:

```
<cfcatch type = "MyApp.BusinessRuleException.InvalidAccount">
```

If it does not find a match, it searches for the next most specific:

```
<cfcatch type = "MyApp.BusinessRuleException">
```

Finally, it searches for the least specific:

```
<cfcatch type = "MyApp">
```

You can code cfcatch tags in any order to catch a custom exception type.

If you specify type = "Application", the cfcatch tag catches only custom exceptions that have the Application type in the cfthrow tag that defines them.

The cfinclude, cfmodule, and cferror tags throw an exception of type = "template".

An exception that is thrown within a cfcatch block cannot be handled by the cftry block that immediately encloses the cfcatch tag. However, you can rethrow the currently active exception with the cfrethrow tag.

The following example shows how to throw an exception from a component method:

```
<cfcomponent>
  <cffunction name="getEmp">
  <cfargument name="lastName" required="yes">
      <cfquery name="empQuery" datasource="ExampleApps" >
        SELECT LASTNAME, FIRSTNAME, EMAIL
        FROM tblEmployees
        WHERE LASTNAME LIKE '#arguments.lastName#'
      </cfquery>
      <cfif empQuery.recordcount LT 1>
        <cfthrow type="noQueryResult"
          message="No results were found. Please try again.">
```

```
        <cfelse>
          <cfreturn empQuery>
        </cfif>
    </cffunction>
</cfcomponent>
```

For an explanation of the example and more information, see the "Building and Using ColdFusion Components" chapter in *Developing ColdFusion Applications*.

The `cfcatch` variables provide the following exception information:

cfcatch variable	Content
`cfcatch.type`	Type: Exception type, as specified in cfcatch.
`cfcatch.message`	Message: Exception's diagnostic message, if provided; otherwise, an empty string; in the cfcatch.message variable
`cfcatch.detail`	Detailed message from CFML interpreter. The message contains HTML formatting. It can help determine which tag threw the exception.
`cfcatch.tagcontext`	Tag stack name, position of each tag in the stack, and absolute pathnames of the files that contain the tags in the stack.
`cfcatch.NativeErrorCode`	Applies to `type = "database"`. Native error code associated with exception. Database drivers typically provide error codes to diagnose failing database operations. Default: -1.
`cfcatch.SQLState`	Applies to `type ="database"`. SQLState associated with exception. Database drivers typically provide error codes to help diagnose failing database operations. Default: -1.
`cfcatch.ErrNumber`	Applies to `type="expression"`. Internal expression error number.
`cfcatch.MissingFileName`	Applies to `type="missingInclude"`. Name of file that could not be included.
`cfcatch.LockName`	Applies to `type="lock"`. Name of affected lock (if the lock is unnamed, the value is "`anonymous`").
`cfcatch.LockOperation`	Applies to `type="lock"`. Operation that failed (Timeout, Create Mutex, or Unknown).
`cfcatch.ErrorCode`	Applies to `type="custom"`. String error code.
`cfcatch.ExtendedInfo`	Applies to `type="application"` and "`custom`". Custom error message; information that the default exception handler does not display.

Advanced Exception types

You can specify the following advanced exception types in the `type` attribute:

ColdFusion Advanced Exception type
COM.Allaire.ColdFusion.CFEXECUTE.OutputError
COM.Allaire.ColdFusion.CFEXECUTE.Timeout
COM.Allaire.ColdFusion.FileException
COM.Allaire.ColdFusion.HTTPAccepted
COM.Allaire.ColdFusion.HTTPAuthFailure
COM.Allaire.ColdFusion.HTTPBadGateway
COM.Allaire.ColdFusion.HTTPBadRequest
COM.Allaire.ColdFusion.HTTPCFHTTPRequestEntityTooLarge
COM.Allaire.ColdFusion.HTTPCGIValueNotPassed
COM.Allaire.ColdFusion.HTTPConflict
COM.Allaire.ColdFusion.HTTPContentLengthRequired
COM.Allaire.ColdFusion.HTTPContinue
COM.Allaire.ColdFusion.HTTPCookieValueNotPassed
COM.Allaire.ColdFusion.HTTPCreated
COM.Allaire.ColdFusion.HTTPFailure
COM.Allaire.ColdFusion.HTTPFileInvalidPath
COM.Allaire.ColdFusion.HTTPFileNotFound
COM.Allaire.ColdFusion.HTTPFileNotPassed
COM.Allaire.ColdFusion.HTTPFileNotRenderable
COM.Allaire.ColdFusion.HTTPForbidden
COM.Allaire.ColdFusion.HTTPGatewayTimeout
COM.Allaire.ColdFusion.HTTPGone
COM.Allaire.ColdFusion.HTTPMethodNotAllowed
COM.Allaire.ColdFusion.HTTPMovedPermanently
COM.Allaire.ColdFusion.HTTPMovedTemporarily
COM.Allaire.ColdFusion.HTTPMultipleChoices
COM.Allaire.ColdFusion.HTTPNoContent
COM.Allaire.ColdFusion.HTTPNonAuthoritativeInfo
COM.Allaire.ColdFusion.HTTPNotAcceptable
COM.Allaire.ColdFusion.HTTPNotFound
COM.Allaire.ColdFusion.HTTPNotImplemented

ColdFusion Advanced Exception type
COM.Allaire.ColdFusion.HTTPNotModified
COM.Allaire.ColdFusion.HTTPPartialContent
COM.Allaire.ColdFusion.HTTPPaymentRequired
COM.Allaire.ColdFusion.HTTPPreconditionFailed
COM.Allaire.ColdFusion.HTTPProxyAuthenticationRequired
COM.Allaire.ColdFusion.HTTPRequestURITooLarge
COM.Allaire.ColdFusion.HTTPResetContent
COM.Allaire.ColdFusion.HTTPSeeOther
COM.Allaire.ColdFusion.HTTPServerError
COM.Allaire.ColdFusion.HTTPServiceUnavailable
COM.Allaire.ColdFusion.HTTPSwitchingProtocols
COM.Allaire.ColdFusion.HTTPUnsupportedMediaType
COM.Allaire.ColdFusion.HTTPUrlValueNotPassed
COM.Allaire.ColdFusion.HTTPUseProxy
COM.Allaire.ColdFusion.HTTPVersionNotSupported
COM.Allaire.ColdFusion.POPAuthFailure
COM.Allaire.ColdFusion.POPConnectionFailure
COM.Allaire.ColdFusion.POPDeleteError
COM.Allaire.ColdFusion.Request.Timeout
COM.Allaire.ColdFusion.SERVLETJRunError
COMCOM.Allaire.ColdFusion.HTTPConnectionTimeout

This tag requires an end tag.

Example

```
<!--- cftry example, using TagContext to display the tag stack. --->
<h3>cftry Example</h3>
<!--- open a cftry block --->
<cftry>
   <!--- note misspelled tablename "employees" as "employeeas" --->
   <cfquery name = "TestQuery" dataSource = "cfsnippets">
      SELECT *
      FROM EMPLOYEEAS
   </cfquery>

   <!--- <p>... other processing goes here --->
   <!--- specify the type of error for which we search --->
   <cfcatch type = "Database">
   <!--- the message to display --->
      <h3>You've Thrown a Database <b>Error</b></h3>
      <cfoutput>
         <!--- and the diagnostic message from the ColdFusion server --->
         <p>#cfcatch.message#</p>
         <p>Caught an exception, type = #CFCATCH.TYPE# </p>
         <p>The contents of the tag stack are:</p>
         <cfloop index = i from = 1
              to = #ArrayLen(CFCATCH.TAGCONTEXT)#>
           <cfset sCurrent = #CFCATCH.TAGCONTEXT[i]#>
           <br>#i# #sCurrent["ID"]#
              (#sCurrent["LINE"]#,#sCurrent["COLUMN"]#)
              #sCurrent["TEMPLATE"]#
         </cfloop>
      </cfoutput>
   </cfcatch>
</cftry>
```

cfupdate

Description Updates records in a data source from data in a ColdFusion form or form Scope.

Category Database manipulation tags

Syntax
```
<cfupdate
   dataSource = "ds_name"
   tableName = "table_name"
   tableOwner = "name"
   tableQualifier = "qualifier"
   username = "username"
   password = "password"
   formFields = "field_names">
```

See also cfinsert, cfprocparam, cfprocresult, cfquery, cfqueryparam, cfstoredproc, cftransaction

History New in ColdFusion MX: The connectString, dbName, dbServer, dbtype, provider and providerDSN attributes are deprecated. Do not use them. They do not work, and might cause an error, in releases later than ColdFusion 5.

Attributes

Attribute	Req/Opt	Default	Description
dataSource	Required		Name of the data source that contains the table.
tableName	Required		Name of table to update. • For ORACLE drivers, must be uppercase. • For Sybase driver: case-sensitive; must be in same case as used when the table was created
tableOwner	Optional		For data sources that support table ownership (for example, SQL Server, Oracle, Sybase SQL Anywhere), the table owner.
tableQualifier	Optional		For data sources that support table qualifiers. The purpose of table qualifiers is as follows: • SQL Server and Oracle: name of database that contains table • Intersolv dBASE driver: directory of DBF files
username	Optional		Overrides username value specified in ODBC setup.
password	Optional		Overrides password value specified in ODBC setup.
formFields	Optional	(all on form, except keys)	Comma-delimited list of form fields to update. If a form field is not matched by a column name in the database, ColdFusion throws an error. The database table key field must be present in the form. It can be hidden.

Example

```
<!--- This example shows the use of CFUPDATE to change
records in a datasource. --->
<!--- if course_ID has been passed to this form, then
perform the update on that record in the datasource --->

<cfif IsDefined("form.Course_ID")>
   <!--- check that course_id is numeric --->
      <cfif Not IsNumeric(form.Course_ID)>
         <cfabort>
      </cfif>
   <!--- Now, do the update --->
   <cfupdate datasource="cfsnippets"
      tablename="Courses"
      formfields="Course_ID,Number,Descript">
</cfif>

<!--- Perform a query to reflect any updated information if Course_ID is passed
through a url, we are selecting a record to update ... select only that
record with the WHERE clause. --->
<cfquery name="GetCourseInfo" DATASOURCE="cfsnippets">
   SELECT Course_Number, Course_ID, Descript
   FROM Courses
      <cfif IsDefined("url.Course_ID")>
         WHERE Course_ID = #Trim(url.Course_ID)#
      </cfif>
   ORDER by Course_Number
</cfquery>
<html>
<head>
   <title>CFUPDATE Example</title>
   <cfset css_path = "../../css">
   <cfinclude template="../../resource/include/mm_browsersniff.cfm">
</head>
<body>

<H3>CFUPDATE Example</H3>
<!--- If we are updating a record, don't show the entire list. --->
<cfif IsDefined("url.Course_ID")>
   <form method="post" action="index.cfm">
   <H3>You can alter the contents of this record, and then click "Update"
      to use CFUPDATE and alter the database</H3>
   <P>Course Number <INPUT TYPE="Text" name="Number"
         value="<cfoutput>#Trim(GetCourseInfo.Course_Number)#</cfoutput>">
   <P>Course Description<BR>
   <textarea name="Descript" cols="40" rows="5">
      <cfoutput>#Trim(GetCourseInfo.Descript)#</cfoutput>
   </textarea><br>
   <input type="Hidden" NAME="Course_ID"
      value="<cfoutput>#Trim(GetCourseInfo.Course_ID)#</cfoutput>">
   <p><input type="Submit" value="Click to Update">
   </form>
```

```
<cfelse>
   <!--- Show the entire record set in CFTABLE form --->
   <cftable query="GetCourseInfo" htmltable colheaders>
      <cfcol text="<a href='index.cfm?Course_ID=#Trim(Course_ID)#'>Edit Me</a>"
         width=10 header="Edit<br>this Entry">
      <cfcol text="#Trim(Course_Number)#" WIDTH="4" HEADER="Course Number">
      <cfcol text="#Trim(Descript)#" WIDTH=100 HEADER="Course Description">
   </cftable>
</cfif>
</body>
</html>
```

cfwddx

Description

Serializes and deserializes CFML data structures to the XML-based WDDX format. The WDDX is an XML vocabulary for describing complex data structures in a standard, generic way. Implementing it lets you use the HTTP protocol to such information among application server platforms, application servers, and browsers.

This tag generates JavaScript statements to instantiate JavaScript objects equivalent to the contents of a WDDX packet or CFML data structure. Interoperates with Unicode.

Category

Extensibility tags

Syntax

```
<cfwddx
   action = "action"
   input = "inputdata"
   output = "resultvariablename"
   topLevelVariable = "toplevelvariablenameforjavascript"
   useTimeZoneInfo = "Yes" or "No"
   validate = "Yes" or "No" >
```

See also

cfcollection, cfdump, cfexecute, cfindex, cfobject, cfreport, cfsearch

History

New in ColdFusion MX: ColdFusion preserves the case of column names in JavaScript. (Earlier releases converted query column names to lowercase.)

New in ColdFusion MX: This tag supports several encoding formats. The default encoding format is UTF-8. The tag interoperates with Unicode.

Attributes

Attribute	Req/Opt	Default	Description
action	Required		• cfml2wddx: serialize CFML to WDDX • wddx2cfml: deserialize WDDX to CFML • cfml2js: serialize CFML to JavaScript • wddx2js: deserialize WDDX to JavaScript
input	Required		A value to process
output	Required if `action = "wddx2cfml"`		Name of variable for output. If `action = "WDDX2JS"` or `"CFML2JS"`, and this attribute is omitted, result is output in HTML stream.
topLevelVariable	Required if `action = "wddx2js"` or `"cfml2js"`		Name of top-level JavaScript object created by deserialization. The object is an instance of the `WddxRecordset` object.
useTimeZoneInfo	Optional	Yes	Whether to output time-zone information when serializing CFML to WDDX. • Yes: the hour-minute offset, represented in ISO8601 format, is output. • No: the local time is output.

Attribute	Req/Opt	Default	Description
validate	Optional	No	Applies if `action = "wddx2cfml"` or `"wddx2js"`. • yes: validates WDDX input with an XML parser using WDDX DTD. If parser processes input without error, packet is deserialized. Otherwise, an error is thrown. • no: no input validation

Usage

ColdFusion preserves the case of column names cases in JavaScript.

The `wddx2js` and `cfml2js` actions create a WddxRecordset javascript object when they encounter a RecordSet java object. The serialized JavaScript code requires a wddx.js file.

This tag performs the following conversions:

From	To
CFML	WDDX
CFML	JavaScript
WDDX	CFML
WDDX	JavaScript

For more information, and a list of the ColdFusion array and structure functions that you can use to manage XML document objects and functions, see *Developing ColdFusion MX Applications with CFML*.

Example

```
<!--- This shows basic use of the cfwddx tag. --->
<html>
<body>
<!--- Create a simple query --->
<cfquery name = "q" dataSource = "cfsnippets">
   select Message_Id, Thread_id, Username from messages
</cfquery>

The recordset data is:...<p>
<cfoutput query = q>
   #Message_ID# #Thread_ID# #Username#<br>
</cfoutput><p>

<!--- Serialize data to WDDX format --->
Serializing CFML data...<p>
<cfwddx action = "cfml2wddx" input = #q# output = "wddxText">

<!--- Display WDDX XML packet --->
Resulting WDDX packet is:
<xmp><cfoutput>#wddxText#</cfoutput></xmp>

<!--- Deserialize to a variable named wddxResult --->
Deserializing WDDX packet...<p>
<cfwddx action = "wddx2cfml" input = #wddxText# output = "qnew">
```

```
The recordset data is:...<p>
<cfoutput query = qnew>
   #Message_ID# #Thread_ID# #Username#<br>
</cfoutput><p>
```

cfxml

Description Creates a ColdFusion XML document object that contains the markup in the tag body. This tag can include XML and CFML tags. ColdFusion processes the CFML code in the tag body, then assigns the resulting text to an XML document object variable.

Category Extensibility tags

Syntax
```
<CFXML
   variable="xmlVarName"
   caseSensitive="yes" or "no">
```

See also IsXmlDoc, IsXmlElement, IsXmlRoot, XmlChildPos, XmlNew, XmlParse, XmlSearch, XmlTransform

History New in ColdFusion MX: This tag is new.

Attributes

Attribute	Req/Opt	Default	Description
variable			name of an xml variable
caseSensitive	Optional	no	• yes: maintains the case of document elements and attributes • no

Usage An XML document object is represented in ColdFusion as a structure.

The following example creates a document object whose root element is MyDoc. The object includes text that displays the value of the ColdFusion variable testVar. The code creates four nested child elements, which are generated by an indexed `cfloop` tag. The `cfdump` tag displays the XML document object.

Example
```
<cfset testVar = True>
<cfxml variable="MyDoc">
   <MyDoc>
      <cfif testVar IS True>
         <cfoutput>The value of testVar is True.</cfoutput>
      <cfelse>
         <cfoutput>The value of testVar is False.</cfoutput>
      </cfif>
      <cfloop index = "LoopCount" from = "1" to = "4">
         <childNode>
            This is Child node <cfoutput>#LoopCount#.</cfoutput>
         </childNode>
      </cfloop>
   </MyDoc>
</cfxml>
<cfdump var=#MyDoc#>
```

CHAPTER 4
ColdFusion Function Summary

This chapter lists and categorizes ColdFusion Markup Language (CFML) functions.

Contents

New functions, parameters, and values

Function	Parameter or value	New in this ColdFusion Server release
CreateObject	All	ColdFusion MX
GetPageContext	All	ColdFusion MX
GetMetaData	All	ColdFusion MX
GetProfileSections	All	ColdFusion MX
IsK2ServerABroker	All	ColdFusion MX
IsK2ServerOnline	All	ColdFusion MX
IsObject	All	ColdFusion MX
IsProtected	All	ColdFusion MX
IsUserInRole	All	ColdFusion MX
IsXmlDoc	All	ColdFusion MX
IsXmlElement	All	ColdFusion MX
IsXmlRoot	All	ColdFusion MX
SetEncoding	All	ColdFusion MX
SetLocale	`variant` attribute	ColdFusion MX
URLDecode	`charset` attribute	ColdFusion MX
URLEncodedFormat	`charset` attribute	ColdFusion MX
XmlChildPos	All	ColdFusion MX
XmlElemNew	All	ColdFusion MX
XmlFormat	All	ColdFusion MX
XmlNew	All	ColdFusion MX
XmlParse	All	ColdFusion MX
XmlSearch	All	ColdFusion MX
XmlTransform	All	ColdFusion MX

Deprecated functions, parameters, and values

The following functions, parameters, and values are deprecated. Do not use them in ColdFusion applications. They might not work, and might cause an error, in releases later than ColdFusion MX.

Function	Parameter or value	Deprecated as of this ColdFusion Server release
getMetricData	cachepops parameter	ColdFusion MX
getK2ServerCollections	All	ColdFusion MX
getTemplatePath	All	ColdFusion MX
parameterExists	All	ColdFusion MX. Use the IsDefined function.
setLocale	locale = "Spanish (Mexican)" value	ColdFusion MX. Use Spanish (Standard).
URLEncodedFormat	All	ColdFusion MX

Obsolete functions, parameters, and values

The following functions, parameters, and values are obsolete. Do not use them in ColdFusion applications. They might not work, and might cause an error, in releases later than ColdFusion 5.

Function	Parameter or value	Deprecated as of this ColdFusion Server release
AuthenticatedContext	All	ColdFusion MX
AuthenticatedUser	All	ColdFusion MX
isAuthenticated	All	ColdFusion MX
isAuthorized	All	ColdFusion MX
isProtected	All	ColdFusion MX

Function summary

ColdFusion Markup Language (CFML) includes a set of functions that you use in ColdFusion pages to perform logical and arithmetic operations and manipulate data.

The following table lists CFML functions:

Abs
ACos
ArrayAppend
ArrayAvg
ArrayClear
ArrayDeleteAt
ArrayInsertAt
ArrayIsEmpty
ArrayLen
ArrayMax
ArrayMin
ArrayNew
ArrayPrepend
ArrayResize
ArraySet
ArraySort
ArraySum
ArraySwap
ArrayToList
Asc
ASin
Atn
BitAnd
BitMaskClear
GetHttpTimeString
GetK2ServerDocCount
GetK2ServerDocCountLimit
GetLocale
GetMetaData
GetMetricData
GetPageContext
GetProfileSections
GetProfileString
GetServiceSettings
GetTempDirectory
GetTempFile
GetTemplatePath
GetTickCount
GetTimeZoneInfo
GetToken
Hash
Hour
HTMLCodeFormat
HTMLEditFormat
IIf
IncrementValue
InputBaseN
Insert
Max
Mid
Min
Minute
Month
MonthAsString
Now
NumberFormat
ParagraphFormat
ParameterExists
ParseDateTime
Pi
PreserveSingleQuotes
Quarter
QueryAddColumn
QueryAddRow
QueryNew
QuerySetCell
QuotedValueList
Rand
Randomize
RandRange
REFind
REFindNoCase

BitMaskRead
BitMaskSet
BitNot
BitOr
BitSHLN
BitSHRN
BitXor
Ceiling
Chr
CJustify
Compare
CompareNoCase
Cos
CreateDate
CreateDateTime
CreateObject
CreateODBCDate
CreateODBCDateTime
CreateODBCTime
CreateTime
CreateTimeSpan
CreateUUID
DateAdd
DateCompare
DateConvert
DateDiff
DateFormat
Int
IsArray
IsBinary
IsBoolean
IsCustomFunction
IsDate
IsDebugMode
IsDefined
IsK2ServerABroker
IsK2ServerDocCountExceeded
IsK2ServerOnline
IsLeapYear
IsNumeric
IsNumericDate
IsObject
IsQuery
IsSimpleValue
IsStruct
IsUserInRole
IsWDDX
IsXmlDoc
IsXmlElement
IsXmlRoot
JavaCast
JSStringFormat
LCase
Left
RemoveChars
RepeatString
Replace
ReplaceList
ReplaceNoCase
REReplace
REReplaceNoCase
Reverse
Right
RJustify
Round
RTrim
Second
SetEncoding
SetLocale
SetProfileString
SetVariable
Sgn
Sin
SpanExcluding
SpanIncluding
Sqr
StripCR
StructAppend
StructClear
StructCopy
StructCount

DatePart
Day
DayOfWeek
DayOfWeekAsString
DayOfYear
DaysInMonth
DaysInYear
DE
DecimalFormat
DecrementValue
Decrypt
DeleteClientVariable
DirectoryExists
DollarFormat
Duplicate
Encrypt
Evaluate
Exp
ExpandPath
FileExists
Find
FindNoCase
FindOneOf
FirstDayOfMonth
Fix
FormatBaseN
GetAuthUser
Len
ListAppend
ListChangeDelims
ListContains
ListContainsNoCase
ListDeleteAt
ListFind
ListFindNoCase
ListFirst
ListGetAt
ListInsertAt
ListLast
ListLen
ListPrepend
ListQualify
ListRest
ListSetAt
ListSort
ListToArray
ListValueCount
ListValueCountNoCase
LJustify
Log
Log10
LSCurrencyFormat
LSDateFormat
LSEuroCurrencyFormat
StructDelete
StructFind
StructFindKey
StructFindValue
StructGet
StructInsert
StructIsEmpty
StructKeyArray
StructKeyExists
StructKeyList
StructNew
StructSort
StructUpdate
Tan
TimeFormat
ToBase64
ToBinary
ToString
Trim
UCase
URLDecode
URLEncodedFormat
URLSessionFormat
Val
ValueList
Week
WriteOutput

GetBaseTagData
GetBaseTagList
GetBaseTemplatePath
GetClientVariablesList
GetCurrentTemplatePath
GetDirectoryFromPath
GetException
GetFileFromPath
GetFunctionList
GetHttpRequestData
LSIsCurrency
LSIsDate
LSIsNumeric
LSNumberFormat
LSParseCurrency
LSParseDateTime
LSParseEuroCurrency
LSParseNumber
LSTimeFormat
LTrim
XmlChildPos
XmlElemNew
XmlFormat
XmlNew
XmlParse
XmlSearch
XmlTransform
Year
YesNoFormat

Array functions

ArrayAppend
ArrayAvg
ArrayClear
ArrayDeleteAt
ArrayInsertAt
ArrayIsEmpty
ArrayLen
ArrayMax
ArrayMin
ArrayNew
ArrayPrepend
ArrayResize
ArraySet
ArraySort
ArraySum
ArraySwap
ArrayToList
IsArray
ListToArray

Authentication functions

GetAuthUser
IsUserInRole

Conversion functions

ArrayToList
Hash
URLEncodedFormat
XmlTransform
ToBase64
LCase
Val
ToBinary
ListToArray
XmlFormat
ToString
URLDecode
XmlParse

Date and time functions

CreateDate
DateFormat
GetTimeZoneInfo
MonthAsString
CreateDateTime
DatePart
Hour
Now
CreateODBCDate
Day
IsDate
ParseDateTime
CreateODBCDateTime
DayOfWeek
IsLeapYear
Quarter
CreateODBCTime
DayOfWeekAsString
IsNumericDate
Second
CreateTime
DayOfYear
LSDateFormat
TimeFormat
CreateTimeSpan
DaysInMonth
LSIsDate
Week
DateAdd
DaysInYear
LSParseDateTime
Year
DateCompare
FirstDayOfMonth
LSTimeFormat
DateConvert
GetHttpTimeString
Minute
DateDiff
GetTickCount
Month

Decision functions

DirectoryExists
FileExists
Iif
IsArray
IsAuthenticated
IsBinary
IsBoolean
IsCustomFunction
IsDate
IsDebugMode
IsDefined
IsK2ServerABroker
IsK2ServerDocCount Exceeded
IsK2ServerOnline
IsLeapYear
IsNumeric
IsNumericDate
IsObject
IsQuery
IsSimpleValue
IsStruct
IsUserInRole
IsWDDX
IsXmlDoc
IsXmlElement
IsXmlRoot
LSIsCurrency
LSIsDate
LSIsNumeric
StructIsEmpty
StructKeyExists
YesNoFormat

Display and formatting functions

CJustify
DateFormat
DecimalFormat
DollarFormat
FormatBaseN
GetLocale
HTMLCodeFormat
HTMLEditFormat
LJustify
LSCurrencyFormat
LSDateFormat
LSEuroCurrencyFormat
LSIsCurrency
LSIsDate
LSNumberFormat
LSParseCurrency
LSParseDateTime
LSParseEuroCurrency
LSParseNumber
LSTimeFormat
NumberFormat
ParagraphFormat
RJustify
SetEncoding
StripCR
TimeFormat
YesNoFormat

Dynamic evaluation functions

DE
Evaluate
Iif
SetVariable

Extensibility functions

CreateObject
XmlChildPos
XmlElemNew
XmlFormat
XmlNew
XmlParse
XmlSearch
XmlTransform

Full-text search functions

GetK2ServerDocCount
IsK2ServerABroker
IsK2ServerDocCountExceeded
IsK2ServerOnline
GetK2ServerDocCountLimit

International functions

DateConvert
LSCurrencyFormat
LSIsNumeric
LSParseNumber
GetHttpTimeString
LSDateFormat
LSNumberFormat
LSTimeFormat
GetLocale
LSEuroCurrencyFormat
LSParseCurrency
GetTimeZoneInfo
LSIsDate
SetLocale
LSIsCurrency
LSParseDateTime
LSParseEuroCurrency

List functions

ArraySort
FindOneOf
ListDeleteAt
ListSetAt
ArrayToList
FormatBaseN
ListFind
ListSort
Asc
ListContainsNoCase
ListFindNoCase
ListToArray
Chr
GetClientVariablesList
ListFirst
ListValueCount
CJustify
LCase
ListGetAt
ListValueCountNoCase
Compare
Left
ListInsertAt
ReplaceList
CompareNoCase
Len
ListLast
Decrypt
ListAppend
ListLen
Encrypt
ListChangeDelims
ListPrepend
Find
ListContains
ListQualify
FindNoCase
ListContainsNoCase
ListRest

Mathematical functions

Abs
ACos
ArrayAvg
ArraySum
ASin
Atn
BitAnd
BitMaskClear
BitMaskRead
BitMaskSet
BitNot
BitOr
BitSHLN
BitSHRN
BitXor
Ceiling
Cos
DecrementValue
Exp
Fix
FormatBaseN
IncrementValue
InputBaseN
Int
Log
Log10
Max
Min
Pi
Rand
Randomize
RandRange
Round
Sgn
Sin
Sqr
Tan

Other functions

CreateUUID
Decrypt
DeleteClientVariable
Duplicate
Encrypt
GetBaseTagData
GetBaseTagList
GetBaseTemplatePath
GetClientVariablesList
GetTickCount
Hash
PreserveSingleQuotes
QuotedValueList
StripCR
ToBase64
ToBinary
ToString
URLDecode
URLEncodedFormat
URLSessionFormat
ValueList
WriteOutput

Query functions

IsQuery
QueryAddColumn
QueryAddRow
QueryNew
QuerySetCell
QuotedValueList
ValueList

String functions

History New in ColdFusion MX: ColdFusion supports the Java UCS-2 representation of Unicode character values 0–65535. (Earlier releases supported ASCII values.)

String-processing functions process any of these characters (including ASCII 0 (NUL) characters), and continue counting subsequent characters of the string, if any. (In earlier releases, some string-processing functions processed the ASCII 0 (NUL) character, but did not process subsequent characters of the string.)

Asc	JavaCast	LTrim	Right
Chr	JSStringFormat	Mid	RJustify
CJustify	LCase	MonthAsString	RTrim
Compare	Left	ParagraphFormat	SpanExcluding
CompareNoCase	Len	ParseDateTime	SpanIncluding
DayOfWeekAsString	LJustify	REFind	StripCR
Decrypt	ListValueCount	REFindNoCase	ToBase64
Encrypt	ListValueCountNoCase	RemoveChars	ToBinary
FormatBaseN	LSIsCurrency	RepeatString	ToString
Find	LSIsDate	Replace	Trim
FindNoCase	LSIsNumeric	ReplaceList	UCase
FindOneOf	LSParseCurrency	ReplaceNoCase	URLDecode
GetToken	LSParseDateTime	REReplace	URLEncodedFormat
Hash	LSParseEuroCurrency	REReplaceNoCase	Val
Insert	LSParseNumber	Reverse	XmlFormat

See also "Conversion functions" on page 329.

Structure functions

Duplicate
IsStruct
StructAppend
StructClear
StructCopy
StructCount
StructDelete
StructFind
StructFindKey
StructFindValue
StructGet
StructInsert
StructIsEmpty
StructKeyArray
StructKeyExists
StructKeyList
StructNew
StructSort
StructUpdate

System functions

DirectoryExists
ExpandPath
FileExists
GetBaseTemplatePath
GetCurrentTemplatePath
GetDirectoryFromPath
GetException
GetFileFromPath
GetFunctionList
GetHttpRequestData
GetLocale
GetMetaData
GetMetricData
GetProfileString
GetTempDirectory
GetTempFile
GetTemplatePath
SetEncoding
SetProfileString
GetPageContext
GetProfileSections
GetServiceSettings
SetLocale

XML functions

IsXmlDoc
XmlChildPos
XmlElemNew
XmlFormat
XmlNew
XmlParse
XmlSearch
XmlTransform

CHAPTER 5
ColdFusion Functions

This chapter describes ColdFusion Markup Language (CFML) functions. The functions are listed alphabetically.

Abs

Description: Absolute-value function. The absolute value of a number is the number without its sign.

Return value: The absolute value of a number.

Category: Mathematical functions

Syntax: `Abs(number)`

See also: Sgn

Parameters:

Parameter	Description
number	A number

Example:

```
<h3>Abs Example</h3>
<p>The absolute value of the following numbers:
1,3,-4,-3.2,6 is
<cfoutput>
#Abs(1)#,#Abs(3)#,#Abs(-4)#,#Abs(-3.2)#,#Abs(6)#
</cfoutput>

<p>The absolute value of a number is the number without its sign.
```

ACos

Description Arccosine function. The arccosine is the angle whose cosine is *number*.

Return value The arccosine of a number, in radians.

Category Mathematical functions

Syntax `ACos(number)`

See also `Cos`, `Sin`, `ASin`, `Tan`, `Pi`

Parameters

Parameter	Description
number	Cosine of an angle. The value must be between -1.0 and 1.0, inclusive.

Usage The range of the result is 0 to π.

To convert degrees to radians, multiply degrees by π/180. To convert radians to degrees, multiply radians by 180/π.

Example

```
<h3>ACos Example</h3>
<!--- output its arccosine value --->
<cfif IsDefined("FORM.CosNum")>
   <cfif IsNumeric(FORM.CosNum)>
      <cfif FORM.CosNum LESS THAN OR EQUAL TO 1>
         <cfif FORM.CosNum GREATER THAN OR EQUAL TO -1>
            ACos(<cfoutput>#FORM.CosNum#</cfoutput>) = <cfoutput>#ACos(FORM.cosNum)#
           Radians</cfoutput>
            <br> or    <br>
            ACos(<cfoutput>#FORM.CosNum#</cfoutput>) =
            <cfoutput>#Evaluate(ACos(FORM.cosNum) * 180/PI())# Degrees</cfoutput>
         <cfelse>
            <!--- if it is empty, output an error message --->
            <h4>Enter a number between -1 and 1</h4>
            </cfif>
      <cfelse>
         <!--- if it is empty, output an error message --->
         <h4>Enter a number between -1 and 1</h4>
      </cfif>
   </cfif>
</cfif>

<form action = "acos.cfm">
<p>Enter a number to get its arccosine in Radians and Degrees.
<br><input type = "Text" name = "cosNum" size = "25">
<p><input type = "Submit" name = ""> <input type = "RESET">
</form>
```

ArrayAppend

Description: Appends an array element to an array.

Return value: True, on successful completion.

Category: Array functions

Syntax: `ArrayAppend(array, value)`

See also: ArrayPrepend

History: New in ColdFusion MX: this function can be used on XML objects.

Parameters

Parameter	Description
array	Name of an array
value	Value to add at end of array

Example

```
<h3>ArrayAppend Example</h3>
<cfquery name = "GetEmployeeNames" datasource = "cfsnippets">
SELECT FirstName, LastName FROM Employees
</cfquery>
<!--- create an array --->
<cfset myArray = ArrayNew(1)>
<!--- set element one to show where we are --->
<cfset myArray[1] = "Test Value">
<!--- loop through the query; append these names successively to the last
         element --->
<cfloop query = "GetEmployeeNames">
   <cfoutput>#ArrayAppend(myArray, "#FirstName# #LastName#")#
   </cfoutput>, Array was appended<br>
</cfloop>
<!--- show the resulting array as a list --->
<cfset myList = ArrayToList(myArray, ",")>
<!--- output the array as a list --->
<cfoutput>
   <p>The contents of the array are as follows:
   <p>#myList#
</cfoutput>
```

ArrayAvg

Description Calculates the average of the values in an array.

Return value Number. If the `array` parameter value is an empty array, returns zero.

Category Array functions, Mathematical functions

Syntax `ArrayAvg(array)`

See also `ArraySum`

Parameters

Parameter	Description
array	Name of an array

Usage The following example uses the ColdFusion built-in variable `Form.fieldNames`, which is available on the action page of a form. It contains a comma-delimited list of the names of the fields on the form.

Example

```
<!--- This example shows the use of ArrayAvg --->
<!-- The following lines of code keep track of the form fields that can
be dynamically generated on the screen. It uses the Fieldnames variable
with the ListLen function to determine the number of fields on the form. --->
<cfset FormElem = 2>
   <cfif Isdefined("Form.Submit")>
      <cfif Form.Submit is "More">
         <cfset FormElem = ListLen(Form.Fieldnames)>
      </cfif>
   </cfif>

<html>
<head>
<title>ArrayAvg Example</title>
</head>
<body>
<h3>ArrayAvg Example</h3>
<p> This example uses ArrayAvg to find the average of the numbers that you enter
          into an array.<br>
To enter more than two numbers press the <b>more</b> button.
</p>
<form action = "arrayavg.cfm">
<!--- The following code initially creates two fields. It adds fields if the user
          presses MORE. FormElem is initialized to two at the beginning of this
          code to show that the form has two fields. ----->
<input type = "submit" name = "submit" value = "more">
<table cellspacing = "2" cellpadding = "2" border = "0">
<cfloop index = "LoopItem" from = "1" to = "#FormElem#">
   <tr>
      <cfoutput>
        <th align = "left">Number #LoopItem#</th>
        <td><input type = "text" name = "number#LoopItem#"></td>
      </cfoutput>
```

```
   </tr>
</cfloop>
</table>
<input type = "submit" name = "submit" value = "get the average">
</form>

<!--- create an array --->
<cfif IsDefined("FORM.submit")>
   <cfset myNumberArray = ArrayNew(1)>
   <cfset Count = 1>
   <cfloop index = "ListItem" list = "#Form.Fieldnames#">
      <cfif Left(ListItem,3) is "Num">
         <cfset myNumberArray[Count] = Val(Evaluate("number#Count#"))>
         <cfset count = IncrementValue(Count)>
      </cfif>
   </cfloop>

   <cfif Form.Submit is "get the average">
      <!--- use ArrayAvg to get the average of the two numbers --->
      <p>The average of the numbers that you entered is
      <cfoutput>#ArrayAvg(myNumberArray)#.</cfoutput>
   <cfelse>
      <cfoutput>Try again. You must enter at least two numeric values.
      </cfoutput>
   </cfif>
</cfif>
</body>
</html>
```

ArrayClear

Description Deletes the data in an array.

Return value True, on successful completion.

Category Array functions

Syntax `ArrayClear(array)`

See also `ArrayDeleteAt`

History New in ColdFusion MX: this function can be used on XML objects.

Parameters

Parameter	Description
array	Name of an array

Example

```
<h3>ArrayClear Example</h3>
<!--- create a new array --->
<cfset MyArray = ArrayNew(1)>
<!--- populate an element or two --->
<cfset MyArray[1] = "Test">
<cfset MyArray[2] = "Other Test">
<!--- output the contents of the array --->
<p>Your array contents are:
<cfoutput>#ArrayToList(MyArray)#</cfoutput>
<!--- check if the array is empty --->
<p>Is the array empty?:
<cfoutput>#ArrayIsEmpty(MyArray)#</cfoutput>
<p>Now, clear the array:
<!--- now clear the array --->
<cfset Temp = ArrayClear(MyArray)>
<!--- check if the array is empty --->
<p>Is the array empty?:
<cfoutput>#ArrayIsEmpty(MyArray)#</cfoutput>
```

ArrayDeleteAt

Description
Deletes an element from an array.

When an element is deleted, ColdFusion recalculates index positions. For example, in an array that contains the months of the year, deleting the element at position 5 removes the entry for May. After this, to delete the entry for June, you would delete the element at position 5 (not 6).

Return value
True, on successful completion.

Category
Array functions

Syntax
`ArrayDeleteAt(array, position)`

See also
`ArrayInsertAt`

History
New in ColdFusion MX: this function can be used on XML objects.

New in ColdFusion MX: this function can throw the InvalidArrayIndexException error.

Parameters

Parameter	Description
array	Name of an array
position	Array position

Throws
If this function attempts to delete an element at position 0, or specifies a value for `position` that is greater than the size of `array`, this function throws an InvalidArrayIndexException error.

Example

```
<h3>ArrayDeleteAt Example</h3><p>
<!--- create an array --->
<cfset DaysArray = ArrayNew(1)>
<!--- populate an element or two --->
<cfset DaysArray[1] = "Monday">
<cfset DaysArray[2] = "Tuesday">
<cfset DaysArray[3] = "Wednesday">
<!--- delete the second element --->
<p>Is the second element gone?:
 <cfoutput>#ArrayDeleteAt(DaysArray,2)#</cfoutput>
<!--- the formerly third element, "Wednesday" is second element --->
<p>The second element is now: <cfoutput>#DaysArray[2]#</cfoutput>
```

ArrayInsertAt

Description: Inserts a value into an array. Array elements whose indexes are greater than the new position are incremented by one. The array length increases by one.

Return value: True, on successful completion.

Category: Array functions

Syntax: `ArrayInsertAt(array, position, value)`

See also: `ArrayDeleteAt`

History: New in ColdFusion MX: this function can be used on XML objects.

New in ColdFusion MX: this function can throw the InvalidArrayIndexException error.

Parameters

Parameter	Description
array	Name of an array
position	Index position at which to insert value
value	Value to insert

Throws: If this function attempts to insert an element at position 0, or specifies a value for `position` that is greater than the size of `array`, this function throws an InvalidArrayIndexException error.

Example

```
<h3>ArrayInsertAt Example</h3><p>
<!--- create a new array --->
<cfset DaysArray = ArrayNew(1)>
<!--- populate an element or two --->
<cfset DaysArray[1] = "Monday">
<cfset DaysArray[2] = "Tuesday">
<cfset DaysArray[3] = "Thursday">
<!--- add an element before position 3 --->
<p>Add an element before position 3:
  <cfoutput>#ArrayInsertAt(DaysArray,3,"Wednesday")#</cfoutput>
<p>Now output the array as a list:
<cfoutput>#ArrayToList(DaysArray)#</cfoutput>
<!--- The array now has four elements. Element 3, "Thursday", has become element
        four --->
```

ArrayIsEmpty

Description: Determines whether an array is empty of data elements.

Return value: True, if the array is empty; otherwise, False.

Category: Array functions

Syntax: `ArrayIsEmpty(array)`

See also: ArrayLen

History: New in ColdFusion MX: this function can be used on XML objects.

Parameters

Parameter	Description
array	Name of an array

Example

```
<h3>ArrayIsEmpty Example</h3>
<!--- create a new array --->
<cfset MyArray = ArrayNew(1)>
<!--- populate an element or two --->
<cfset MyArray[1] = "Test">
<cfset MyArray[2] = "Other Test">
<!--- output the contents of the array --->
<p>Your array contents are:
<cfoutput>#ArrayToList(MyArray)#</cfoutput>
<!--- check if the array is empty --->
<p>Is the array empty?:
<cfoutput>#ArrayIsEmpty(MyArray)#</cfoutput>
<p>Now, clear the array:
<!--- now clear the array --->
<cfset Temp = ArrayClear(MyArray)>
<!--- check if the array is empty --->
<p>Is the array empty?:
<cfoutput>#ArrayIsEmpty(MyArray)#</cfoutput>
```

ArrayLen

Description Determines the number of elements in an array.

Return value The number of elements in an array.

Category Array functions

Syntax `ArrayLen(array)`

See also `ArrayIsEmpty`

History New in ColdFusion MX: this function can be used on child XML objects.

Parameters

Parameter	Description
array	Name of an array

Example

```
<h3>ArrayLen Example</h3>
<cfquery name = "GetEmployeeNames" datasource = "cfsnippets">
SELECT FirstName, LastName FROM Employees
</cfquery>
<!--- create an array --->
<cfset myArray = ArrayNew(1)>
<!--- set element one to show where we are --->
<cfset myArray[1] = "Test Value">
<!--- loop through the query and append these names
successively to the last element --->
<cfloop query = "GetEmployeeNames">
   <cfset temp = ArrayAppend(myArray, "#FirstName# #LastName#")>
</cfloop>
<!--- show the resulting array as a list --->
<cfset myList = ArrayToList(myArray, ",")>
<!--- output the array as a list --->
<cfoutput>
   <p>The contents of the array are as follows:
   <p>#myList#
   <p>This array has #ArrayLen(MyArray)# elements.
</cfoutput>
```

ArrayMax

Description — Array maximum function.

Return value — The largest numeric value in an array. If the `array` parameter value is an empty array, returns zero.

Category — Array functions

Syntax — `ArrayMax(array)`

Parameters

Parameter	Description
array	Name of an array

Example

```
<h3>ArrayMax Example</h3>
<p>This example uses ArrayMax to find the largest number in an array.<br> </p>
<!--- After checking whether the form has been submitted, the code creates an array
         and assigns the form fields to the first two elements in the array. ---->
<cfif IsDefined("FORM.submit")>
   <cfset myNumberArray = ArrayNew(1)>
   <cfset myNumberArray[1] = number1>
   <cfset myNumberArray[2] = number2>
   <cfif Form.Submit is "Maximum Value">
      <!--- use ArrayMax to find the largest number in the array --->
      <p>The largest number that you entered is
      <cfoutput>#ArrayMax(myNumberArray)#.</cfoutput>
   </cfif>
</cfif>
<!---- The following form provides two numeric fields that are compared when the
         form is submitted. --->
<form action = "arraymax.cfm">
<input type = "hidden" name = "number1_Float">
<input type = "hidden" name = "number2_Float">
<input type = "text" name = "number1"><br>
<input type = "text" name = "number2"><br>
<input type = "submit" name = "submit" value = "Maximum Value">
</form>
```

ArrayMin

Description: Array minimum function.

Return value: The smallest numeric value in an array. If the `array` parameter value is an empty array, returns zero.

Category: Array functions

Syntax: `ArrayMin(array)`

Parameters

Parameter	Description
array	Name of an array

Example

```
<h3>ArrayMin Example</h3>
<p>This example uses ArrayMin to find the smallest number in an array.<br></p>
<!--- After checking whether the form has been submitted, this code creates an
         array and assigns the form fields to the first two elements. ----->
<cfif IsDefined("FORM.submit")>
   <cfset myNumberArray = ArrayNew(1)>
   <cfset myNumberArray[1] = FORM.number1>
   <cfset myNumberArray[2] = FORM.number2>

   <cfif Form.Submit is "Minimum Value">
      <!--- use ArrayMin to find the smallest number in the array --->
      <p>The smallest number that you entered is
      <cfoutput>#ArrayMin(myNumberArray)#.</cfoutput>
   </cfif>
</cfif>
<!---- The following form provides two numeric fields that are compared when the
         form is submitted. ----->
<form action = "arraymin.cfm">
<input type = "hidden" name = "number1_Float">
<input type = "hidden" name = "number2_Float">
<input type = "text" name = "number1"><br>
<input type = "text" name = "number2"><br>
<input type = "submit" name = "submit" value = "Minimum Value">
</form>
```

ArrayNew

Description: Creates an array of 1–3 dimensions. Index array elements with square brackets: [ ]. ColdFusion arrays expand dynamically as data is added.

Return value: An array

Category: Array functions

Syntax: `ArrayNew(dimension)`

Parameters

Parameter	Description
dimension	Number of dimensions in new array. 1, 2, or 3

Example

```
<h3>ArrayNew Example</h3>
<!--- Make an array --->
<cfset MyNewArray = ArrayNew(1)>
<!--- ArrayToList does not function properly if the Array is not initialized with
         ArraySet --->
<cfset temp = ArraySet(MyNewArray, 1,6, "")>

<!--- set some elements --->
<cfset MyNewArray[1] = "Sample Value">
<cfset MyNewArray[3] = "43">
<cfset MyNewArray[6] = "Another Value">

<!--- is it an array? --->
<cfoutput>
   <p>Is this an array? #IsArray(MyNewArray)#
   <p>It has #ArrayLen(MyNewArray)# elements.
   <p>Contents: #ArrayToList(MyNewArray)#
<!--- the array has expanded dynamically to six elements with the use of ArraySet,
         even though we only set three values --->
</cfoutput>
```

ArrayPrepend

Description Inserts an array element at the beginning of an array.

Return value True, on successful completion.

Category Array functions

Syntax `ArrayPrepend(array, value)`

See also ArrayAppend

Parameters

Parameter	Description
array	Name of an array
value	Value to insert at beginning of array

Example

```
<h3>ArrayPrepend Example</h3>
<cfquery name = "GetEmployeeNames" datasource = "cfsnippets">
   SELECT FirstName, LastName FROM Employees
</cfquery>
<!--- create an array --->
<cfset myArray = ArrayNew(1)>
<!--- set element one to show where we are --->
<cfset myArray[1] = "Test Value">
<!--- loop through query. Append names successively before last element (list
          reverses itself from the standard queried output, as it keeps
          prepending the array entry) --->
<cfloop query = "GetEmployeeNames">
   <cfoutput>#ArrayPrepend(myArray, "#FirstName# #LastName#")#
   </cfoutput>, Array was prepended<br>
</cfloop>
<!--- show the resulting array as a list --->
<cfset myList = ArrayToList(myArray, ",")>
<!--- output the array as a list --->
<cfoutput>
   <p>The contents of the array are as follows:
   <p>#myList#
</cfoutput>
```

ArrayResize

Description

Resets an array to a specified minimum number of elements. This can improve performance, if used to size an array to its expected maximum. For more than 500 elements, use `ArrayResize` immediately after using the `ArrayNew` tag.

ColdFusion arrays expand dynamically as data is added.

Return value

True, on successful completion.

Category

Array functions

Syntax

`ArrayResize(array, minimum_size)`

Parameters

Parameter	Description
array	Name of an array
minimum_size	Minimum array size

Example

```
<h3>ArrayResize Example</h3>
<!--- perform a query to get the list --->
<cfquery name = "GetCourses" datasource = "cfsnippets">
SELECT * FROM Courses
</cfquery>
<!--- make a new array --->
<cfset MyArray = ArrayNew(1)>
<!--- resize that array to the number of records
in the query --->
<cfset temp = ArrayResize(MyArray, GetCourses.RecordCount)>
<cfoutput>
The array is now #ArrayLen(MyArray)# elements, to match
the query of #GetCourses.RecordCount# records.
</cfoutput>
```

ArraySet

Description: In a one-dimensional array, sets the elements in a specified index range to a value. Useful for initializing an array after a call to `ArrayNew`.

Return value: True, on successful completion.

Category: Array functions

Syntax: `ArraySet(array, start_pos, end_pos, value)`

See also: `ArrayNew`

History: New in ColdFusion MX: this function can be used on XML objects.

Parameters

Parameter	Description
array	Name of an array.
start_pos	Starting index position of range to set.
end_pos	Ending index position of range to set. If this value is greater than array length, ColdFusion adds elements to array.
value	Value to which to set each element in the range.

Example

```
<h3>ArraySet Example</h3>

<!--- Make an array --->
<cfset MyNewArray = ArrayNew(1)>
<!--- ArrayToList does not function properly if the Array has not been initialized
         with ArraySet --->
<cfset temp = ArraySet(MyNewArray, 1,6, "Initial Value")>

<!--- set some elements --->
<cfset MyNewArray[1] = "Sample Value">
<cfset MyNewArray[3] = "43">
<cfset MyNewArray[6] = "Another Value">
...
```

ArraySort

Description: Sorts array elements numerically or alphanumerically.

Return value: True, if sort is successful; False, otherwise.

Category: Array functions, List functions

Syntax: `ArraySort(array, sort_type [, sort_order ])`

History: New in ColdFusion MX: this function can throw the ArraySortSimpleValueException error and ValueNotNumeric error.

New in ColdFusion MX: in a `textnocase`, descending sort, this function might return elements in a different sort order than in earlier releases. If `sort_type = "textnocase"` and `sort_order = "desc"`, ColdFusion MX processes elements that *differ only in case* differently from earlier releases, as follows:

- ColdFusion MX reverses the elements' original order
- Earlier releases of ColdFusion do not change the elements' original order

For example, in a `textnocase`, `desc` sort of `d,a,a,b,A`, the following occurs:

- ColdFusion MX returns `d,b,A,a,a`
- Earlier ColdFusion releases return `d,b,a,a,A`

(In a `textnocase`, *`asc`* sort, all ColdFusion releases return `a,a,A,b,d`.)

Parameters

Parameter	Description
array	Name of an array
sort_type	• numeric: sorts numbers • text: sorts text alphabetically, taking case into account (also known as case sensitive). All letters of one case precede the first letter of the other case: - aabzABZ, if `sort_order = "asc"` (ascending sort) - ZBAzbaa, if `sort_order = "desc"` (descending sort) • textnocase: sorts text alphabetically, without regard to case (also known as case-insensitive). A letter in varying cases precedes the next letter: - aAaBbBzzZ, in an ascending sort; preserves original intra-letter order - ZzzBbBaAa, in a descending sort; reverses original intra-letter order
sort_order	• asc - ascending sort order. Default. - aabzABZ or aAaBbBzzZ, depending on value of `sort_type`, for letters - from smaller to larger, for numbers • desc - descending sort order. - ZBAzbaa or ZzzBbBaAa, depending on value of `sort_type`, for letters - from larger to smaller, for numbers

Throws: If an array element is other than a simple element, this function throws an ArraySortSimpleValueException error. If `sort_type` is numeric and an array element is not numeric, this function throws a ValueNotNumeric error.

Example

```
<!--- This example shows ArraySort --->
<cfquery name = "GetEmployeeNames" datasource = "cfsnippets">
SELECT FirstName, LastName FROM Employees
</cfquery>
<!--- create an array --->
<cfset myArray = ArrayNew(1)>
<!--- loop through the query and append these names successively to the last
         element --->
<cfloop query = "GetEmployeeNames">
   <cfset temp = ArrayAppend(myArray, "#FirstName# #LastName#")>
</cfloop>
<!--- show the resulting array as a list --->
<cfset myList = ArrayToList(myArray, ",")>
<!--- sort that array descending alphabetically --->
<cfset isSuccessful = ArraySort(myArray, "textnocase", "desc")>
...
```

ArraySum

Description — Array sum function.

Return value — The sum of values in an array. If the `array` parameter value is an empty array, returns zero.

Category — Array functions, Mathematical functions

Syntax — `ArraySum(array)`

Parameters

Parameter	Description
array	Name of an array

Example

```
<h3>ArraySum Example</h3>
<p>This example uses ArraySum to add two numbers together.<br> </p>
<!--- After checking whether the form has been submitted, the code creates
        an array and assigns the form fields to the first two elements in
        the array. --->
<cfif IsDefined("FORM.submit")>
   <cfset myNumberArray = ArrayNew(1)>
   <cfset myNumberArray[1] = number1>
   <cfset myNumberArray[2] = number2>

   <cfif Form.Submit is "Add">
      <!--- use ArraySum to add the number in the array --->
      <p>The sum of the numbers is
      <cfoutput>#ArraySum(myNumberArray)#.</cfoutput>
   </cfif>
</cfif>
<!--- This form provides two numeric fields that are added when the form is
        submitted. --->
<form action = "arraysum.cfm" method="post">
   <input type = "hidden" name = "number1_Float">
   <input type = "hidden" name = "number2_Float">
   <input type = "text" name = "number1">
   <br>
   <input type = "text" name = "number2">
   <br>
   <input type = "submit" name = "submit" value = "Add">
</form>
```

ArraySwap

Description Swaps array values of an array at specified positions. This function is more efficient than multiple `cfset` tags.

Return value True, on successful completion.

Category Array functions

Syntax `ArraySwap(array, position1, position2)`

Parameters

Parameter	Description
array	Name of an array
position1	Position of first element to swap
position2	Position of second element to swap

Example

```
<h3>ArraySwap Example</h3>

<cfset month = ArrayNew(1)>
<cfset month[1] = "February">
<cfset month[2] = "January">
<cfset temp = ArraySwap(month, 1, 2)>
<cfset temp = ArrayToList(month)>

<p>Show the results: <cfoutput>#temp#</cfoutput>
```

ArrayToList

Description: Converts a one-dimensional array to a list.

Return value: Delimited list, as a string.

Category: Array functions, Conversion functions, List functions

Syntax: `ArrayToList(array [, delimiter ])`

Parameters

Parameter	Description
array	Name of array
delimiter	Character to separate list elements. Default: comma. ColdFusion uses only the first character of this parameter.

Example

```
<h3>ArrayToList Example</h3>
<cfquery name = "GetEmployeeNames" datasource = "cfsnippets">
SELECT FirstName, LastName FROM Employees
</cfquery>
<!--- create an array --->
<cfset myArray = ArrayNew(1)>
<!--- loop through query, append names successively to last element --->
<cfloop query = "GetEmployeeNames">
   <cfset temp = ArrayAppend(myArray, "#FirstName# #LastName#")>
</cfloop>
<!--- show the resulting array as a list --->
<cfset myList = ArrayToList(myArray, ",")>
<!--- sort that array descending alphabetically --->
<cfset myAlphaArray = ArraySort(myArray, "textnocase", "desc")>
<!--- show the resulting alphabetized array as a list --->
<cfset myAlphaList = ArrayToList(myArray, ",")>
<!--- output the array as a list --->
<cfoutput>
   <p>The contents of the array are as follows:
   <p>#myList#
   <p>This array, alphabetized by first name (descending):
   <p>#myAlphaList#
   <p>This array has #ArrayLen(MyArray)# elements.
</cfoutput>
```

Asc

Description Determines the value of a character.

Return value The value of the first character of a string; if string is empty, returns zero.

Category String functions

Syntax `Asc(string)`

See also `Chr`

History New in ColdFusion MX: ColdFusion supports the Java UCS-2 representation of Unicode characters, up to a value of 65536. (Earlier releases supported 1-255.)

Parameters

Parameter	Description
string	A string

Example

```
<h3>Asc Example</h3>
<!--- if the character string is not empty, output its ASCII value --->
<cfif IsDefined("FORM.charVals")>

   <cfif FORM.charVals is not "">
      <cfoutput>#Left(FORM.charVals,1)# =
     #Asc(FORM.charVals)#</cfoutput>
   <cfelse>
<!--- if it is empty, output an error message --->
      <h4>Enter a character</h4>
   </cfif>
</cfif>

<form action = "asc.cfm">
<p>Enter a character to see its ASCII value
<br><input type = "Text" name = "CharVals" size = "1" MAXLENGTH = "1">
<p><input type = "Submit" name = ""> <input type = "RESET">
</form>
```

ASin

Description Determines the arcsine of a number. The arcsine is the angle whose sine is *number*.

Return value The arcsine of a number, in radians.

Category Mathematical functions

Syntax `ASin(number)`

See also `Sin`, `Cos`, `Pi`, `Tan`

Parameters

Parameter	Description
number	Sine of an angle. The value must be between -1 and 1, inclusive.

Usage The range of the result is -π/2 to π/2 radians. To convert degrees to radians, multiply degrees by π/180. To convert radians to degrees, multiply radians by 180/π.

Example

```
<h3>ASin Example</h3>
<!--- output its arcsine value --->
<cfif IsDefined("FORM.SinNum")>
   <cfif IsNumeric(FORM.SinNum)>
      <cfif FORM.SinNum LESS THAN OR EQUAL TO 1>
         <cfif FORM.SinNum GREATER THAN OR EQUAL TO -1>
            ASin(<cfoutput>#FORM.SinNum#</cfoutput>) =
             <cfoutput>#Evaluate(ASin(FORM.sinNum))# Radians</cfoutput>
             <br> or <br>ASin(<cfoutput>#FORM.SinNum#</cfoutput>) =
            <cfoutput>
              #Evaluate(ASin(FORM.sinNum) * 180/Pi())# Degrees
            </cfoutput>
         <cfelse>
<!--- if it is less than negative one, output an error message --->
            <h4>Enter the sine of the angle to calculate, in degrees and radians.
            The value must be between 1 and -1, inclusive.</h4>
         </cfif>
      <cfelse>
<!--- if it is greater than one, output an error message --->
      <h4>Enter the sine of the angle to calculate, in degrees and radians. The
          value must be between 1 and -1, inclusive.</h4>
      </cfif>
   <cfelse>
<!--- if it is empty, output an error message --->
      <h4>Enter the sine of the angle to calculate, in degrees and radians. The
          value must be between 1 and -1,inclusive.</h4>
   </cfif>
</cfif>
<form action = "asin.cfm">
<p>Enter a number to get its arcsine in Radians and Degrees.
<br><input type = "Text" name = "sinNum" size = "25">
<p><input type = "Submit" name = ""> <input type = "RESET">
</form>
```

Atn

Description Arctangent function. The arctangent is the angle whose tangent is *number*.

Return value The arctangent of a number.

Category Mathematical functions

Syntax `Atn(number)`

See also `Tan`, `Sin`, `Cos`, `Pi`

Parameters

Parameter	Description
number	Tangent of an angle

Usage The range of the result is -π/2 to π/2 radians. To convert degrees to radians, multiply degrees by π/180. To convert radians to degrees, multiply radians by 180/π.

Example

```
<h3>Atn Example</h3>
<!--- output its Atn value --->
<cfif IsDefined("FORM.AtnNum")>
   <cfif IsNumeric(FORM.AtnNum)>
      Atn(<cfoutput>#FORM.AtnNum#</cfoutput>) =
      <cfoutput>#Atn(FORM.AtnNum)# radians =
       #Evaluate(Atn(FORM.AtnNum * 180/PI())#
        Degrees</cfoutput>
   <cfelse>
<!--- if it is empty, output an error message --->
      <h4>Enter a number</h4>
   </cfif>
</cfif>
<form action = "atn.cfm">
<p>Enter a number to get its arctangent in Radians and Degrees
<br><input type = "Text" name = "atnNum" size = "25">
<p><input type = "Submit" name = ""> <input type = "RESET">
</form>
```

AuthenticatedContext

Description This function is obsolete. Use the newer security tools; see "Authentication functions" on page 328 and the Application Security chapter in *Developing ColdFusion MX Applications with CFML*.

History New in ColdFusion MX: This function is obsolete. It does not work in ColdFusion MX and later ColdFusion releases.

AuthenticatedUser

Description This function is obsolete. Use the newer security tools; see "Authentication functions" on page 328 and the Application Security chapter in *Developing ColdFusion MX Applications with CFML*.

History New in ColdFusion MX: This function is obsolete. It does not work in ColdFusion MX and later ColdFusion releases.

BitAnd

Description: Performs a bitwise logical AND operation.

Return value: The bitwise AND of two long integers.

Category: Mathematical functions

Syntax: `BitAnd(number1, number2)`

See also: BitNot, BitOr, BitXor

Parameters

Parameter	Description
number1	32-bit signed integer
number2	32-bit signed integer

Usage: Bit functions operate on 32-bit signed integers, in the range -2147483648 – 2147483647.

Example:

```
<h3>BitAnd Example</h3>

<p>Returns the bitwise AND of two long integers.
<p>BitAnd(5,255): <cfoutput>#BitAnd(5,255)#</cfoutput>
<p>BitAnd(5,0): <cfoutput>#BitAnd(5,0)#</cfoutput>
<p>BitAnd(128,128): <cfoutput>#BitAnd(128,128)#</cfoutput>
```

BitMaskClear

Description — Performs a bitwise mask clear operation.

Return value — A number, bitwise cleared, with *length* bits beginning at *start*.

Category — Mathematical functions

Syntax — `BitMaskClear(number, start, length)`

See also — `BitMaskRead`, `BitMaskSet`

Parameters

Parameter	Description
number	32-bit signed integer
start	Integer, in the range 0-31, inclusive; start bit for mask
length	Integer, in the range 0-31, inclusive; length of mask

Usage — Bit functions operate on 32-bit signed integers, in the range -2147483648 – 2147483647.

Example

```
<h3>BitMaskClear Example</h3>

<p>Returns number bitwise cleared with length bits beginning from start.

<p>BitMaskClear(255, 4, 4): <cfoutput>#BitMaskClear(255, 4, 4)#
</cfoutput>
<p>BitMaskClear(255, 0, 4): <cfoutput>#BitMaskClear(255, 0, 4)#
</cfoutput>
<p>BitMaskClear(128, 0, 7): <cfoutput>#BitMaskClear(128, 0, 7)#
</cfoutput>
```

BitMaskRead

Description: Performs a bitwise mask read operation.

Return value: An integer, created from *length* bits of *number,* beginning at *start*.

Category: Mathematical functions

Syntax: `BitMaskRead(number, start, length)`

See also: `BitMaskClear`, `BitMaskSet`

Parameters

Parameter	Description
number	32-bit signed integer to mask
start	Integer, in the range 0-31, inclusive; start bit for read
length	Integer, in the range 0-31, inclusive; length of mask

Usage: Bit functions operate on 32-bit signed integers, in the range -2147483648 – 2147483647.

Example

```
<h3>BitMaskRead Example</h3>
<p>Returns integer created from <em>length</em> bits of <em>number</em>, beginning
         with <em>start</em>.

<p>BitMaskRead(255, 4, 4): <cfoutput>#BitMaskRead(255, 4, 4)#
</cfoutput>
<p>BitMaskRead(255, 0, 4): <cfoutput>#BitMaskRead(255, 0, 4)#
</cfoutput>
<p>BitMaskRead(128, 0, 7): <cfoutput>#BitMaskRead(128, 0, 7)#
</cfoutput>
```

BitMaskSet

Description — Performs a bitwise mask set operation.

Return value — A number, bitwise masked with *length* bits of *mask* beginning at *start*.

Category — Mathematical functions

Syntax — `BitMaskSet(number, mask, start, length)`

See also — `BitMaskClear`, `BitMaskRead`

Parameters

Parameter	Description
number	32-bit signed integer
mask	32-bit signed integer; mask
start	Integer, in the range 0-31, inclusive; start bit for mask
length	Integer, in the range 0-31, inclusive; length of mask

Usage — Bit functions operate on 32-bit signed integers, in the range -2147483648 – 2147483647.

Example

```
<h3>BitMaskSet Example</h3>
<p>Returns number bitwise masked with length bits of mask beginning at start.

<p>BitMaskSet(255, 255, 4, 4):
<cfoutput>#BitMaskSet(255, 255, 4, 4)#</cfoutput>
<p>BitMaskSet(255, 0, 4, 4):
<cfoutput>#BitMaskSet(255, 0, 4, 4)#</cfoutput>
<p>BitMaskSet(0, 15, 4, 4):
<cfoutput>#BitMaskSet(0, 15, 4, 4)#</cfoutput>
```

BitNot

Description: Performs a bitwise logical NOT operation.

Return value: A number; the bitwise NOT of a long integer.

Category: Mathematical functions

Syntax: `BitNot(number)`

See also: BitAnd, BitOr, BitXor

Parameters

Parameter	Description
number	32-bit signed integer

Usage: Bit functions operate on 32-bit signed integers, in the range -2147483648 – 2147483647.

Example:

```
<h3>BitNot Example</h3>
<p>Returns the bitwise NOT of a long integer.

<p>BitNot(0): <cfoutput>#BitNot(0)#</cfoutput>
<p>BitNot(255): <cfoutput>#BitNot(255)#</cfoutput>
```

BitOr

Description: Performs a bitwise logical OR operation.

Return value: A number; the bitwise OR of two long integers.

Category: Mathematical functions

Syntax: `BitOr(number1, number2)`

See also: BitAnd, BitNot, BitXor

Parameters:

Parameter	Description
number1	32-bit signed integer
number2	32-bit signed integer

Usage: Bit functions operate on 32-bit signed integers, in the range -2147483648 – 2147483647.

Example:

```
<h3>BitOr Example</h3>
<p>Returns the bitwise OR of two long integers.

<p>BitOr(5,255): <cfoutput>#BitOr(5,255)#</cfoutput>
<p>BitOr(5,0): <cfoutput>#BitOr(5,0)#</cfoutput>
<p>BitOr(7,8): <cfoutput>#BitOr(7,8)#</cfoutput>
```

BitSHLN

Description Performs a bitwise shift-left, no-rotation operation.

Return value A number, bitwise shifted without rotation to the left by *count* bits.

Category Mathematical functions

Syntax `BitSHLN(number, count)`

See also `BitSHRN`

Parameters

Parameter	Description
number	32-bit signed integer
count	Integer, in the range 0-31, inclusive; number of bits to shift the number

Usage Bit functions operate on 32-bit signed integers, in the range -2147483648 – 2147483647.

Example

```
<h3>BitSHLN Example</h3>
<p>Returns the number, bitwise shifted, without rotation, to the left by count
         bits.

<p>BitSHLN(1,1): <cfoutput>#BitSHLN(1,1)#</cfoutput>
<p>BitSHLN(1,30): <cfoutput>#BitSHLN(1,30)#</cfoutput>
<p>BitSHLN(1,31): <cfoutput>#BitSHLN(1,31)#</cfoutput>
<p>BitSHLN(2,31): <cfoutput>#BitSHLN(2,31)#</cfoutput>
```

BitSHRN

Description — Performs a bitwise shift-right, no-rotation operation.

Return value — A number, bitwise shifted, without rotation, to the right by *count* bits.

Category — Mathematical functions

Syntax — `BitSHRN(number, count)`

See also — BitSHLN

Parameters

Parameter	Description
number	32-bit signed integer
count	Integer, in the range 0-31, inclusive. Number of bits to shift the number

Usage — Bit functions operate on 32-bit signed integers, in the range -2147483648 – 2147483647.

Example

```
<h3>BitSHRN Example</h3>
<p>Returns a number, bitwise shifted, without rotation, to the right, by count
         bits.

<p>BitSHRN(1,1): <cfoutput>#BitSHRN(1,1)#</cfoutput>
<p>BitSHRN(255,7): <cfoutput>#BitSHRN(255,7)#</cfoutput>
<p>BitSHRN(-2147483548,1): <cfoutput>#BitSHRN(-2147483548,1)#
</cfoutput>
```

BitXor

Description: Performs a bitwise logical XOR operation.

Return value: Bitwise XOR of two long integers.

Category: Mathematical functions

Syntax: `BitXor(number1, number2)`

See also: BitAnd, BitNot, BitOr

Parameters

Parameter	Description
number1	32-bit signed integer
number2	32-bit signed integer

Usage: Bit functions operate on 32-bit signed integers, in the range -2147483648 – 2147483647.

Example:

```
<h3>BitXOr Example</h3>
<p>Returns the bitwise XOR of two long integers.

<p>BitXOr(5,255): <cfoutput>#BitXOr(5,255)#</cfoutput>
<p>BitXOr(5,0): <cfoutput>#BitXOr(5,0)#</cfoutput>
<p>BitXOr(128,128): <cfoutput>#BitXOr(128,128)#</cfoutput>
```

Ceiling

Description: Determines the closest integer that is greater than a specified number.

Return value: The closest integer that is greater than a given number.

Category: Mathematical functions

Syntax: `Ceiling(number)`

See also: `Int`, `Fix`, `Round`

Parameters

Parameter	Description
number	A real number

Example

```
<h3>Ceiling Example</h3>

<cfoutput>
<p>The ceiling of 3.4 is #ceiling(3.4)#
<p>The ceiling of 3 is #ceiling(3)#
<p>The ceiling of 3.8 is #ceiling(3.8)#
<p>The ceiling of -4.2 is #ceiling(-4.2)#
</cfoutput>
```

Chr

Description ASCII Character function.

Return value A character, of the given ASCII character code.

Category String functions

Syntax `Chr(number)`

See also `Asc`

History New in ColdFusion MX: ColdFusion supports the Java UCS-2 representation of Unicode characters, up to a value of 65536. (Earlier releases supported 1-255.)

Parameters

Parameter	Description
number	A value (a number in the range 0 to 255, inclusive)

Usage The values 0 – 31 are standard, nonprintable codes. For example:

- `Chr(10)` returns a linefeed character
- `Chr(13)` returns a carriage return character
- The two-character string `Chr(13) & Chr(10)` returns a newline

Example

```
<h3>CHR Example</h3>
<!--- if the character string is not empty, output its CHR value --->
<cfif IsDefined("FORM.Submit")>
   <cfoutput>#FORM.charVals# = #CHR(FORM.charVals)#</cfoutput>
</cfif>
<form action = "chr.cfm">
<p>Enter a number between 1 and 256 to see the ASCII representation.
<input type = "hidden" name = "CharVals_range" Value = "Min = 1 Max = 256">
<input type = "hidden" name = "CharVals_required" Value = "Enter an integer from 1
         to 256">
<br><input type = "Text" name = "CharVals">
<p><input type = "Submit" name = "Submit"> <input type = "RESET">
</form>
```

CJustify

Description: Centers a string in a field length.

Return value: String, center-justified. If *length* is less than the length of the string, the string is returned unchanged.

Category: Display and formatting functions, String functions

Syntax: `Cjustify(string, length)`

See also: LJustify, RJustify

Parameters

Parameter	Description
string	A string or a variable that contains one. May be empty. If it is a variable that is defined as a number, the function processes it as a string.
length	A positive integer or a variable that contains one. Length of field. Can be coded as: • A number; for example, 6 • A string representation of a number; for example, "6" Any other value causes ColdFusion to throw an error.

Example

```
<!--- This example shows how to use CJustify --->
<CFPARAM name = "jstring" DEFAULT = "">

<cfif IsDefined("FORM.justifyString")>
   <cfset jstring = Cjustify("#FORM.justifyString#", 35)>
</cfif>
<html>
<head>
<title>CJustify Example</title>
</head>
<body>
<h3>CJustify</h3>
<p>Enter a string; it will be center-justified within the sample field.
<form action = "cjustify.cfm">
<p><input type = "Text" value = "<cfoutput>#jString#</cfoutput>"
     size = 35 name = "justifyString">
<p><input type = "Submit" name = "">
<input type = "RESET">
</form>
</body>
</html>
```

Compare

Description — Performs a case-sensitive comparison of two strings.

Return value
- -1, if *string1* is less than *string2*
- 0, if *string1* is equal to *string2*
- 1, if *string1* is greater than *string2*

Category — String functions

Syntax — `Compare(string1, string2)`

See also — `CompareNoCase`, `Find`

Parameters

Parameter	Description
string1	A string or a variable that contains one
string2	A string or a variable that contains one

Usage — Compares the values of corresponding characters in *string1* and *string2*.

Example

```
<h3>Compare Example</h3>
<p>The compare function performs a <I>case-sensitive</I> comparison of two strings.

<cfif IsDefined("FORM.string1")>
 <cfset comparison = Compare(FORM.string1, FORM.string2)>
 <!--- switch on the variable to give various responses --->
 <cfswitch expression = #comparison#>
   <cfcase value = "-1">
      <h3>String 1 is less than String 2</h3>
      <I>The strings are not equal</I>
   </cfcase>
   <cfcase value = "0">
      <h3>String 1 is equal to String 2</h3>
      <I>The strings are equal!</I>
   </cfcase>
   <cfcase value = "1">
      <h3>String 1 is greater than String 2</h3>
      <I>The strings are not equal</I>
   </cfcase>
   <CFDEFAULTCASE>
      <h3>This is the default case</h3>
   </CFDEFAULTCASE>
   </cfswitch>
</cfif>
<form action = "compare.cfm">
<p>String 1
<br><input type = "Text" name = "string1">
<p>String 2
<br><input type = "Text" name = "string2">
<p><input type = "Submit" value = "Compare these Strings" name = "">
   <input type = "RESET">
</form>
```

CompareNoCase

Description: Performs a case-insensitive comparison of two strings.

Return value: An indicator of the difference:

- A negative number, if *string1* is less than *string2*
- 0, if *string1* is equal to *string2*
- A positive number, if *string1* is greater than *string2*

Category: String functions

Syntax: `CompareNoCase(string1, string2)`

See also: `Compare`, `FindNoCase`

Parameters

Parameter	Description
string1	A string or a variable that contains one
string2	A string or a variable that contains one

Example

```
<H3>CompareNoCase Example</H3>
<P>This function performs a <I>case-insensitive</I> comparison of two strings.
<CFIF IsDefined("form.string1")>
<CFSET comparison = Comparenocase(form.string1, form.string2)>
<!--- switch on the variable to give various responses --->
<CFSWITCH EXPRESSION=#comparison#>
   <CFCASE value="-1">
      <H3>String 1 is less than String 2</H3>
      <I>The strings are not equal</I>
   </CFCASE>
   <CFCASE value="0">
      <H3>String 1 is equal to String 2</H3>
      <I>The strings are equal!</I>
   </CFCASE>
   <CFCASE value="1">
      <H3>String 1 is greater than String 2</H3>
      <I>The strings are not equal</I>
   </CFCASE>
   <CFDEFAULTCASE>
      <H3>This is the default case</H3>
   </CFDEFAULTCASE>
</CFSWITCH>
</CFIF>
<FORM ACTION="comparenocase.cfm" METHOD="POST">
<P>String 1
<BR><INPUT TYPE="Text" NAME="string1">
<P>String 2
<BR><INPUT TYPE="Text" NAME="string2">
<P><INPUT TYPE="Submit" VALUE="Compare these Strings" NAME="">
         <INPUT TYPE="RESET">
</FORM>
```

Cos

Description: Cosine function.

Return value: The cosine of an angle, in radians.

Category: Mathematical functions

Syntax: `Cos(number)`

See also: Sin, Tan, Pi

Parameters

Parameter	Description
number	Angle, in radians, for which to calculate the cosine

Usage: The range of the result is -1 to 1.

To convert degrees to radians, multiply degrees by π/180. To convert radians to degrees, multiply radians by 180/π.

Example

```
<h3>Cos Example</h3>
<!--- output its Cos value --->
<cfif IsDefined("FORM.CosNum")>
   <cfif IsNumeric(#FORM.CosNum#)>
      Cos(<cfoutput>#FORM.CosNum#</cfoutput>) =
      <cfoutput>#Cos(FORM.cosNum)#
      radians = #Evaluate(Cos(FORM.cosNum) * 180/PI())#
      Degrees</cfoutput>
   <cfelse>
<!--- if it is empty, output an error message --->
      <h4>Enter a number between -1 and 1</h4>
   </cfif>
</cfif>
<form action = "cos.cfm" method="post">
<p>Enter a number to get its cosine in Radians and Degrees
<br><input type = "Text" name = "cosNum" size = "25">
<p><input type = "Submit" name = ""> <input type = "RESET">
</form>
```

CreateDate

Description: Creates a date/time object.

Return value: A date/time value.

Category: Date and time functions

Syntax: `CreateDate(year, month, day)`

See also: `CreateDateTime`, `CreateODBCDate`

Parameters

Parameter	Description
year	Integer, in the range 0-9999. See "How ColdFusion processes two-digit year values" on page 377.
month	Number in the range 1 (January) - 12 (December)
day	Number in the range 1 - 31

Usage: `CreateDate` is a subset of `CreateDateTime`.

The time in the returned object is set to 00:00:00.

How ColdFusion processes two-digit year values

For most locales, if the year part of a date string uses the abbreviated pattern "y" or "yy", it is interpreted relative to a century, by adjusting dates to within 80 years before and **20** years after the date instance is created.

For example, if the pattern is "mm/dd/yy", and a date instance is created on Jan 1, 1997, the string "01/11/12" is interpreted as Jan 11, 2012. The string "05/04/64" is interpreted as May 4, 1964.

For the following locales, the algorithm is similar, but the adjustment is made to within **72** years before and **28** years after the date instance is created:

- English (Australian)
- English (New Zealand)
- German (Austrian)
- German (Standard)
- German (Swiss)
- Portuguese (Brazilian)
- Portuguese (Standard)
- Swedish

This applies to the Sun JRE Version 1.4 and the IBM JRE Version 1.3.0.

Example

```
<h3>CreateDate Example</h3>
<CFIF IsDefined("form.year")>
<p>Your date value, generated with CreateDate:
<CFSET yourDate = CreateDate(form.year, form.month, form.day)>
```

```
<cfoutput>
<ul>
   <li>Formatted with CreateDate: #CreateDate(form.year, form.month, form.day)#
   <li>Formatted with CreateDateTime: #CreateDateTime(form.year, form.month,
          form.day, 12,13,0)#
   <li>Formatted with CreateODBCDate: #CreateODBCDate(yourDate)#
   <li>Formatted with CreateODBCDateTime: #CreateODBCDateTime(yourDate)#
</ul>

<p>The same value can be formatted with dateFormat:
<ul>
   <li>Formatted with CreateDate and dateFormat:
        #DateFormat(CreateDate(form.year, form.month, form.day), "mmm-dd-yyyy")#
   <li>Formatted with CreateDateTime and dateFormat:
        #DateFormat(CreateDateTime(form.year, form.month, form.day, 12,13,0))#
   <li>Formatted with CreateODBCDate and dateFormat:
        #DateFormat(CreateODBCDate(yourDate), "mmmm d, yyyy")#
   <li>Formatted with CreateODBCDateTime and dateFormat:
        #DateFormat(CreateODBCDateTime(yourDate), "d/m/yy")#
</ul>
</cfoutput>
</CFIF>
<CFFORM ACTION="createdate.cfm" METHOD="POST">
<p>Enter the year, month and day, as integers:
<PRE>
Year<CFINPUT TYPE="Text" NAME="year" VALUE="1998" VALIDATE="integer"
     REQUIRED="Yes">
Month<CFINPUT TYPE="Text" NAME="month" VALUE="6" VALIDATE="integer"
     REQUIRED="Yes">
Day<CFINPUT TYPE="Text" NAME="day" VALUE="8" VALIDATE="integer"
     REQUIRED="Yes">
</PRE>
<p><INPUT TYPE="Submit" NAME=""> <INPUT TYPE="RESET">
</cfform>
```

CreateDateTime

Description: Creates a date-time object.

Return value: A date/time value.

Category: Date and time functions

Syntax: `CreateDateTime(year, month, day, hour, minute, second)`

See also: `CreateDate`, `CreateTime`, `CreateODBCDateTime`, `Now`

Parameters

Parameter	Description
year	Integer, in the range 0-9999. See "How ColdFusion processes two-digit year values" on page 377.
month	Number in the range 1 (January)–12 (December)
day	Number in the range 1–31
hour	Number in the range 0–23
minute	Number in the range 0–59
second	Number in the range 0–59

Example

```
<h3>CreateDateTime Example</h3>

<CFIF IsDefined("form.year")>
Your date value, generated with CreateDateTime:
<CFSET yourDate = CreateDateTime(form.year, form.month, form.day,
        form.hour, form.minute, form.second)>

<cfoutput>
<ul>
   <li>Formatted with CreateDate: #CreateDate(form.year, form.month, form.day)#
   <li>Formatted with CreateDateTime: #CreateDateTime(form.year, form.month,
        form.day, form.hour, form.minute, form.second)#
   <li>Formatted with CreateODBCDate: #CreateODBCDate(yourDate)#
   <li>Formatted with CreateODBCDateTime: #CreateODBCDateTime(yourDate)#
</ul>

<p>The same value can be formatted with dateFormat:
<ul>
   <li>Formatted with CreateDate and dateFormat:
     #DateFormat(CreateDate(form.year, form.month, form.day), "mmm-dd-yyyy")#
   <li>Formatted with CreateDateTime and dateFormat:
     #DateFormat(CreateDateTime(form.year, form.month, form.day,
        form.hour, form.minute, form.second))#
   <li>Formatted with CreateODBCDate and dateFormat:
     #DateFormat(CreateODBCDate(yourDate), "mmmm d, yyyy")#
   <li>Formatted with CreateODBCDateTime and dateFormat:
     #DateFormat(CreateODBCDateTime(yourDate), "d/m/yy")#
</ul>
</cfoutput>
```

```
</CFIF>

<CFFORM ACTION="createdatetime.cfm" METHOD="POST">
<p>Please enter the year, month, and day, in integer format, for a date to view:
<PRE>
Year<CFINPUT TYPE="Text" NAME="year" VALUE="1998" VALIDATE="integer"
     REQUIRED="Yes">
Month<CFINPUT TYPE="Text" NAME="month" VALUE="6" RANGE="1,12"
     MESSAGE="Please enter a month (1-12)" VALIDATE="integer"
     REQUIRED="Yes">
Day<CFINPUT TYPE="Text" NAME="day" VALUE="8" RANGE="1,31"
     MESSAGE="Please enter a day of the month (1-31)" VALIDATE="integer"
     REQUIRED="Yes">
Hour<CFINPUT TYPE="Text" NAME="hour" VALUE="16" RANGE="0,23"
     MESSAGE="You must enter an hour (0-23)" VALIDATE="integer"
     REQUIRED="Yes">
Minute<CFINPUT TYPE="Text" NAME="minute" VALUE="12" RANGE="0,59"
     MESSAGE="You must enter a minute value (0-59)" VALIDATE="integer"
     REQUIRED="Yes">
Second<CFINPUT TYPE="Text" NAME="second" VALUE="0" RANGE="0,59"
     MESSAGE="You must enter a value for seconds (0-59)" VALIDATE="integer"
     REQUIRED="Yes">
</PRE>
<p><INPUT TYPE="Submit" NAME=""> <INPUT TYPE="RESET">
</cfform>
```

CreateObject

Description Creates a ColdFusion object, of a specified type.

Return value An object, of the specified type.

Note: You can enable and disable this function in the ColdFusion Administrator, ColdFusion Basic Security, Tag Restrictions page.

Category Extensibility functions

History New in ColdFusion MX: This function, and the `cfobject` tag, can instantiate ColdFusion components and web services. Executing operations on a CFC object executes CFML code that implements the CFC's method in the CFC file.

For more information, see *Developing ColdFusion MX Applications with CFML*.

CreateObject object types

For information about using this function, see these sections:

- CreateObject: COM object on page 382
- CreateObject: component object on page 383
- CreateObject: CORBA object on page 384
- CreateObject: Java or EJB object on page 386
- CreateObject: web service object on page 387

Note: On UNIX, this function does not support COM objects.

CreateObject: COM object

Description

The `CreateObject` function can create a Component Object Model (COM) object.

To create a COM object, you must provide this information:

- The object's program ID or filename
- The methods and properties available to the object through the IDispatch interface
- The arguments and return types of the object's methods

For most objects, you can get this information from the OLEView utility.

Note: On UNIX, this function does not support COM objects.

Return value

A COM object.

Syntax

`CreateObject(`*`type, class, context, serverName`*`)`

Parameters

Parameter	Description
type	Type of object to create. • com • corba • java • component • webservice
class	Component ProgID for the object to invoke
context	• InProc • Local • Remote
serverName	Server name, using UNC or DNS convention, in one of these forms: • \\lanserver • lanserver • http://www.servername.com • www.servername.com • 127.0.0.1 If `context = "remote"`, this parameter is required.

Usage

The following example creates the Windows Collaborative Data Objects (CDO) for NTS NewMail object to send mail. You would use this code within a `cfscript` tag.

```
Mailer = CreateObject("COM", "CDONTS.NewMail");
```

For more information, see *Developing ColdFusion MX Applications with CFML*.

CreateObject: component object

Description The `CreateObject` function can create a ColdFusion component (CFC) object.

Return value A component object.

Syntax `CreateObject(type, component-name)`

Parameters

Parameter	Description
type	Type of object to create. • com • corba • java • component • webservice
component-name	name of a component method

Usage In the following example, the CFScript statements assign the `tellTimeCFC` variable to the `tellTime` component using the `createObject` function. The `createObject` function references the component in another directory. To invoke component methods, you use function syntax. For more information, see *Developing ColdFusion MX Applications with CFML*.

Example

```
<b>Server's Local Time:</b>
<cfscript>
   tellTimeCFC=createObject("component","appResources.components.
     tellTime");
   tellTimeCFC.getLocalTime();
</cfscript>
<br>
<b>Calculated UTC Time:</b>
<cfscript>
   tellTimeCFC.getUTCTime();
</cfscript>
```

CreateObject: CORBA object

Description The `CreateObject` function can call a method on a CORBA object. The object must be defined and registered for use.

Return value A handle to a CORBA interface.

Syntax `CreateObject(type, context, class, locale)`

History New in ColdFusion MX: the Naming Service separator format for addresses has changed from a dot to a forward slash. For example, if "`context=NameService`", for a class, use either of the following formats for the `class` parameter:

- "`Macromedia/Eng/CF`"
- "`Macromedia.current/Eng.current/CF`"

(In earlier releases, the format was "`Macromedia.Eng.CF`".)

New in ColdFusion MX: the `locale` attribute specifies the Java config that contains the properties file.

Parameters

Parameter	Description
type	Type of object to create. • com • corba • java • component • webservice
context	• IOR: ColdFusion uses IOR to access CORBA server • NameService: ColdFusion uses naming service to access server. Valid only with the InitialContext of a VisiBroker ORB.
class	• If `context = "ior"`: absolute path of file that contains string version of the Interoperable Object Reference (IOR). ColdFusion must be able to read file; it should be local to ColdFusion server or accessible on network • If `context = "nameservice`:" forward slash-delimited naming context for naming service. For example: `Allaire//Doc/empobject`
locale	The name of the Java config that holds the properties file. For more information, see Administering ColdFusion MX.

Usage In the `class` attribute, if "`context=NameService`", use a dot separator for the first part of the string. Use either of the following formats:

- "`Macromedia/Eng/CF`"
- "`Macromedia.current/Eng.current/CF`"

ColdFusion Enterprise supports CORBA through the Dynamic Invocation Interface (DII). To use this function with CORBA objects, you must provide the name of the file that contains a string version of the IOR, or the object's naming context in the naming service. You must provide the object's attributes, method names and method signatures.

This function supports user-defined types (structures, arrays, and sequences).

Example

```
myobj = CreateObject("corba", "d:\temp\tester.ior", "ior",
          "visibroker") //    uses IOR

myobj = CreateObject("corba", "Macromedia/Eng/CF",
          "nameservice", "visibroker")    // uses nameservice

myobj = CreateObject("corba", "d:\temp\tester.ior",
          "nameservice")    // uses nameservice and default configuration
```

CreateObject: Java or EJB object

Description The `CreateObject` function can create a Java object, and, by extension, an EJB object.

Return value A Java object.

Syntax `CreateObject(type, class)`

Parameters

Parameter	Description
type	Type of object to create. • com • corba • java • component • webservice
class	A Java class name

Usage Any Java class available in the class path that is specified in the ColdFusion Administrator can be loaded and used from ColdFusion with the `CreateObject` function.

To access Java methods and fields:

1 Call the `CreateObject` function or the `cfobject` tag to load the class.

2 Use the `init` method, with appropriate arguments, to call an instance of the class. For example:

```
<cfset ret = myObj.init(arg1, arg2)>
```

Calling a public method on the object without first calling the "init" method invokes a static method. Arguments and return values can be any Java type (simple, array, object). If strings are passed as arguments, ColdFusion does the conversions; if strings are received as return values, ColdFusion does no conversion.

Overloaded methods are supported if the number of arguments is different. Future enhancements will let you use cast functions that allow method signatures to be built more accurately.

CreateObject: web service object

Description This function can create a web service object.

Return value A web service object.

Syntax `CreateObject(type, urltowsdl)`

Parameters

Parameter	Description
type	Type of object to create. • com • corba • java • component • webservice
urltowsdl	WSDL file URL; location of web service

Usage You can use the `createObject` function to create a web service.

Example

```
newobject2 = createObject("webservice","wsdlurl")
```

CreateODBCDate

Description: Creates an ODBC date object.

Return value: A date object, in normalized ODBC date format.

Category: Date and time functions

Syntax: `CreateODBCDate(date)`

See also: `CreateDate`, `CreateODBCDateTime`

Parameters

Parameter	Description
date	Date or date/time object in the range 100 AD–9999 AD. See "How ColdFusion processes two-digit year values" on page 377.

Usage: This function does not parse or validate values. To ensure that dates are entered and processed correctly (for example, to ensure that a day/month/year entry is not confused with a month/day/year entry, and so on), Macromedia recommends that you parse entered dates with the `DateFormat` function, using the mm-dd-yyyy mask, into three elements. Ensure that values are within appropriate ranges; for example, to validate a month value, use the attributes `validate = "integer"` and `range = "1,12"`.

```
<h3>CreateODBCDate Example</h3>
<CFIF IsDefined("form.year")>
<p>Your date value, generated with CreateDateTime:
<cfset yourDate = CreateDateTime(form.year, form.month, form.day, form.hour,
        form.minute, form.second)>
<cfoutput>
<ul>
  <li>Formatted with CreateDate: #CreateDate(form.year, form.month, form.day)#
  <li>Formatted with CreateDateTime:
    #CreateDateTime(form.year, form.month, form.day, form.hour, form.minute,
        form.second)#
  <li>Formatted with CreateODBCDate: #CreateODBCDate(yourDate)#
  <li>Formatted with CreateODBCDateTime: #CreateODBCDateTime(yourDate)#
</ul>
<p>The same value can be formatted with dateFormat:
<ul>
  <li>Formatted with CreateDate and dateFormat:
    #DateFormat(CreateDate(form.year,form.month, form.day), "mmm-dd-yyyy")#
  <li>Formatted with CreateDateTime and dateFormat:
    #DateFormat(CreateDateTime(form.year, form.month, form.day, form.hour,
        form.minute, form.second))#
  <li>Formatted with CreateODBCDate and dateFormat:
    #DateFormat(CreateODBCDate(yourDate), "mmmm d, yyyy")#
  <li>Formatted with CreateODBCDateTime and dateFormat:
    #DateFormat(CreateODBCDateTime(yourDate), "d/m/yy")#
</ul>
</cfoutput>
</cfif>
<cfform action="createodbcdate.cfm" method="POST">
```

```
<p>Enter the year, month and day, as integers:
<pre>
Year <cfinput type="text" name="year" value="1998" validate="integer"
     required="yes">
Month<cfinput type="text" name="month" value="6" range="1,12"
     message="please enter a month (1-12)" validate="integer"
     required="yes">
Day    <cfinput type="text" name="day" value="8" range="1,31"
     message="please enter a day of the month (1-31)" validate="integer"
     required="yes">
Hour <cfinput type="text" name="hour" value="16" range="0,23"
     message="you must enter an hour (0-23)" validate="integer"
     required="yes">
Minute<cfinput type="text" name="minute" value="12" range="0,59"
     message="you must enter a minute value (0-59)" validate="integer"
     required="yes">
Second<cfinput type="text" name="second" value="0" range="0,59"
     message="you must enter a value for seconds (0-59)" validate="integer"
     required="yes">
</pre>
<p><input type="submit" name=""> <input type="reset">
</cfform>
```

CreateODBCDateTime

Description: Creates an ODBC date-time object.

Return value: A date/time object, in ODBC timestamp format.

Category: Date and time functions

Syntax: `CreateODBCDateTime(date)`

See also: `CreateDateTime`, `CreateODBCDate`, `CreateODBCTime`, `Now`

Parameters

Parameter	Description
date	Date/time object in the range 100 AD–9999 AD. See "How ColdFusion processes two-digit year values" on page 377.

Usage: When passing a date/time value as a string, you must enclose it in quotation marks. Otherwise, it is interpreted as a number representation of a date/time object.

Example

```
<!--- This example shows how to use Createdate, Createdatetime, createODBCdate
         and createODBCDateTime --->
<h3>CreateODBCDateTime Example</h3>

<cfif IsDefined("form.year")>
Your date value, generated using CreateDateTime:
<cfset yourDate = CreateDateTime (form.year, form.month, form.day,
         form.hour,form.minute, form.second)>
<cfoutput>
<ul>
   <li>Formatted with CreateDate: #CreateDate(form.year, form.month,form.day)#
   <li>Formatted with CreateDateTime: #CreateDateTime(form.year,form.month,
      form.day,form.hour,form.minute,form.second)#
   <li>Formatted with CreateODBCDate: #CreateODBCDate(yourDate)#
   <li>Formatted with CreateODBCDateTime: #CreateODBCDateTime(yourDate)#
</ul>
<p>The same value can be formatted with dateFormat:
<ul>
   <li>Formatted with CreateDate and DateFormat:
      #DateFormat(CreateDate(form.year,form.month,form.day), "mmm-dd-yyyy")#
   <li>Formatted with CreateDateTime and DateFormat:
      #DateFormat(CreateDateTime(form.year,form.month,form.day,
      form.hour,form.minute,form.second))#
   <li>Formatted with CreateODBCDate and DateFormat:
      #DateFormat(CreateODBCDate(yourDate), "mmmm d, yyyy")#
   <li>Formatted with CreateODBCDateTime and DateFormat:
      #DateFormat(CreateODBCDateTime(yourDate), "d/m/yy")#
</ul>
</cfoutput>
</cfif>
<cfform action="createodbcdatetime.cfm" method="post">
<p>Enter a year, month and day, as integers:
<pre>
```

```
year <cfinput
        type="text"  name="year" value="1998" validate="integer" required="yes">

Month<cfinput
        type="text" name="month"  value="6"  range="1,12"
        message="enter a month (1-12)" validate="integer" required="yes">

Day  <cfinput type="text" name="day" value="8" range="1,31"
        message="enter a day of the month (1-31)" validate="integer"
          required="yes">

Hour <cfinput type="text" name="hour" value="16" range="0,23"
        message="you must enter an hour (0-23)" validate="integer" required="yes">

Minute<cfinput type="text" name="minute" value="12" range="0,59"
        message="you must enter a minute value (0-59)" validate="integer"
          required="yes">

Second<cfinput type="text" name="second" value="0" range="0,59"
        message="you must enter a seconds value (0-59)" validate="integer"
          required="yes">
</pre>
<p><input type="submit" name=""> <input type="reset">
</cfform>
```

CreateODBCTime

Description: Creates an ODBC time object.

Return value: A time object, in ODBC timestamp format.

Category: Date and time functions

Syntax: `CreateODBCTime(date)`

See also: `CreateODBCDateTime`, `CreateTime`

Parameters

Parameter	Description
date	Date/time object in the range 100 AD–9999 AD. See "How ColdFusion processes two-digit year values" on page 377.

Usage: When passing a date/time value as a string, you must enclose it in quotation marks. Otherwise, it is interpreted as a number representation of a date/time object.

Example:

```
<h3>CreateODBCTime Example</h3>
<cfif IsDefined("form.hour")>
Your time value, created with CreateTime...
<cfset yourTime = CreateTime(form.hour, form.minute, form.second)>
<cfoutput>
<ul>
   <li>Formatted with CreateODBCTime: #CreateODBCTime(yourTime)#
   <li>Formatted with TimeFormat: #TimeFormat(yourTime)#
</ul></cfoutput>
</cfif>
<cfform action="createodbctime.cfm" method="post">
<pre>
Hour <cfinput type="Text" name="hour" value="16" range="0,23" message="You must
         enter an hour (0-23)" validate="integer" required="Yes">
Minute<cfinput type="Text" name="minute" value="12" range="0,59" message="You must
         enter a minute value (0-59)" validate="integer" required="yes">
second<cfinput type="text" name="second" value="0" range="0,59" message="You must
         enter a value for seconds (0-59)" validate="integer" required="yes">
</pre>
<p><input type="Submit" name=""> <input type="reset">
</cfform>
```

CreateTime

Description: Creates a time variable.

Return value: A time variable.

Category: Date and time functions

Syntax: `CreateTime(hour, minute, second)`

See also: `CreateODBCTime`, `CreateDateTime`

Parameters

Parameter	Description
hour	Number in the range 0–23
minute	Number in the range 0–59
second	Number in the range 0–59

Usage: `CreateTime` is a subset of `CreateDateTime`.

A time variable is a special case of a date/time variable. The date part of a time variable is set to December 30, 1899.

Example

```
<h3>CreateTime Example</h3>
<cfif IsDefined("FORM.hour")>
Your time value, presented using CreateTime time function:
<cfset yourTime = CreateTime(FORM.hour, FORM.minute, FORM.second)>
<cfoutput><ul>
   <li>Formatted with timeFormat: #TimeFormat(yourTime)#
   <li>Formatted with timeFormat and hh:mm:ss: #TimeFormat(yourTime, 'hh:mm:ss')#
</ul></cfoutput>
</cfif>
<cfform action="createtime.cfm" method="post">
<pre>hour<cfinput type="text" name="hour" value="16" range="0,23"
   message="you must enter an hour (0-23)" validate="integer" required="yes">
Minute  <cfinput type="text" name="minute" value="12" range="0,59"
   message="you must enter a minute value (0-59)" validate="integer"
          required="yes">
Second  <cfinput type="text" name="second" value="0" range="0,59"
   message="you must enter a value for seconds (0-59)" validate="integer"
          required="yes">
</pre>
<p><input type="submit" name=""> <input type="reset">
</cfform>
```

CreateTimeSpan

Description: Creates a date/time object that defines a time period. You can add or subtract it from other date/time objects and use it with the `cachedWithin` attribute of `cfquery`.

Return value: A date/time object.

Category: Date and time functions

Syntax: `CreateTimeSpan(days, hours, minutes, seconds)`

See also: `CreateDateTime`, `DateAdd`, `DateConvert`

Parameters:

Parameter	Description
days	Integer in the range 0–32768; number of days in time period
hours	Number of hours in time period
minutes	Number of minutes in time period
seconds	Number of seconds in time period

Usage: Creates a special date/time object that should be used only to add and subtract from other date/time objects or with the `cfquery cachedWithin` attribute.

If you use the `cachedWithin` attribute of `cfquery`, and the original query date falls within the time span you define, cached query data is used. In this case, the `CreateTimeSpan` function is used to define a period of time from the present backwards. The `cachedWithin` attribute takes effect only if you enable query caching in the ColdFusion Administrator. For more information, see `cfquery`.

Example:

```
<!--- This example shows the use of CreateTimeSpan with cfquery --->
<h3>CreateTimeSpan Example</h3>
<!--- define startrow and maxrows to facilitate 'next N' style browsing --->
<cfparam name = "MaxRows" default = "10">
<cfparam name = "StartRow" default = "1">
<!--- Query database for information, if cached database information has not been
         updated in the last six hours. -------->
<cfoutput>
<cfquery name = "GetParks" datasource = "cfsnippets"
   cachedWithin = "#CreateTimeSpan(0, 6, 0, 0)#">
SELECT   PARKNAME, REGION, STATE
FROM     Parks
ORDER by ParkName, State
</cfquery>
</cfoutput>
<!--- build HTML table to display query --->
<table cellpadding = 1 cellspacing = 1>
<tr>
   <td colspan = 2 bgcolor = f0f0f0>
   <b><I>Park Name</i></b>
   </td>
```

```
   <td bgcolor = f0f0f0>
   <b><i>Region</i></b>
   </td>
   <td bgcolor = f0f0f0>
   <b><i>State</i></b>
   </td>
</tr>
<!--- Output query, define startrow and maxrows. Use query variable CurrentCount to
         track the row you are displaying. --->
<cfoutput query = "GetParks" StartRow = "#StartRow#"
         maxrows = "#maxrows#">
<tr>
   <td valign = top bgcolor = ffffed>
   <b>#GetParks.CurrentRow#</b>
   </td>
   <td valign = top>
   <font size = "-1">#ParkName#</font>
   </td>
   <td valign = top>
   <font size = "-1">#Region#</font>
   </td>
   <td valign = top>
   <font size = "-1">#State#</font>
   </td>
</tr>
</cfoutput>
<!--- If number of records is less than or equal to number of rows, offer link to
         same page, with startrow value incremented by maxrows (in this example,
         incremented by 10) --->
<tr>
   <td colspan = 4>
   <cfif (StartRow + MaxRows) LTE GetParks.RecordCount>
      <a href = "cfquery.cfm?startrow = <cfoutput>
      #Evaluate(StartRow + MaxRows)#</cfoutput>">
      See next <cfoutput>#MaxRows#</cfoutput> rows</A>
   </cfif>
   </td>
</tr>
</table>
```

CreateUUID

Description

Creates a Universally Unique Identifier (UUID). A UUID is a 35-character string representation of a unique 128-bit integer.

The ColdFusion UUID generation algorithm uses the unique time-of-day value, the IEEE 802 Host ID, and a cryptographically strong random number generator to generate UUIDs that conform to the principles laid out in the draft IEEE RFC "*UUIDs and GUIDs.*"

The ColdFusion UUID format is as follows:

`xxxxxxxx-xxxx-xxxx-xxxxxxxxxxxxxxxx` (8-4-4-16).

This does not conform to the Microsoft/DCE standard, which is as follows:

`xxxxxxxx-xxxx-xxxx-xxxxxx-xxxxxxxxxx` (8-4-4-6-10)

There are UUID test tools and a user-defined function called CreateGUID, which converts CFML UUIDs to UUID/Microsoft GUID format, available on the web at www.cflib.org.

Return value

A UUID, in the format `xxxxxxxx-xxxx-xxxx-xxxxxxxxxxxxxxxx`, where `x` is a hexadecimal digit (0-9 or A-F). (The character groups are 8-4-4-16.)

Category

Other functions

Syntax

```
CreateUUID()
```

Usage

Use this function to generate a persistent identifier in a distributed environment. To a very high degree of certainty, this function returns a unique value; no other invocation on the same or any other system returns the same value.

UUIDs are used by distributed computing frameworks, such as DCE/RPC, COM+, and CORBA. In ColdFusion, you can use UUIDs as primary table keys for applications in which data is stored in shared databases. In such cases, using numeric keys can cause primary-key constraint violations during table merges. Using UUIDs, you can eliminate these violations.

Example

```
<h3>CreateUUID Example</h3>
<p> This example uses CreateUUID to generate a UUID when you submit the form.
         You can submit the form more than once. </p>
<!--- Checks whether the form was submitted; if so, creates UUID. --->
<cfif IsDefined("Form.CreateUUID") Is True>
   <hr>
   <p>Your new UUID is: <cfoutput>#CreateUUID()#</cfoutput></p>
</cfif>
<form action = "createuuid.cfm">
<p><input type = "Submit" name = "CreateUUID"> </p>
</form>
```

DateAdd

Description: Adds units of time to a date.

Return value: A date/time object.

Category: Date and time functions

Syntax: `DateAdd("datepart", number, "date")`

See also: `DateConvert`, `DatePart`, `CreateTimeSpan`

Parameters

Parameter	Description
datepart	String: • yyyy: Year • q: Quarter • m: Month • y: Day of year • d: Day • w: Weekday • ww: Week • h: Hour • n: Minute • s: Second
number	Number of units of datepart to add to date (positive, to get dates in the future; negative, to get dates in the past)
date	Date/time object, in the range 100 AD–9999 AD. See "How ColdFusion processes two-digit year values" on page 377.

Usage: The *datepart* specifiers `y`, `d`, and `w` add a number of days to a date.

When passing a date/time object as a string, you must enclose it in quotation marks. Otherwise, it is interpreted as a numeric representation of a date/time object.

Example:

```
<!--- This example shows the use of DateAdd --->
...
<cfquery name = "GetMessages" datasource = "cfsnippets">
SELECT UserName, Subject, Posted
FROM Messages
</cfquery>
<p>This example uses DateAdd to determine when a message in the database
          expires. (The value selected is messages older than <cfoutput>#value#
<cfswitch expression = #type#>
   <cfcase value = "yyyy">years</cfcase>
   <cfcase value = "q">quarters</cfcase>
   <cfcase value = "m">months</cfcase>
   <cfcase value = "y">days of year</cfcase>
   <cfcase value = "w">weekdays</cfcase>
   <cfcase value = "ww">weeks</cfcase>
   <cfcase value = "h">hours</cfcase>
   <cfcase value = "n">minutes</cfcase>
```

```
   <cfcase value = "s">seconds</cfcase>
   <cfdefaultcase>years</cfdefaultcase>
</cfswitch>
</cfoutput>).

<table>
<tr>
   <td>username</td>
   <td>subject</td>
   <td>posted</td>
</tr>
<cfoutput query = "GetMessages">
<tr>
   <td>#UserName#</td>
   <d>#Subject#</d>
   <d>#Posted# <cfif DateAdd(type, value, posted)
          LT Now()>EXPIRED</cfif></d>
</tr>
</cfoutput>
</table>
```

DateCompare

Description: Performs a full date/time comparison of two dates.

Return value:
- -1, if *date1* is less than *date2*
- 0, if *date1* is equal to *date2*
- 1, if *date1* is greater than *date2*

Category: Date and time functions

Syntax:
```
DateCompare("date1", "date2" [, "datePart"])
```

See also: CreateDateTime, DatePart

Parameters:

Parameter	Description
date1	Date/time object, in the range 100 AD–9999 AD. See "How ColdFusion processes two-digit year values" on page 377.
date2	Date/time object, in the range 100 AD–9999 AD. See "How ColdFusion processes two-digit year values" on page 377.
datePart	Optional. String. Precision of the comparison. • s Precise to the second (default) • n Precise to the minute • h Precise to the hour • d Precise to the day • m Precise to the month • yyyy Precise to the year

Usage: When passing a date/time object as a string, you must enclose it in quotation marks. Otherwise, it is interpreted as a numeric representation of a date/time object.

Example:
```
<h3>DateCompare Example</h3>
<p>The datecompare function compares two date/time values.
<cfif IsDefined("FORM.date1")>
   <cfif IsDate(FORM.date1) and IsDate(FORM.date2)>
      <cfset comparison = DateCompare(FORM.date1, FORM.date2, FORM.precision)>

<!--- switch on the variable to give various responses --->
      <cfswitch expression = #comparison#>
         <cfcase value = "-1">
            <h3><cfoutput>#DateFormat(FORM.date1)#
            #TimeFormat(FORM.date1)#</cfoutput> (Date 1) is
            earlier than <cfoutput>#DateFormat(FORM.date2)#
            #TimeFormat(FORM.date2)#</cfoutput> (Date 2)</h3>
            <I>The dates are not equal</I>
         </cfcase>
         <cfcase value = "0">
            <h3><cfoutput>#DateFormat(FORM.date1)#
            #TimeFormat(FORM.date1)#</cfoutput> (Date 1) is equal
            to <cfoutput>#DateFormat(FORM.date2)#
            #TimeFormat(FORM.date2)#</cfoutput> (Date 2)</h3>
```

```
            <I>The dates are equal!</I>
          </cfcase>
          <cfcase value = "1">
            <h3><cfoutput>#DateFormat(FORM.date1)#
            #TimeFormat(FORM.date1)#</cfoutput> (Date 1) is later
            than <cfoutput>#DateFormat(FORM.date2)#
            #TimeFormat(FORM.date2)#</cfoutput> (Date 2)</h3>
            <I>The dates are not equal</I>
          </cfcase>
          <cfdefaultcase>
            <h3>This is the default case</h3>
          </cfdefaultcase>
        </cfswitch>
    <cfelse>
      <h3>Enter two valid date values</h3>
    </cfif>
</cfif>

<form action = "datecompare.cfm">
<hr size = "2" color = "#0000A0">
<p>Date 1
<br><input type = "Text" name = "date1"
     value = "<cfoutput>#DateFormat(Now())# #TimeFormat(Now())#
</cfoutput>">
<p>Date 2
<br><input type = "Text" name = "date2"
     value = "<cfoutput>#DateFormat(Now())# #TimeFormat(Now())#
</cfoutput>">
<p>Specify precision to the:
<br><select name = "precision">
        <option value = "s">
          Second
        </option>
        <option value = "n">
          Minute
        </option>
        <option value = "h">
          Hour
        </option>
        <option value = "d">
          Day
        </option>
        <option value = "m">
          Month
        </option>
        <option value = "yyyy">
          Year
        </option>
    </select>
<p><input type = "Submit" value = "Compare these dates" name = "">
<input type = "reset">
</form>
```

DateConvert

Description — Converts local time to Coordinated Universal Time (UTC), or UTC to local time. The function uses the daylight savings settings in the executing computer to compute daylight savings time, if required.

Return value — UTC- or local-formatted time object.

Category — Date and time functions

Syntax — `DateConvert("conversion-type", "date")`

See also — `GetTimeZoneInfo`, `CreateDateTime`, `DatePart`

Parameters

Parameter	Description
conversion-type	• local2Utc: Converts local time to UTC time. • utc2Local: Converts UTC time to local time.
date	Date and time string or a variable that contains one. To create, use "CreateDateTime".

Usage — When passing a date/time object as a string, you must enclose it in quotation marks. Otherwise, it is interpreted as a numeric representation of a date/time object.

Note: You can pass the CreateDate or Now function as the `date` parameter of this function; for example: `#DateConvert(CreateDate(2001, 3, 3))#`

Example

```
<h3>DateConvert Example</h3>
<!--- This shows conversion of current date - time to UTC and back. --->
<cfset curDate = Now()>
<p><cfoutput>The current date and time: #curDate#. </cfoutput></p>
<cfset utcDate = DateConvert("local2utc", curDate)>
<cfoutput>
   <p>The current date and time converted to UTC time: #utcDate#.</p>
</cfoutput>
<!--- This code checks whether form was submitted. If so, the code generates
         the CFML date with the CreateDateTime function. --->
<cfif IsDefined("FORM.submit")>
   <cfset yourDate = CreateDateTime(FORM.year, FORM.month, FORM.day,
         FORM.hour,FORM.minute, FORM.second)>
   <p><cfoutput>Your date value, presented as a ColdFusion date/time
         string:#yourdate#.</cfoutput></p>
  <cfset yourUTC = DateConvert("local2utc", yourDate)>
   <p><cfoutput>Your date and time value, converted to Coordinated Universal Time
         (UTC): #yourUTC#.</cfoutput></p>
   <p><cfoutput>Your UTC date and time, converted back to local date and time:
         #DateConvert("utc2local", yourUTC)#.
         </cfoutput></p>
<cfelse>
   Type the date and time, and press Enter to see the conversion.
</cfif>
<Hr size = "2" color = "#0000A0">
<form action = "dateconvert.cfm">
```

```
<p>Enter year, month and day in integer format for date value to view:
<table cellspacing = "2" cellpadding = "2" border = "0">
<tr>
  <td>Year</td>
  <td><input type = "Text" name = "year" value = "1998"
         validate = "integer" required = "Yes"></td></tr>
<tr>
  <td>Month</td>
  <td><input type = "Text" name = "month" value = "6"
         range = "1,12" message = "Enter a month (1-12)"
         validate = "integer" required = "Yes"></td></tr>
<tr>
  <td>Day</td>
  <td><input type = "Text" name = "day" value = "8"
         range = "1,31"
         message = "Enter a day of the month (1-31)"
         validate = "integer" required = "Yes"></td></tr>
<tr>
  <td>Hour</td>
  <td><input type = "Text" name = "hour" value = "16"
         range = "0,23"
         message = "You must enter an hour (0-23)"
         validate = "integer" required = "Yes"></td></tr>
<tr>
  <td>Minute</td>
  <td><input type = "Text" name = "minute" value = "12"
         range = "0,59"
         message = "You must enter a minute value (0-59)"
         validate = "integer" required = "Yes"></td></tr>
<tr>
  <td>Second</td>
  <td><input type = "Text" name = "second" value = "0"
         range = "0,59"
         message = "You must enter a value for seconds (0-59)"
         validate = "integer" required = "Yes"></td></tr>
<tr>
  <td><input type = "Submit" name = "submit" value = "Submit"></td>
  <td><input type = "RESET"></td></tr>
</table>
```

DateDiff

Description: Determines the number of units by which *date1* is less than *date2*.

Return value: A number of units, of type *datepart*.

Category: Date and time functions

Syntax: `DateDiff("datepart", "date1", "date2")`

See also: `DateAdd`, `DatePart`, `CreateTimeSpan`

History: New in ColdFusion MX: This function now calculates negative date differences correctly; its output may be different from that in earlier releases.

Parameters

Parameter	Description
datepart	String: • yyyy: Year • q: Quarter • m: Month • y: Day of year • d: Day • w: Weekday • ww: Week • h: Hour • n: Minute • s: Second
date1	Date/time object, in the range 100 AD–9999 AD. See "How ColdFusion processes two-digit year values" on page 377.
date2	Date/time object, in the range 100 AD–9999 AD. See "How ColdFusion processes two-digit year values" on page 377.

Usage: To find the number of days between *date1* and *date2*, use `DayofYear` or `Day`.

When *datepart* is Weekday, DateDiff returns the number of weeks between the dates. If *date1* falls on a Monday, DateDiff counts the number of Mondays to *date2*. It counts *date2* but not *date1*.

If interval is Week, however, DateDiff returns the number of calendar weeks between the dates. It counts the number of Sundays between *date1* and *date2*. DateDiff counts *date2* if it falls on a Sunday; it does not count *date1*, even if it falls on a Sunday.

If *date1* refers to a later date than *date2*, DateDiff returns a negative number.

When passing date/time object as a string, enclose it in quotation marks. Otherwise, it is interpreted as a numeric representation of a date/time object.

Example:

```
<!--- This example shows the use of DateDiff --->
<cfif IsDefined("FORM.date1") and IsDefined("FORM.date2")>
   <cfif IsDate(FORM.date1) and IsDate(FORM.date2)>
      <p>This example uses DateDiff to determine the difference
```

```
      in <cfswitch expression = #type#>
         <cfcase value = "yyyy">years</cfcase>
         <cfcase value = "q">quarters</cfcase>
         <cfcase value = "m">months</cfcase>
         <cfcase value = "y">days of year</cfcase>
         <cfcase value = "d">days</cfcase>
         <cfcase value = "w">weekdays</cfcase>
         <cfcase value = "ww">weeks</cfcase>
         <cfcase value = "h">hours</cfcase>
         <cfcase value = "n">minutes</cfcase>
         <cfcase value = "s">seconds</cfcase>
         <cfdefaultcase>years</cfdefaultcase></cfswitch>
          dateparts between date1 and date2.
      <cfif DateCompare(form.date1, form.date2) is not 0>
      <p>The difference is <cfoutput>#Abs(DateDiff
       (type, form.date2, form.date1))#</cfoutput>
      <cfswitch expression = #type#>
         <cfcase value = "yyyy">years</cfcase>
         <cfcase value = "q">quarters</cfcase>
         <cfcase value = "m">months</cfcase>
         <cfcase value = "y">days of year</cfcase>
         <cfcase value = "d">days</cfcase>
         <cfcase value = "w">weekdays</cfcase>
         <cfcase value = "ww">weeks</cfcase>
         <cfcase value = "h">hours</cfcase>
         <cfcase value = "n">minutes</cfcase>
         <cfcase value = "s">seconds</cfcase>
         <cfdefaultcase>years</cfdefaultcase></cfswitch>.
      <cfelse>
...
```

DateFormat

Description: Formats a date value. Supports dates in the U.S. date format. For international date support, use `LSDateFormat`.

Return value: A formatted date/time object. If no mask is specified, returns the value in *dd-mmm-yy* format.

Category: Date and time functions

Syntax: `DateFormat("date" [, "mask" ])`

See also: `Now`, `CreateDate`, `LSDateFormat`, `LSParseDateTime`, `LSTimeFormat`, `TimeFormat`, `ParseDateTime`

History: New in ColdFusion MX: This function supports the `short`, `medium`, `long`, and `full` `mask` attribute options.

Parameters

Parameter	Description
date	Date/time object, in the range 100 AD–9999 AD. See "How ColdFusion processes two-digit year values" on page 377.
mask	Characters that show how ColdFusion displays a date: • d: Day of the month as digits; no leading zero for single-digit days. • dd: Day of the month as digits; leading zero for single-digit days. • ddd: Day of the week as a three-letter abbreviation. • dddd: Day of the week as its full name. • m: Month as digits; no leading zero for single-digit months. • mm: Month as digits; leading zero for single-digit months. • mmm: Month as a three-letter abbreviation. • mmmm: Month as its full name. • y: Year as last two digits; no leading zero for years less than 10. • yy: Year as last two digits; leading zero for years less than 10. • yyyy: Year represented by four digits. • gg: Period/era string. Ignored. Reserved. • short • medium • long • full

Usage: When passing a date/time object as a string, you must enclose it in quotation marks. Otherwise, it is interpreted as a numeric representation of a date/time object.

Note: You can pass the `CreateDate` function or `Now` function as the `date` parameter of this function; for example: `#DateFormat(CreateDate(2001, 3, 3))#`

If the switch is set, the default date format returned by this function cannot be parsed in an expression. However, if you specify a mask to indicate order (such as `mm/dd/yyyy`), the date returned by this function can be parsed.

Date and time values in database query results can vary in sequence and formatting unless you use functions to format them. To ensure that application users correctly

understand displayed dates and times, Macromedia recommends that you use this function and the LSDateFormat, TimeFormat, and LSTimeFormat functions to format resultset values. For more information and examples, see TechNote 22183, "*ColdFusion Server (5 and 4.5.x) with Oracle: Formatting Date and Time Query Results*," on our website at http://www.coldfusion.com/Support/KnowledgeBase/SearchForm.cfm.

Example

```
<cfset todayDate = Now()>
<body>
<h3>DateFormat Example</h3>
<p>Today's date is <cfoutput>#todayDate#</cfoutput>.
<p>Using DateFormat, we can display that date in different ways:
<cfoutput>
<ul>
   <li>#DateFormat(todayDate)#
   <li>#DateFormat(todayDate, "mmm-dd-yyyy")#
   <li>#DateFormat(todayDate, "mmmm d, yyyy")#
   <li>#DateFormat(todayDate, "mm/dd/yyyy")#
   <li>#DateFormat(todayDate, "d-mmm-yyyy")#
   <li>#DateFormat(todayDate, "ddd, mmmm dd, yyyy")#
   <li>#DateFormat(todayDate, "d/m/yy")#
</ul>
</cfoutput>
```

DatePart

Description: Extracts a part from a date value.

Return value: Part of a date, as an integer.

Category: Date and time functions

Syntax: `DatePart("datepart", "date")`

See also: DateAdd, DateConvert

Parameters

Parameter	Description
datepart	String: • yyyy: Year • q: Quarter • m: Month • y: Day of year • d: Day • w: Weekday • ww: Week • h: Hour • n: Minute • s: Second
date	Date/time object, in the range 100 AD–9999 AD. See "How ColdFusion processes two-digit year values" on page 377.

Usage: When passing a date/time object as a string, you must enclose it in quotation marks. Otherwise, it is interpreted as a numeric representation of a date/time object.

Example:

```
<!--- This example shows information available from DatePart --->
<cfset todayDate = Now()>
<h3>DatePart Example</h3>
<p>Today's date is <cfoutput>#todayDate#</cfoutput>.
<p>Using datepart, we extract an integer representing the dateparts from that value
<cfoutput>
<ul>
   <li>year: #DatePart("yyyy", todayDate)#
   <li>quarter: #DatePart("q", todayDate)#
   <li>month: #DatePart("m", todayDate)#
   <li>day of year: #DatePart("y", todayDate)#
   <li>day: #DatePart("d", todayDate)#
   <li>weekday: #DatePart("w", todayDate)#
   <li>week: #DatePart("ww", todayDate)#
   <li>hour: #DatePart("h", todayDate)#
   <li>minute: #DatePart("n", todayDate)#
   <li>second: #DatePart("s", todayDate)#
</ul>
</cfoutput>
```

Day

Description: Determines the day of the month, in a date.

Return value: The ordinal for the day of the month, ranging from 1 to 31.

Category: Date and time functions

Syntax: `Day("date")`

See also: DayOfWeek, DayOfWeekAsString, DayOfYear, DaysInMonth, DaysInYear, FirstDayOfMonth

Parameters

Parameter	Description
date	Date/time object, in the range 100 AD–9999 AD. See "How ColdFusion processes two-digit year values" on page 377.

Usage: When passing a date/time object as a string, you must enclose it in quotation marks. Otherwise, it is interpreted as a numeric representation of a date/time object.

Note: You can pass the CreateDate function or Now function as the date parameter of this function; for example: #Day(CreateDate(2001, 3, 3))#

Example

```
<h3>Day Example</h3>
<cfif IsDefined("FORM.year")>
   <p>More information about your date:
   <cfset yourDate = CreateDate(FORM.year, FORM.month, FORM.day)>
   <cfoutput>
      <p>Date: #DateFormat(yourDate)#. <br>
          It is #DayofWeekAsString(DayOfWeek(yourDate))#,
          day #DayOfWeek(yourDate)# in the week.
      <br>This is day #Day(YourDate)#
          in the month of #MonthAsString(Month(yourDate))#,
          which has #DaysInMonth(yourDate)# days.
      <br>We are in week #Week(yourDate)# of #Year(YourDate)#
      (day #DayofYear(yourDate)# of #DaysinYear(yourDate)#).
      <br><cfif IsLeapYear(Year(yourDate))>This is a leap year
           <cfelse>This is not a leap year</cfif>
   </cfoutput>
</cfif>
```

DayOfWeek

Description: Determines the day of the week, in a date.

Return value: The ordinal for the day of the week, as an integer in the range 1 (Sunday) to 7 (Saturday).

Category: Date and time functions

Syntax: `DayOfWeek("date")`

See also: `Day`, `DayOfWeekAsString`, `DayOfYear`, `DaysInMonth`, `DaysInYear`, `FirstDayOfMonth`

Parameters

Parameter	Description
date	Date/time object, in the range 100 AD–9999 AD. See "How ColdFusion processes two-digit year values" on page 377.

Usage: When passing a date/time object as a string, you must enclose it in quotation marks. Otherwise, it is interpreted as a numeric representation of a date/time object.

Note: You can pass the CreateDate function or Now function as the `date` parameter of this function; for example, `#DayOfWeek(CreateDate(2001, 3, 3))#`

Example:

```
<h3>DayOfWeek Example</h3>
<cfif IsDefined("FORM.year")>
   More information about your date:
   <cfset yourDate = CreateDate(FORM.year, FORM.month, FORM.day)>
   <cfoutput>
   <p>Your date, #DateFormat(yourDate)#.
   <br>It is #DayofWeekAsString(DayOfWeek(yourDate))#, day
           #DayOfWeek(yourDate)# in the week.
   <br>This is day #Day(YourDate)# in the month of
           #MonthAsString(Month(yourDate))#, which has
    #DaysInMonth(yourDate)# days.
   <br>We are in week #Week(yourDate)# of #Year(YourDate)# (day
    #DayofYear(yourDate)# of #DaysinYear(yourDate)#).
   <br><cfif IsLeapYear(Year(yourDate))>This is a leap year
     <cfelse>This is not a leap year</cfif>
   </cfoutput>
</cfif>
```

DayOfWeekAsString

Description: Determines the day of the week, in a date, as a string function.

Return value: The day of the week, as a string that corresponds to *day_of_week*.

Category: Date and time functions, String functions

Syntax: `DayOfWeekAsString(day_of_week)`

See also: Day, DayOfWeek, DayOfYear, DaysInMonth, DaysInYear, FirstDayOfMonth

Parameters

Parameter	Description
day_of_week	Integer, in the range 1 (Sunday) - 7 (Saturday).

Example

```
<h3>DayOfWeekAsString Example</h3>
<cfif IsDefined("FORM.year")>
More information about your date:
<cfset yourDate = CreateDate(FORM.year, FORM.month, FORM.day)>

<cfoutput>
<p>Your date, #DateFormat(yourDate)#.
<br>It is #DayofWeekAsString(DayOfWeek(yourDate))#, day
   #DayOfWeek(yourDate)# in the week.
<br>This is day #Day(YourDate)# in the month of
         #MonthAsString(Month(yourDate))#, which has
   #DaysInMonth(yourDate)# days.
<br>We are in week #Week(yourDate)# of #Year(YourDate)# (day #DayofYear(yourDate)#
         of #DaysinYear(yourDate)#).
<br><cfif IsLeapYear(Year(yourDate))>This is a leap year
  <cfelse>This is not a leap year</cfif>
</cfoutput>
</cfif>
```

DayOfYear

Description: Determines the day of the year, in a date.

Return value: The ordinal value of day of the year, as an integer.

Category: Date and time functions

Syntax: `DayOfYear("date")`

See also: `Day`, `DayOfWeek`, `DayOfWeekAsString`, `DaysInMonth`, `DaysInYear`, `FirstDayOfMonth`

Parameters

Parameter	Description
date	Date/time object, in the range 100 AD–9999 AD. See "How ColdFusion processes two-digit year values" on page 377.

Usage: This function accounts for leap years.

When passing a date/time object as a string, you must enclose it in quotation marks. Otherwise, it is interpreted as a numeric representation of a date/time object.

Note: You can pass the `CreateDate` function or `Now` function as the `date` parameter of this function; for example, `#DayOfYear(CreateDate(2001, 3, 3))#`

Example:

```
<h3>Day7OfYear Example</h3>
<cfif IsDefined("FORM.year")>
   More information about your date:
   <cfset yourDate = CreateDate(FORM.year, FORM.month, FORM.day)>
   <cfoutput>
   <p>Your date, #DateFormat(yourDate)#.
   <br>It is #DayofWeekAsString(DayOfWeek(yourDate))#,
           day #DayOfWeek(yourDate)# in the week.
   <br>This is day #Day(yourDate)# in the month of
           #MonthAsString(Month(yourDate))#, which has
    #DaysInMonth(yourDate)# days.
   <br>We are in week #Week(yourDate)# of #Year(yourDate)#
          (day #DayofYear(yourDate)# of #DaysinYear(yourDate)#).
   <br><cfif IsLeapYear(Year(yourDate))>This is a leap year
     <cfelse>This is not a leap year</cfif>
   </cfoutput>
</cfif>
```

DaysInMonth

Description: Determines the number of days in a month.

Return value: The number of days in the month in *Date*.

Category: Date and time functions

Syntax: `DaysInMonth("date")`

See also: `Day`, `DayOfWeek`, `DayOfWeekAsString`, `DayOfYear`, `DaysInYear`, `FirstDayOfMonth`

Parameters

Parameter	Description
date	Date/time object, in the range 100 AD–9999 AD. See "How ColdFusion processes two-digit year values" on page 377.

Usage: When passing a date/time object as a string, you must enclose it in quotation marks. Otherwise, it is interpreted as a numeric representation of a date/time object.

Note: You can pass the `Now` function or the `CreateDate` function as the `date` parameter of this function; for example: `#DaysInMonth(CreateDate(2001, 3, 3))#`

Example

```
<h3>DaysInMonth Example</h3>
<cfif IsDefined("FORM.year")>
   More information about your date:
   <cfset yourDate = CreateDate(FORM.year, FORM.month, FORM.day)>
   <cfoutput>
   <p>Your date, #DateFormat(yourDate)#.
   <br>It is #DayofWeekAsString(DayOfWeek(yourDate))#, day
           #DayOfWeek(yourDate)# in the week.
   <br>This is day #Day(YourDate)# in the month of
           #MonthAsString(Month(yourDate))#, which has
    #DaysInMonth(yourDate)# days.
   <br>We are in week #Week(yourDate)# of #Year(YourDate)#
           (day #DayofYear(yourDate)# of #DaysinYear(yourDate)#).
   <br><cfif IsLeapYear(Year(yourDate))>This is a leap year
     <cfelse>This is not a leap year</cfif>
   </cfoutput>
</cfif>
```

DaysInYear

Description: Determines the number of days in a year.

Return value: The number of days in a year.

Category: Date and time functions

Syntax: `DaysInYear("date")`

See also: `Day`, `DayOfWeek`, `DayOfWeekAsString`, `DayOfYear`, `DaysInMonth`, `DaysInYear`, `FirstDayOfMonth`, `IsLeapYear`

Parameters

Parameter	Description
date	• Date/time object, in the range 100 AD–9999 AD. See "How ColdFusion processes two-digit year values" on page 377.

Usage: DaysInYear accounts for leap years.

When passing a date/time object as a string, you must enclose it in quotation marks. Otherwise, it is interpreted as a numeric representation of a date/time object.

Note: You can pass the `CreateDate` function or the `Now` function as the `date` parameter of this function; for example: `#DaysInYear(CreateDate(2001, 3, 3))#`

Example:

```
<h3>DaysInYear Example</h3>
<cfif IsDefined("FORM.year")>
   More information about your date:
   <cfset yourDate = CreateDate(FORM.year, FORM.month, FORM.day)>
   <cfoutput>
   <p>Your date, #DateFormat(yourDate)#.
   <br>It is #DayofWeekAsString(DayOfWeek(yourDate))#, day
           #DayOfWeek(yourDate)# in the week.
   <br>This is day #Day(YourDate)# in the month of
           #MonthAsString(Month(yourDate))#, which has
    #DaysInMonth(yourDate)# days.
   <br>We are in week #Week(yourDate)# of #Year(yourDate)# (day
    #DayofYear(yourDate)# of #DaysinYear(yourDate)#).
</cfif>
```

DE

Description

Postpones evaluation of a string as an expression, when it is passed as a parameter to the IIf or Evaluate functions. Escapes any double quotation marks in the parameter and wraps the result in double quotation marks.

This function is especially useful with the IIf function, to prevent the function from evaluating a string that is to be output.

This applies to expressions that are *not* surrounded by pound signs. (If pound signs surround any part of an expression, a ColdFusion function evaluates that part first, regardless of whether the DE function is present.)

Return value

Parameter, surrounded by double quotation marks, with any inner double quotation marks escaped.

Category

Dynamic evaluation functions

Syntax

DE(*string*)

See also

Evaluate, IIf

Parameters

Parameter	Description
string	String to evaluate, after delay

Usage

Consider this example:

```
<condition> <true expression> <false expression>
      iif( 1 eq 2, DE('#Var1#'), DE('#Var2#'))
```

ColdFusion evaluates whatever is surrounded by pounds in the expression before executing it. So, although this expression is never true (because Var1 does not exist), the expression fails with an 'Error Resolving Parameter' error, because ColdFusion evaluates #Var1#' and #Var2#' before executing either expression.

This example returns 'Var2':

```
iif( 1 eq 2, DE('Var1'), DE('Var2'))
```

The following example uses IIF to alternate table-row background colors, white and gray. It uses the DE function to prevent ColdFusion from evaluating the color strings.

```
<cfoutput>
<table border="1" cellpadding="3">
<cfloop index="i" from="1" to="10">
  <tr bgcolor="#IIF( i mod 2 eq 0, DE("white"), DE("gray") )#">
    <td>
      hello #i#
    </td>
  </tr>
</cfloop>
</table>
</cfoutput>
```

For more information and code examples, see *Developing ColdFusion MX Applications with CFML*.

Example

```
<!--- This example shows the use of DE and Evaluate --->
<h3>DE Example</h3>
<cfif IsDefined("FORM.myExpression")>
<h3>The Expression Result</h3>
<cftry>
<!--- Evaluate the expression --->
<cfset myExpression = Evaluate(FORM.myExpression)>
<!--- Use DE to output the value of the variable, unevaluated --->
<cfoutput>
<I>The value of the expression #Evaluate(DE(FORM.MyExpression))#
is #MyExpression#.</I>
</cfoutput>
<!--- specify the type of error for which we are searching --->
<cfcatch type = "Any">
<!--- the message to display --->
   <h3>Sorry, there's been an <B>Error</B>.
   Try a simple expression, such as "2+2".</h3>
<cfoutput>
<!--- and the diagnostic message from ColdFusion Server --->
  <p>#cfcatch.message#
</cfoutput>
</cfcatch>
</cftry>
</cfif>
```

DecimalFormat

Description: Converts a number to a decimal-formatted string.

Return value: A *number* as a string formatted with two decimal places and a thousands separator.

Category: Display and formatting functions

Syntax: `DecimalFormat(number)`

See also: `DollarFormat`, `NumberFormat`

Parameters

Parameter	Description
number	Number to format

Example

```
<h3>DecimalFormat Function</h3>
<p>Returns a number to two decimal places.
<p>
<cfloop FROM = 1 TO = 20 INDEX = "counter">
   <cfoutput>
   #counter# * Square Root of 2:
      #DecimalFormat(Evaluate(counter * sqr(2)))#
   </cfoutput>
   <br>
</cfloop>
```

DecrementValue

Description Decrements the integer part of a number.

Return value Integer part of *number,* decremented by one.

Category Mathematical functions

Syntax `DecrementValue(`*number*`)`

See also `IncrementValue`

Parameters

Parameter	Description
number	Number to decrement

Example

```
<h3>DecrementValue Example</h3>
<p>Returns the integer part of a number decremented by one.
<p>DecrementValue(0):
   <cfoutput>#DecrementValue(0)#</cfoutput>
<p>DecrementValue("1"):
   <cfoutput>#DecrementValue("1")#</cfoutput>
<p>DecrementValue(123.35):
   <cfoutput>#DecrementValue(123.35)#</cfoutput>
```

Decrypt

Description Decrypts a string that is encrypted with the `Encrypt` function.

Return value String, unencrypted.

Category Other functions, String functions

Syntax `Decrypt(encrypted_string, seed)`

See also `Duplicate`, `Encrypt`

Parameters

Parameter	Description
encrypted_string	String or a variable that contains one. String to decrypt
seed	String. The 32-bit key that was used to encrypt the string.

Example

```
<!--- This example shows the use of Encrypt and Decrypt --->
<h3>Decrypt Example</h3>
<p>This function encrypts/decrypts a string. Enter a string and a key.
<cfif IsDefined("FORM.myString")>
   <cfset string = FORM.myString>
   <cfset key = FORM.myKey>
   <cfset encrypted = encrypt(string, key)>
   <cfset decrypted = decrypt(encrypted, key)>
   <cfoutput>
      <h4><B>The string:</B></h4> #string# <br>
      <h4><B>The key:</B></h4> #key#<br>
      <h4><B>Encrypted:</B></h4> #encrypted#<br>
      <h4><B>Decrypted:</B></h4> #decrypted#<br>
   </cfoutput>
</cfif>
<form action = "encrypt.cfm">
<p>Input your key:
<p><input type = "Text" name = "myKey" value = "foobar">
<p>Enter string to encrypt:
<p><textArea name = "myString" cols = "40" rows = "5" WRAP = "VIRTUAL">
This string will be encrypted (try typing some more)</textArea>
<input type = "Submit" value = "Encrypt my String">
</form>
```

DeleteClientVariable

Description: Deletes a client variable. (To test for the existence of a variable, use `IsDefined`.)

Return value: True, if the variable is successfully deleted; false, otherwise.

Category: Other functions

Syntax: `DeleteClientVariable("name")`

See also: `GetClientVariablesList`

History: New in ColdFusion MX: if the variable is not present, this function returns False. (In earlier releases, it threw an error.)

Parameters

Parameter	Description
name	Name of a client variable to delete, surrounded by double quotation marks

Example

```
<!--- This view-only example shows DeleteClientVariable --->
<h3>DeleteClientVariable Example</h3>

<p>This view-only example deletes a client variable called "User_ID", if it
   exists in the list of client variables returned by GetClientVariablesList.
<p>This example requires the existence of an Application.cfm file and client
   management to be in effect.
<!---
<cfset client.somevar = "">
<cfset client.user_id = "">
<p>Client variable list:<cfoutput>#GetClientVariablesList()#</cfoutput>
<cfif ListFindNoCase(GetClientVariablesList(), "User_ID") is not 0>

     <cfset temp = DeleteClientVariable("User_ID")>
     <p>Was variable "User_ID" Deleted? <cfoutput>#temp#</cfoutput>
</cfif>
<p>Amended Client variable list:<cfoutput>#GetClientVariablesList()#
</cfoutput>
--->
```

DirectoryExists

Description: Determines whether a directory exists.

Return value: Yes, if the specified directory exists; No, otherwise.

Category: System functions

Syntax: `DirectoryExists(absolute_path)`

See also: `FileExists`

Parameters

Parameter	Description
absolute_path	An absolute path

Example

```
<h3>DirectoryExists Example</h3>
<h3>Enter a directory to check for existence.</h2>
<form action = "directoryexists.cfm" method="post">
   <input type = "text" name = "yourDirectory">
   <br>
   <input type = "submit" name = "submit">
</form>

<cfif IsDefined("FORM.yourDirectory")>
   <cfif FORM.yourDirectory is not "">
   <cfset yourDirectory = FORM.yourDirectory>
      <cfif DirectoryExists(yourDirectory)>
         <cfoutput>
         <p>Your directory exists. Directory name: #yourDirectory#
         </cfoutput>
      <cfelse>
         <p>Your directory does not exist.</p>
      </cfif>
   </cfif>
</cfif>
```

DollarFormat

Description: Formats a string in U.S. format. (For other currencies, use `LSCurrencyFormat` or `LSEuroCurrencyFormat`.)

Return value: A number as a string, formatted with two decimal places, thousands separator, and dollar sign. If *number* is negative, the return value is enclosed in parentheses. If *number* is an empty string, returns zero.

Category: Display and formatting functions

Syntax: `DollarFormat(number)`

See also: `DecimalFormat`, `NumberFormat`

Parameters

Parameter	Description
number	Number to format

Example

```
<!--- This example shows the use of DollarFormat --->
<html>
<head>
<title>DollarFormat Example</title>
</head>
<body>
<h3>DollarFormat Example</h3>
<cfloop from = 8 to = 50 index = counter>
   <cfset full = counter>
   <cfset quarter = Evaluate(counter + (1/4))>
   <cfset half = Evaluate(counter + (1/2))>
   <cfset threefourth = Evaluate(counter + (3/4))>
   <cfoutput>
   <pre>
   bill#DollarFormat(full)##DollarFormat(quarter)#
    #DollarFormat(half)# #DollarFormat(threefourth)#
   18% tip#DollarFormat(Evaluate(full * (18/100)))#
    #DollarFormat(Evaluate(quarter * (18/100)))#
    #DollarFormat(Evaluate(half * (18/100)))#
    #DollarFormat(Evaluate(threefourth * (18/100)))#
   </pre>
   </cfoutput>
</cfloop>
</body>
</html>
```

Duplicate

Description: Returns a clone, also known as a deep copy, of a variable. There is no reference to the original variable.

Return value: A clone of a variable.

Category: Structure functions, System functions

Syntax: `Duplicate(variable_name)`

See also: `StructCopy`, other Structure functions

History: New in ColdFusion MX: this function can be used on XML objects.

Parameters:

Parameter	Description
variable_name	Name of a variable to duplicate

Usage: Use this function to duplicate complex structures, such as nested structures and queries.

Note: With this function, you cannot duplicate a COM, CORBA, or JAVA object returned from the `cfobject` tag or the `CreateObject` function. If an array element or structure field is a COM, CORBA, or JAVA object, you cannot duplicate the array or structure.

Example:

```
<h3>Duplicate Example</h3>
<cfset s1 = StructNew()>
<cfset s1.nested  = StructNew()>
<cfset s1.nested.item = "original">
<cfset copy = StructCopy(s1)>
<cfset clone = Duplicate(s1)>
<!--- modify the original --->
<cfset s1.nested.item = "modified">
<cfoutput>
<p>The copy contains the modified value: #copy.nested.item#</p>
<p>The duplicate contains the original value: #clone.nested.item#</p>
</cfoutput>
```

Encrypt

Description: Encrypts a string. Uses a symmetric key-based algorithm, in which the same key is used to encrypt and decrypt a string. The security of the encrypted string depends on maintaining the secrecy of the key. Uses an XOR-based algorithm that uses a pseudo-random 32-bit key, based on a seed passed by the user as a function parameter.

Return value: String; can be much longer than the original string.

Category: Other functions, String functions

Syntax: `Encrypt(string, seed)`

See also: `Decrypt`

Parameters

Parameter	Description
string	String to encrypt
seed	String. Seed used to generate 32-bit encryption key.

Example

```
<h3>Encrypt Example</h3>
<p>This function allows for the encryption and decryption of a string.
         Try it by entering a string and a key to see the results.
<cfif IsDefined("FORM.myString")>
  <cfset string = FORM.myString>
  <cfset key = FORM.myKey>
  <cfset encrypted = encrypt(string, key)>
  <cfset decrypted = decrypt(encrypted, key)>
   <cfoutput>
      <h4><B>The string:</B></h4> #string# <br>
      <h4><B>The key:</B></h4> #key#<br>
      <h4><B>Encrypted:</B></h4> #encrypted#<br>
      <h4><B>Decrypted:</B></h4> #decrypted#<br>
   </cfoutput>
</cfif>
<form action = "encrypt.cfm">
<p>Input your key:
<p><input type = "Text" name = "myKey" value = "foobar">
<p>Input your string to be encrypted:
<p><textArea name = "myString" cols = "40" rows = "5" WRAP = "VIRTUAL">
This string will be encrypted (try typing some more)
</textArea>
<input type = "Submit" value = "Encrypt my String">
</form>
```

Evaluate

Description: Evaluates one or more string expressions, dynamically, from left to right. (The results of an evaluation on the left can have meaning in an expression to the right.) Returns the result of evaluating the rightmost expression.

Return value: An object; the result of the evaluation(s).

Category: Dynamic evaluation functions

Syntax: `Evaluate(string_expression1 [, string_expression2 [, … ] ] )`

See also: `DE`, `IIf`

Parameters

Parameter	Description
string_expression1, string_expression2...	Expressions to evaluate

Usage: String expressions can be complex. If a string expression contains a single or double quotation mark, it must be escaped.

This function is useful for forming one variable from multiple variables. For example, to reference a column of the query qNames with a variable, `var`, using an index value to traverse rows, you could use the following code:

```
<cfset var=Evaluate("qNames.#colname#[#index#]")>
```

For more information, see *Developing ColdFusion MX Applications with CFML*.

Example:

```
<!--- This shows the use of DE and Evaluate --->
<h3>Evaluate Example</h3>
<cfif IsDefined("FORM.myExpression")>
<h3>The Expression Result</h3>
<cftry>
<!--- Evaluate the expression --->
<cfset myExpression = Evaluate(FORM.myExpression)>
<!--- Use DE to output the value of the variable, unevaluated --->
<cfoutput>
<I>The value of the expression #Evaluate(DE(FORM.MyExpression))#
is #MyExpression#.</I>
</cfoutput>
...
```

Exp

Description: Calculates the exponent whose base is e that represents *number*. The constant e equals 2.71828182845904, the base of the natural logarithm. This function is the inverse of `Log`, the natural logarithm of *number*.

Return value: The constant e, raised to the power of *number*.

Category: Mathematical functions

Syntax: `Exp(number)`

See also: `Log`, `Log10`

Parameters:

Parameter	Description
number	Exponent to apply to the base e

Usage: To calculate powers of other bases, use the exponentiation operator (^).

Example:

```
<h3>Exp Example</h3>
<cfif IsDefined("FORM.Submit")>
   <cfoutput>
   <p>Your number, #FORM.number#
   <br>#FORM.number# raised to the E power: #exp(FORM.number)#
   <cfif FORM.number LTE 0>
      <br>You must enter a positive real number to see its natural logarithm
      <cfelse><br>
      The natural logarithm of #FORM.number#: #log(FORM.number)#
   </cfif>
   <cfif FORM.number LTE 0><br>
      You must enter a positive real number to see its logarithm to base 10
      <cfelse><br>
         The logarithm of #FORM.number# to base 10: #log10(FORM.number)#
</cfif>
</cfoutput>
</cfif>
<cfform action = "exp.cfm">
Enter a number to see its value raised to the E power,its natural logarithm,
          and the logarithm of number to base 10.
<cfinput type = "Text" name = "number" message = "You must enter a number"
          validate = "float" required = "No">
<input type = "Submit" name = "Submit">
</cfform>
```

ExpandPath

Description
Creates an absolute, platform-appropriate path that is equivalent to the value of `relative_path`, appended to the base path. This function (despite its name) can accept an absolute or relative path in the `relative_path` attribute

The base path is the currently executing page's directory path. It is stored in pageContext.getServletContext().

Return value
A string. If the relative path contains a trailing forward slash or backward slash, the return value contains the same trailing character.

Category
System functions

Syntax
`ExpandPath(relative_path)`

See also
`FileExists`, `GetCurrentTemplatePath`, `GetFileFromPath`

History
New in ColdFusion MX: this function (despite its name) can accept an absolute or relative path in the `relative_path` attribute. To resolve a path, this function uses virtual mappings that are defined in the ColdFusion Administrator. This function does not reliably use virtual mappings that are defined in IIS, Apache, or other Web servers.

Parameters

Parameter	Description
relative_path	Relative or absolute directory reference or file name, within the current directory, (.\ and ..\) to convert to an absolute path. Can include forward backward slashes.

Usage
If the parameter or the returned path is invalid, the function throws an error.

These examples show the valid constructions of relative_path:

- `ExpandPath( "*.*")`
- `ExpandPath( "/")`
- `ExpandPath( "\")`
- `ExpandPath( "/mycfpage.cfm")`
- `ExpandPath( "mycfpage.cfm")`
- `ExpandPath( "myDir/mycfpage.cfm")`
- `ExpandPath( "/myDir/mycfpage.cfm")`
- `ExpandPath( "../../mycfpage.cfm")`

Example

```
<h3>ExpandPath Example - View Only</h3>
<!---
<cfset thisPath=ExpandPath("*.*")>
<cfset thisDirectory=GetDirectoryFromPath(thisPath)>
<cfoutput>
The current directory is: #GetDirectoryFromPath(thisPath)#

<cfif IsDefined("form.yourFile")>
<cfif form.yourFile is not "">
<cfset yourFile = form.yourFile>
```

```
   <cfif FileExists(ExpandPath(yourfile))>
   <p>Your file exists in this directory.  You entered
   the correct file name, #GetFileFromPath("#thisPath#/#yourfile#")#
   <CFELSE>
   <p>Your file was not found in this directory:
   <br>Here is a list of the other files in this directory:
   <!--- use CFDIRECTORY to give the contents of the
   snippets directory, order by name and size --->
   <CFDIRECTORY DIRECTORY="#thisDirectory#"
   NAME="myDirectory"
   SORT="name ASC, size DESC">
   <!--- Output the contents of the CFDIRECTORY as a CFTABLE --->
   <CFTABLE QUERY="myDirectory">
   <CFCOL HEADER="NAME:"
        TEXT="#Name#">
   <CFCOL HEADER="SIZE:"
        TEXT="#Size#">
   </CFTABLE>
   </cfif>
</cfif>
<cfelse>
<h3>Please enter a file name</h3>
</CFIF>
</cfoutput>

<FORM action="expandpath.cfm" METHOD="post">
<h3>Enter the name of a file in this directory <I>
         <FONT SIZE="-1">(try expandpath.cfm)</FONT></I></h3>
<INPUT TYPE="Text" NAME="yourFile">
<INPUT TYPE="Submit" NAME="">
</form>
 --->
```

FileExists

Description: Determines whether a file exists.

Return value: Yes, if the file specified in the parameter exists; No, otherwise.

Category: System functions, Decision functions

Syntax: `FileExists(absolute_path)`

See also: DirectoryExists, ExpandPath, GetTemplatePath

Parameters

Parameter	Description
absolute_path	An absolute path

Example

```
<h3>FileExists Example</h3>

<cfset thisPath = ExpandPath("*.*")>
<cfset thisDirectory = GetDirectoryFromPath(thisPath)>
<cfoutput>
The current directory is: #GetDirectoryFromPath(thisPath)#
<cfif IsDefined("FORM.yourFile")>
<cfif FORM.yourFile is not "">
<cfset yourFile = FORM.yourFile>
   <cfif FileExists(ExpandPath(yourfile))>
   <p>Your file exists in this directory. You entered
   the correct file name, #GetFileFromPath("#thisPath#/#yourfile#")#
   <cfelse>
```

Find

Description Finds the first occurrence of a *substring* in a *string*, from a specified start position. The search is case-sensitive.

Return value A number; the position of *substring* in *string*, *or* 0, if *substring* is not in *string*.

Category String functions

Syntax `Find(substring, string [, start ])`

See also `FindNoCase`, `Compare`, `FindOneOf`, `REFind`, `Replace`

Parameters

Parameter	Description
substring	A string or a variable that contains one. String for which to search.
string	A string or a variable that contains one. String in which to search.
start	Start position of search.

Example

```
<cfoutput>
   <cfset stringToSearch = "The quick brown fox jumped over the lazy dog.">
   #find("the",stringToSearch)#<br>
   #find("the",stringToSearch,35)#<br>
   #find("no such substring",stringToSearch)#<br>
   <br>
   #findnocase("the",stringToSearch)#<br>
   #findnocase("the",stringToSearch,5)#<br>
   #findnocase("no such substring",stringToSearch)#<br>
   <br>
   #findoneof("aeiou",stringToSearch)#<br>
   #findoneof("aeiou",stringToSearch,4)#<br>
   #findoneof("@%^*()",stringToSearch)#<br>
</cfoutput>
```

FindNoCase

Description: Finds the first occurrence of a *substring* in a *string* from a specified start position. If *substring* is not in *string*, returns zero. The search is case-insensitive.

Return value: The position of *substring* in *string*; *or* 0, if *substring* is not in *string*.

Category: String functions

Syntax: `FindNoCase(substring, string [, start ])`

See also: `Find`, `CompareNoCase`, `FindOneOf`, `REFind`, `Replace`

Parameters:

Parameter	Description
substring	A string or a variable that contains one. String for which to search.
string	A string or a variable that contains one. String in which to search.
start	Start position of search.

Example:

```
<cfset stringToSearch = "The quick brown fox jumped over the lazy dog.">
#find("the",stringToSearch)#<br>
#find("the",stringToSearch,35)#<br>
#find("no such substring",stringToSearch)#<br>
<br>
#findnocase("the",stringToSearch)#<br>
#findnocase("the",stringToSearch,5)#<br>
#findnocase("no such substring",stringToSearch)#<br>
<br>
#findoneof("aeiou",stringToSearch)#<br>
#findoneof("aeiou",stringToSearch,4)#<br>
#findoneof("@%^*()",stringToSearch)#<br>
```

FindOneOf

Description — Finds the first occurrence of *any one of a set of characters* in a *string*, from a specified start position. The search is case-sensitive.

Return value — The position of the first member of *set* found in *string*; *or* 0, if no member of *set* is found in *string*.

Category — String functions

Syntax — `FindOneOf(set, string [, start ])`

See also — `Find`, `Compare`, `REFind`

Parameters

Parameter	Description
set	A string or a variable that contains one. String that contains one or more characters to search for.
string	A string or a variable that contains one. String in which to search.
start	Start position of search.

Example

```
<cfset stringToSearch = "The quick brown fox jumped over the lazy dog.">
#find("the",stringToSearch)#<br>
#find("the",stringToSearch,35)#<br>
#find("no such substring",stringToSearch)#<br>
<br>
#findnocase("the",stringToSearch)#<br>
#findnocase("the",stringToSearch,5)#<br>
#findnocase("no such substring",stringToSearch)#<br>
<br>
#findoneof("aeiou",stringToSearch)#<br>
#findoneof("aeiou",stringToSearch,4)#<br>
#findoneof("@%^*()",stringToSearch)#<br>
```

FirstDayOfMonth

Description: Determines the ordinal (day number, in the year) of the first day of the month in which a given date falls.

Return value: A number corresponding to a day-number in a year.

Category: Date and time functions

Syntax: `FirstDayOfMonth(date)`

See also: `Day`, `DayOfWeek`, `DayOfWeekAsString`, `DayOfYear`, `DaysInMonth`, `DaysInYear`

Parameters

Parameter	Description
date	• Date/time object, in the range 100 AD–9999 AD. See "How ColdFusion processes two-digit year values" on page 377.

Usage: When passing a date/time value as a string, you must enclose it in quotation marks. Otherwise, it is interpreted as a number representation of a date/time object.

Example:

```
<h3>FirstDayOfMonth Example</h3>

<cfoutput>
The first day of #MonthAsString(Month(Now()))#,
 #Year(Now())# was
  a #DayOfWeekAsString(DayOfWeek(FirstDayOfMonth(Now())))#,
   day #FirstDayOfMonth(Now())# of the year.
</cfoutput>
```

Fix

Description: Converts a real number to an integer.

Return value: If *number* is greater than or equal to 0, the closest integer less than *number*.
If *number* is less than 0, the closest integer greater than *number*.

Category: Mathematical functions

Syntax: `Fix(number)`

See also: `Ceiling`, `Int`, `Round`

Parameters

Parameter	Description
number	A number

Example

```
<h3>Fix Example</h3>
<p>Fix returns the closest integer less than the number if the number is greater
         than or equal to 0. Fix returns the closest integer greater than the
         number if number is less than 0.
<cfoutput>
<p>The fix of 3.4 is #fix(3.4)#
<p>The fix of 3 is #fix(3)#
<p>The fix of 3.8 is #fix(3.8)#
<p>The fix of -4.2 is #fix(-4.2)#
</cfoutput>
```

FormatBaseN

Description: Converts *number* to a string, in the base specified by *radix*.

Return value: String that represents *number*, in the base *radix*.

Category: Display and formatting functions, Mathematical functions, String functions

Syntax: `FormatBaseN(number, radix)`

See also: `InputBaseN`

Parameters

Parameter	Description
number	Number to convert
radix	Base of the result

Example

```
<h3>FormatBaseN Example</h3>
<p>Converts a number to a string in the base specified by Radix.
<p>
<cfoutput>
<br>FormatBaseN(10,2): #FormatBaseN(10,2)#
<br>FormatBaseN(1024,16): #FormatBaseN(1024,16)#
<br>FormatBaseN(125,10): #FormatBaseN(125,10)#
<br>FormatBaseN(10.75,2): #FormatBaseN(10.75,2)#
</cfoutput>
<h3>InputBaseN Example</h3>
<p>InputBaseN returns the number obtained by converting a string,
         using base specified by Radix (an integer from 2 to 36).

<cfoutput>
<br>InputBaseN("1010",2): #InputBaseN("1010",2)#
<br>InputBaseN("3ff",16): #InputBaseN("3ff",16)#
<br>InputBaseN("125",10): #InputBaseN("125",10)#
<br>InputBaseN(1010,2): #InputBaseN(1010,2)#
</cfoutput>
```

GetAuthUser

Description Gets the name of an authenticated user.

Return value The name of an authenticated user.

Category Authentication functions

Syntax `GetAuthUser()`

See also `isUserInRole`

History New in ColdFusion MX: this function is new.

Example

```
<H3>GetAuthUser Example</H3>

<P>Authenticated User: <cfoutput>GetAuthUser()</cfoutput>
```

GetBaseTagData

Description: Used within a custom tag. Finds calling (ancestor) tag by name and accesses its data.

Return value: An object that contains data (variables, scopes, and so on) from an ancestor tag. If there is no ancestor by the specified name, or if the ancestor does not expose data (for example, `cfif`), an exception is thrown.

Category: Other functions

Syntax: `GetBaseTagData(tagname [, instancenumber ] )`

See also: `GetBaseTagList`

Parameters

Parameter	Req/Opt	Default	Description
tagname	Required		Ancestor tag name for which to return data
instancenumber	Optional	1 (closest ancestor)	Number of ancestor levels to jump before returning data

Example

```
<!--- This example shows the use of GetBaseTagData
          function. Typically used in custom tags.--->
...
<cfif trim(inCustomTag) neq "">
     <cfoutput>
       Running in the context of a custom
       tag named #inCustomTag#.<p>
     </cfoutput>
     <!--- Get the tag instance data --->
     <cfset tagData = GetBaseTagData(inCustomTag)>
     <!--- Find the tag's execution mode --->
     Located inside the
     <cfif tagData.thisTag.executionMode neq 'inactive'>
       template
     <cfelse>
       BODY
     </cfif>
```

GetBaseTagList

Description: Gets ancestor tag names, starting with the parent tag.

Return value: A comma-delimited list of uppercase ancestor tag names, as a string. The first list element is the current tag. If the current tag is nested, the next element is the parent tag. If the function is called for a top-level tag, it returns an empty string. If an ancestor does not expose data (see `GetBaseTagData`), its name might not be returned.

Category: Other functions

Syntax: `GetBaseTagList()`

See also: `GetBaseTagData`

Usage: This function does not display the following tags or end tags in the ancestor tag list:

- `cfif, cfelseif, cfelse`
- `cfswitch`
- `cfcase, cfdefaultcase`
- `cfoutput`
- `cftry, cfcatch`

This function displays the following tags only under the following conditions:

- `cfloop`: if there is a `cfquery` tag within it
- `cfoutput`: if at least one of its children is a complex expression
- `cfprocessingdirective`: if it has at least one other attribute besides `pageencoding`

Example:

```
<!--- This example shows the use of GetBaseTagList function.
         Typically used in custom tags. --->
...
<cfif thisTag.executionMode is "start">

   <!--- Get the tag context stack
   The list will look something like "CFIF,MYTAGNAME..." --->
   <cfset ancestorList = GetBaseTagList()>

   <!--- Output current tag name --->
   <cfoutput>This is custom tag #ListGetAt(ancestorList,2)#</cfoutput>
   <p>
   <!--- Determine whether this is nested inside a loop --->
   <cfset inLoop = ListFindNoCase(ancestorList, "cfloop")>
   <cfif inLoop neq 0>
      Running in the context of a cfloop tag.
   </cfif>
```

GetBaseTemplatePath

Description: Gets the absolute path of an application's base page.

Return value: The absolute path of the application base page, as a string.

Category: Other functions, System functions

Syntax:
```
GetBaseTemplatePath()
```

See also: GetCurrentTemplatePath, FileExists, ExpandPath

Example:
```
<h3>GetBaseTemplatePath Example</h3>

<p>The template path of the current page is:
<cfoutput>#GetBaseTemplatePath()#</cfoutput>
```

GetClientVariablesList

Description: Finds the client variables to which a page has write access.

Return value: Comma-delimited list of non-read-only client variables, as a string.

Category: List functions, Other functions

Syntax:
```
GetClientVariablesList()
```

See also: DeleteClientVariable

Usage: The list of variables returned by this function is compatible with ColdFusion list functions.

Example:
```
<!--- this example is view only --->

<h3>GetClientVariablesList Example</h3>

<p>This view-only example deletes a client variable called "User_ID", if it exists
         in the list of client variables returned by GetClientVariablesList().
<p>This example requires the existence of an Application.cfm file and that client
         management be in effect.
<!---
<cfset client.somevar = "">
<cfset client.user_id = "">
<p>Client variable list:<cfoutput>#GetClientVariablesList()#</cfoutput>

<cfif ListFindNoCase(GetClientVariablesList(), "User_ID") is not 0>
<!--- delete that variable
   <cfset temp = DeleteClientVariable("User_ID")>
   <p>Was variable "User_ID" Deleted? <cfoutput>#temp#</cfoutput>
</cfif>

<p>Amended Client variable list:<cfoutput>#GetClientVariablesList()#
 </cfoutput>
--->
```

GetCurrentTemplatePath

Description: Gets the path of the page that calls this function.

Return value: The absolute path of the page that contains the call to this function, as a string.

Category: System functions

Syntax: `GetCurrentTemplatePath()`

See also: `GetBaseTemplatePath`, `FileExists`, `ExpandPath`

Usage: If the function call is made from a page included with a `cfinclude` tag, this function returns the page path of an included page. Contrast this with the `GetBaseTemplatePath` function, which returns the path of the top-level page, even if it is called from an included page.

Example:

```
<!--- This example uses GetCurrentTemplatePath to show the
         template path of the current page --->
<h3>GetCurrentTemplatePath Example</h3>

<p>The template path of the current page is:
<cfoutput>#GetCurrentTemplatePath()#</cfoutput>
```

GetDirectoryFromPath

Description Extracts a filename and directory from an absolute path.

Return value Absolute path, without the filename. The last character is a backward slash.

Category System functions

Syntax `GetDirectoryFromPath(path)`

See also `ExpandPath`, `GetFileFromPath`

Parameters

Parameter	Description
path	Absolute path (drive, directory, filename, and extension)

Example

```
<h3>GetDirectoryFromPath Example</h3>
<cfset thisPath = ExpandPath("*.*")>
<cfset thisDirectory = GetDirectoryFromPath(thisPath)>
<cfoutput>
The current directory is: #GetDirectoryFromPath(thisPath)#
<cfif IsDefined("FORM.yourFile")>
   <cfif FORM.yourFile is not "">
      <cfset yourFile = FORM.yourFile>
         <cfif FileExists(ExpandPath(yourfile))>
         <p>Your file exists in this directory. You entered the correct file name,
           #GetFileFromPath("#thisPath#/#yourfile#")#
   <cfelse>
      <p>Your file was not found in this directory:
      <br>Here is a list of the other files in this directory:
      <!--- use cfdirectory show directory, order by name & size --->
      <cfdirectory directory = "#thisDirectory#"
         name = "myDirectory" SORT = "name ASC, size DESC">
      <!--- Output the contents of the cfdirectory as a CFTABLE --->
      <cftable query = "myDirectory">
         <cfcol header = "NAME:" text = "#Name#">
      <cfcol header = "SIZE:" text = "#Size#">
      </cftable>
      </cfif>
   </cfif>
<cfelse>
   <H3>Please enter a file name</H3>
   </cfif>
   </cfoutput>
   <form action="getdirectoryfrompath.cfm" METHOD="post">
      <H3>Enter the name of a file in this directory <I><FONT SIZE="-1">
          (try expandpath.cfm)</FONT></I></H3>
      <input type="Text" NAME="yourFile">
      <input type="Submit" NAME="">
   </form> --->
```

GetException

Description: Used with the `cftry` and `cfcatch` tags. Retrieves a Java exception object from a Java object.

Return value: Any Java exception object raised by a previous method call on the Java object.

Category: System functions

Syntax: `getException(object)`

Parameters

Parameter	Description
object	A Java object.

Usage: ColdFusion stores a Java exception object for each method call on a Java object. Subsequent method calls reset the exception object. To get the current exception object, you must call `getException` on the Java object before other methods are invoked on it.

Example:

```
<!--- Create the Java object reference --->
<cfobject action = create type = java class = primativetype name = myObj>
<!--- Calls the object's constructor --->
<cfset void  =  myObj.init()>
<cftry>
<cfset void  =  myObj.DoException() >
<Cfcatch type = "Any">
   <cfset exception = getException(myObj)>
<!--- user can call any valid method on the exception object--->
   <cfset message = exception.toString()>
   <cfoutput>
      Error<br>
      I got exception <br>
      <br> The exception message is: #message# <br>
   </cfoutput>
</cfcatch>
</cftry>
```

GetFileFromPath

Description: Extracts a filename from an absolute path.

Return value: File name, as a string.

Category: System functions

Syntax: `GetFileFromPath(path)`

See also: `ExpandPath`, `GetCurrentTemplatePath`

Parameters

Parameter	Description
path	Absolute path (drive, directory, filename, and extension)

Example

```
<h3>GetFileFromPath Example</h3>
<cfset thisPath  =  ExpandPath("*.*")>
<cfset thisDirectory  =  GetDirectoryFromPath(thisPath)>
<cfoutput>
The current directory is: #GetDirectoryFromPath(thisPath)#
<cfif IsDefined("FORM.yourFile")>
<cfif FORM.yourFile is not "">
<cfset yourFile = FORM.yourFile>
<cfif FileExists(ExpandPath(yourfile))>
     <p>Your file exists in this directory. You entered the correct file
         name, #GetFileFromPath("#thisPath#/#yourfile#")#
   <cfelse>
     <p>Your file was not found in this directory:
     <br>Here is a list of the other files in this directory:
     <!--- use cfdirectory to give the contents of the snippets
         directory, order by name and size --->
     <cfdirectory
       directory = "#thisDirectory#"
       name = "myDirectory"
       sort = "name ASC, size DESC">
     <!--- Output the contents of the cfdirectory as a cftable --->
     <cftable query = "myDirectory">
     <cfcol header = "NAME:" text = "#Name#">
   <cfcol header = "SIZE:" text = "#Size#">
...
```

GetFunctionList

Description: Displays a list of the functions that are available in ColdFusion.

Return value: A structure of functions.

Category: System functions

Syntax:

```
GetFunctionList()
```

Example:

```
<!----- This example shows the use of GetFunctionList. ---->
<cfset fList = GetFunctionList()>
<cfoutput>#StructCount(fList)# functions<br><br>
   </cfoutput>
<cfloop collection = "#fList#" item = "key">
   <cfoutput>#key#<br>
   </cfoutput>
</cfloop>
```

GetHttpRequestData

Description Makes HTTP request headers and body available to CFML pages. Useful for capturing SOAP request data, which can be delivered in an HTTP header.

Return value A ColdFusion structure.

Category System functions

Syntax `GetHttpRequestData()`

Usage The structure returned by this function contains the following entries:

Parameter	Description
headers	Structure that contains HTTP request headers as value pairs. Includes custom headers, such as SOAP requests.
content	Raw content from form submitted by client, in string or binary format. For content to be considered string data, the FORM request header "CONTENT_TYPE" must start with "text/" or be special case "application/x-www-form-urlencoded". Other types are stored as a binary object.
method	String that contains the CGI variable Request_Method.
protocol	String that contains the Server_Protocol CGI variable.

Note: To determine whether data is binary, use `IsBinary(x.content)`. To convert data to a string value, if it can be displayed as a string, use `toString(x.content)`.

The following example shows how this function can return HTTP header information.

Example

```
<cfset x = GetHttpRequestData()>
<cfoutput>
<table cellpadding = "2" cellspacing = "2">
  <tr>
    <td><b>HTTP Request item</b></td>
    <td><b>Value</b></td> </tr>
<cfloop collection = #x.headers# item = "http_item">
   <tr>
     <td>#http_item#</td>
     <td>#StructFind(x.headers, http_item)#</td></tr>
</cfloop>
<tr>
   <td>request_method</td>
   <td>#x.method#</td></tr>
<tr>
   <td>server_protocol</td>
   <td>#x.protocol#</td></tr>
</table>
<b>http_content --- #x.content#</b>
</cfoutput>
```

GetHttpTimeString

Description: Gets the current time, in the Universal Time code (UTC).

Return value: The time, as a string, according to the HTTP standard described in RFC 1123.

Category: Date and time functions, International functions

Syntax: `GetHttpTimeString(date_time_object)`

See also: GetLocale, GetTimeZoneInfo, SetLocale

Parameters

Parameter	Description
date_time_object	A ColdFusion date-time object string or Java Date object

Usage: The time in the returned string is UTC, consistent with the HTTP standard.

Example:

```
<cfoutput>
#GetHttpTimeString("#Now()#")#<br>
</cfoutput>
```

GetK2ServerDocCount

Description: Determines the number of documents that can be searched by the ColdFusion registered K2 Server.

This function is used primarily by the ColdFusion Verity and K2Server Administrator pages. This function uses Verity K2Server Release K2.2.0.

Return value: The number of collection metadata items stored in Verity collections.

Category: Full-text search functions, Query functions

Syntax:
```
GetK2ServerDocCount()
```

See also: GetK2ServerDocCountLimit

History: New in ColdFusion MX: This function is new.

Example:
```
<cfoutput>GetK2ServerDocCount =
        $*#GetK2ServerDocCount()#*$</cfoutput>
```

GetK2ServerDocCountLimit

Description: Gets the maximum number of documents that the ColdFusion registered K2 Server is permitted to return from a search.

This function is used primarily by the ColdFusion Verity and K2Server Administrator pages. This function uses Verity K2Server Release K2.2.0.

Return value: Number of collection metadata items that the K2 server permits, as an integer

Category: Full-text search functions, Query functions

Syntax:

```
GetK2ServerDocCountLimit()
```

See also: GetK2ServerDocCount

History: New in ColdFusion MX: This function is new.

Usage: If a search generates a larger number of documents than the limit, ColdFusion puts a warning message in the Administrator and in the log file.

The restricted version of K2 Server version that is installed with ColdFusion has the following document search limits:

- For ColdFusion MX Evaluation: 250,000
- For ColdFusion MX Developer: 10,000
- For ColdFusion MX Professional: 125,000
- For ColdFusion MX Enterprise: 250,000
- For ColdFusion MX with Macromedia Spectra: 750,000

K2Broker with ColdFusion MX has no limit.

Example:

```
<cfoutput>GetK2ServerDocCountLimit =
        $*#GetK2ServerDocCountLimit()#*$</cfoutput>
```

GetLocale

Description: Gets the current geographic/language locale value.

To set the default display format of date, time, number, and currency values in a ColdFusion application session, you use the `SetLocale` function.

Return value: The current locale value, as a string.

Category: Display and formatting functions, International functions, System functions

Syntax:
```
GetLocale()
```

See also: `SetLocale`

History: New in ColdFusion MX: this function determines whether a locale value is set. (The value is set with the `setlocale` function.)

- If the locale value is present, the function returns it.
- If the locale has not been explicitly set, ColdFusion determines whether the default locale of the ColdFusion Server computer operating system is among the locale values it supports. (The default locale is stored in the user environment variables user.language and user.region.)
  - If the default locale value is supported, the function returns this value
  - If the default locale value is not supported, the function returns English (US). (The code is "`en_us`"). (When ColdFusion is started, it stores the supported locale values in the variable Server.ColdFusion.SupportedLocales.) ColdFusion sets the locale in the JVM to this value; it persists until the server is restarted or it is reset with the `setlocale` function.

Usage: This function does not access a web browser's Accept-Language HTTP header setting.

Example:
```
<h3>GetLocale Example</h3>
<p>The locale for this system is <cfoutput>#GetLocale()#</cfoutput>
```

GetMetaData

Description: Gets metadata (the methods, properties, and parameters of a component) associated with an object that is deployed on the ColdFusion server. This functionality, called introspection, lets applications dynamically determine how to use a component.

Return value: Key-value pairs, as a component descriptor data structure or as structured XML

Category: System functions

Syntax:

```
GetMetaData(object)

or, if used within a ColdFusion component:

GetMetaData(this)
```

History: New in ColdFusion MX: This function is new.

Parameters:

Parameter	Description
object	Reference to an object; use this attribute to call a function from a CFML page
this	Reference to an object; use this attribute to call a function from a component.

Usage: The `this` scope is available at runtime to the component body and to the invoked method's function body. It is used to read and write variables that are present during the life of the component.

Component metadata contains at least the following keys:

- `name`: the component name
- `path`: an absolute path to the component
- `extends`: ancestor component metadata
- `functions`: an array of metadata for each component function

Other component attributes are returned as additional keys.

Function metadata contains at least the following keys:

- `name`: the function name
- `parameters`: an array of argument metadata

Other function attributes are returned as additional keys.

Argument metadata contains at least the following key:

- `name`: the argument name

Other argument attributes are returned as additional keys.

Property metadata contains at least the following key:

- `name`: the property name

Other property attributes are returned as additional keys.

GetMetricData

Description: Gets server performance metrics.

Return value: ColdFusion structure that contains metric data, depending on the `mode` value.

Category: System functions

Syntax: `GetMetricData(mode)`

History: New in ColdFusion MX: the `cachepops` parameter is deprecated. Do not use it in new applications. It might not work, and it might cause an error, in later releases.

Parameters

Parameter	Option	Description
mode	perf_monitor	Returns internal data, in a structure. To receive data, you must enable PerfMonitor in ColdFusion Administrator before executing the function. On Windows, this data is otherwise displayed in the Windows PerfMonitor.
	simple_load	Returns an integer value that is computed from the state of the server's internal queues. Indicates the overall server load.
	prev_req_time	Returns the time, in milliseconds, that it took the server to process the previous request.
	avg_req_time	Returns the average time, in milliseconds, that it takes the server to process a request. Changing the setting to 0 prevents the server from calculating the average and removes overhead associated with gathering data. Default: 120 seconds.

Usage: If `mode = "perf_monitor"`, the function returns a structure with these data fields:

Field	Description
InstanceName	The name of the ColdFusion server. Default: cfserver
PageHits	Number of HTTP requests received since ColdFusion Server was started.
ReqQueued	Number of HTTP requests in the staging queue, waiting for processing.
DBHits	Number of database requests since the server was started.
ReqRunning	Number of HTTP requests currently running. In the ColdFusion Administrator, you can set the maximum number of requests that run concurrently.
ReqTimedOut	Number of HTTP requests that timed out while in the staging queue or during processing.
BytesIn	Number of bytes in HTTP requests to ColdFusion Server
BytesOut	Number of bytes in HTTP responses from ColdFusion Server

Field	Description
AvgQueueTime	For the last two HTTP requests (current and previous), the average length of time the request waited in the staging queue.
AvgReqTime	For the last two HTTP requests (current and previous), the average length of time the server required to process the request
AvgDBTime	For the last two HTTP requests (current and previous), the average length of time the server took to process CFQueries in the request.
cachepops	This parameter is deprecated. Do not use it. ColdFusion automatically sets its value to -1.

Example

```
<!--- This example gets and displays metric data from Windows NT PerfMonitor --->
<cfset pmData = GetMetricData( "PERF_MONITOR" ) >
<cfoutput>
   Current PerfMonitor data is: <p>
   InstanceName:#pmData.InstanceName# <p>
   PageHits:#pmData.PageHits# <p>
   ReqQueued: #pmData.ReqQueued# <p>
   DBHits: #pmData.DBHits# <p>
   ReqRunning: #pmData.ReqRunning# <p>
   ReqTimedOut: #pmData.ReqTimedOut# <p>
   BytesIn: #pmData.BytesIn# <p>
   BytesOut: #pmData.BytesOut# <p>
   AvgQueueTime: #pmData.AvgQueueTime# <p>
   AvgReqTime: #pmData.AvgReqTime# <p>
   AvgDBTime: #pmData.AvgDBTime# <p>
</cfoutput>
```

GetPageContext

Description — Gets the current ColdFusion MX PageContext object that provides access to page attributes and configuration, request and response objects.

Return value — The current ColdFusion MX Java PageContext object.

Category — System functions

Syntax — `GetPageContext()`

History — New in ColdFusion MX: This function is new.

Usage — The ColdFusion MX PageContext class is a wrapper class for the Java PageContext object that can resolve scopes and perform case-insensitive variable lookups.

The PageContext object exposes fields and methods that can be useful in J2EE integration. It includes the include and forward methods that provide the equivalent of the corresponding standard JSP tags. You use these methods to call JSP pages and servlets. For example, you use the following code in CFScript to include the JSP page hello.jsp and pass it a `name` parameter:

```
GetPageContext().include("hello.jsp?name=Bobby"); ==
```

For more information, see your Java Server Pages (JSP) documentation).

Example

```
<!--- this example shows using the page context to set a page
        variable and access the language of the current locale --->
<cfset pc = GetPageContext()>

<cfset pc.setAttribute("name","John Doe")>
<cfoutput>name: #variables.name#<br></cfoutput>

<cfoutput>Language of the current locale is
        #pc.getRequest().getLocale().getDisplayLanguage()#</cfoutput>>.
```

GetProfileSections

Description

Gets all the sections of an initialization file.

An initialization file assigns values to configuration variables, also known as entries, that are set when the system boots, the operating system comes up, or an application starts. An initialization file has the suffix INI; for example, boot.ini, Win32.ini.

Return value

An initialization file, as a struct whose format is as follows:

- Each initialization file section name is a key in the struct
- Each list of entries in a section of an initialization file is a value in the struct

If there is no value, returns an empty string.

Category

System functions

Syntax

```
GetProfileSections(iniFile)
```

See also

GetProfileString, SetProfileString

History

New in ColdFusion MX: This function is new.

Parameters

Parameter	Description
iniFile	Absolute path (drive, directory, filename, extension) of initialization file; for example, C:\boot.ini

GetProfileString

Description

Gets an initialization file entry.

An initialization file assigns values to configuration variables, also known as entries, that are set when the system boots, the operating system comes up, or an application starts. An initialization file has the suffix INI; for example, boot.ini, Win32.ini.

Return value

An entry in an initialization file, as a string. If there is no value, returns an empty string.

Category

System functions

Syntax

`GetProfileString(iniPath, section, entry)`

See also

`GetProfileSections`, `GetProfileString`, `SetProfileString`

Parameters

Parameter	Description
iniPath	Absolute path (drive, directory, filename, extension) of initialization file; for example, C:\boot.ini
section	Section of initialization file from which to extract information
entry	Name of value to get

Example

```
<h3>GetProfileString Example</h3>
Uses GetProfileString to get the value of timeout in an initialization file. Enter
        the full path of your initialization file, and submit the form.
<!--- If the form was submitted, it gets the initialization path and timeout value
        specified in the form --->
<cfif Isdefined("Form.Submit")>
   <cfset IniPath = FORM.iniPath>
   <cfset Section = "boot loader">
   <cfset timeout = GetProfileString(IniPath, Section, "timeout")>
   <h4>Boot Loader</h4>
   <!--- If no entry in an initialization file, nothing displays --->
   <p>Timeout is set to: <cfoutput>#timeout#</cfoutput>.</p>
</cfif>
<form action = "getprofilestring.cfm">
<table cellspacing = "2" cellpadding = "2" border = "0">
   <tr>
      <td>Full Path of Init File</td>
      <td><input type = "Text" name = "IniPath" value = "C:\myboot.ini">
     </td>
   </tr>
   <tr>
    <td><input type = "Submit" name = "Submit" value = "Submit"></td>
    <td></td>
   </tr></table>
</form>
```

GetServiceSettings

Description: Gets the internal Services settings that are available for access; for example, a list of registered ColdFusion data sources.

Services settings include the following:

- Data items that, in ColdFusion 5 and earlier releases, were stored in the registry
- New data items that ColdFusion MX and later releases use
- Internal data items that are not accessible with this function

Return value: The internal data structures that contain Services settings.

Category: System functions

Syntax:

```
GetServiceSettings()
```

History: New in ColdFusion MX: This function is new.

GetTempDirectory

Description: Gets the path of the directory that ColdFusion uses for temporary files. The directory depends on the account under which ColdFusion is running and other factors. Before using this function in an application, test to determine the directory it returns under your account.

Return value: The absolute pathname of a directory, including a trailing slash, as a string.

Category: System functions

Syntax:
```
GetTempDirectory()
```

See also: GetTempFile

History: New in ColdFusion MX: On Windows, this function returns the temporary directory of the embedded Java application server. On other platforms, it returns the temporary directory of the operating system.

Example:
```
<h3>GetTempDirectory Example</h3>

<p>The temporary directory for this ColdFusion server is
        <cfoutput>#GetTempDirectory()#</cfoutput>.
<p>We have created a temporary file called:
<cfoutput>#GetTempFile(GetTempDirectory(),"testFile")#</cfoutput>
```

GetTempFile

Description: Creates a temporary file in a directory whose name starts with (at most) the first three characters of *prefix*.

Return value: Name of a temporary file, as a string.

Category: System functions

Syntax: `GetTempFile(dir, prefix)`

See also: `GetTempDirectory`

Parameters

Parameter	Description
dir	Directory name
prefix	Prefix of a temporary file to create in the directory dir

Example

```
<h3>GetTempFile Example</h3>
<p>The temporary directory for this ColdFusion Server is
        <cfoutput>#GetTempDirectory()#</cfoutput>.
<p>We have created a temporary file called:
<cfoutput>#GetTempFile(GetTempDirectory(),"testFile")#</cfoutput>
```

GetTemplatePath

Description This function is deprecated. Use the `GetBaseTemplatePath` function instead.

Gets the absolute path of an application's base page.

History New in ColdFusion MX: This function is deprecated. Do not use it in new applications. It might not work, and it might cause an error, in later releases.

GetTickCount

Description — Determines the differences between the results of `GetTickCount` at successive points of page processing.

Return value — An integer value

Category — Date and time functions

Syntax — `GetTickCount()`

Usage — This function is useful for timing CFML code segments or other page processing elements.

Example

```
<!--- Setup timing test --->
<cfset iterationCount = 1000>
<!--- Time an empty loop with this many iterations --->
<cfset tickBegin = GetTickCount()>
<cfloop Index = i From = 1 To = #iterationCount#></cfloop>
<cfset tickEnd = GetTickCount()>
<cfset loopTime = tickEnd - tickBegin>

<!--- Report --->
<cfoutput>Loop time (#iterationCount# iterations) was: #loopTime#
   milliseconds</cfoutput>
```

GetTimeZoneInfo

Description

Gets local time zone information for the computer on which it is called, relative to Universal Time Coordinated (UTC). UTC is the mean solar time of the meridian of Greenwich, England, used as the basis for calculating standard time throughout the world.

ColdFusion stores date and time values as date-time objects: real numbers on a universal time line. It converts an object to a time zone when it formats an object; it converts a date/time value from a time zone to a real number when it parses a value.

Return value

Structure that contains these elements and keys:

- utcTotalOffset: offset of local time, in seconds, from UTC
  - A plus sign indicates a time zone west of UTC (such as a zone in North America)
  - A minus sign indicates a time zone east of UTC (such as a zone in Germany)
- utcHourOffset: offset, in hours of local time, from UTC
- utcMinuteOffset: offset, in minutes, beyond the hours offset. For North America, this is 0. For countries that are not exactly on the hour offset, the number is between 0 and 60. For example, standard time in Adelaide, Australia is offset 9 hours and 30 minutes from UTC.
- isDSTOn: True, if Daylight Savings Time (DST) is on in the host; False, otherwise

Category

Date and time functions, International functions

Syntax

```
GetTimeZoneInfo()
```

See also

DateConvert, CreateDateTime, DatePart

Example

```
<h3>GetTimeZoneInfo Example</h3>
<!--- This example shows the use of GetTimeZoneInfo --->
<cfoutput>
The local date and time are #now()#.
</cfoutput>

<cfset info = GetTimeZoneInfo()>
<cfoutput>
<p>Total offset in seconds is #info.utcTotalOffset#.</p>
<p>Offset in hours is #info.utcHourOffset#.</p>
<p>Offset in minutes minus the offset in hours is
  #info.utcMinuteOffset#.</p>
<p>Is Daylight Savings Time in effect? #info.isDSTOn#.</p>
</cfoutput>
```

GetToken

Description — Determines whether a token of the list in the `delimiters` parameter is present in a string.

Return value — The token found at position *index* of the string, as a string. If *index* is greater than the number of tokens in the string, returns an empty string.

Category — String functions

Syntax — `GetToken(string, index [, delimiters ])`

See also — `Left`, `Right`, `Mid`, `SpanExcluding`, `SpanIncluding`

Parameters

Parameter	Description
string	A string or a variable that contains one. String in which to search.
index	Positive integer or a variable that contains one. The position of a token.
delimiters	A string or a variable that contains one. A delimited list of delimiters. Elements may consist of multiple characters. Default list of delimiters: space character, tab character, newline character; or their codes: "chr(32)", "chr(9)", chr(10). Default list delimiter: comma character.

Usage — The following examples show how this function works.

Example A: Consider the following code:

```
GetToken("red,blue::;red,black,tan::;red,pink,brown::;red,three", 2, "::;")
```

This function call requests element number 2 from the string, using the delimiter ":;". The output is as follows:

```
red,black,tan
```

Example B: Consider the following code:

```
<cfset mystring = "four,"
          & #chr(32)# & #chr(9)# & #chr(10)#
          & ",five, nine,zero::;"
          & #chr(10)#
          & "nine,ten:, eleven::;twelve::;thirteen,"
          & #chr(32)# & #chr(9)# & #chr(10)#
          & ",four">
<cfoutput>
          #mystring#<br><br>
</cfoutput>
```

The output is as follows:

```
four,
,five, nine,zero::;
nine,ten:, eleven::;twelve::;thirteen,
,four
```

The gettoken function recognizes explicit spaces, tabs, or newline characters as the parameter delimiters (To specify a space character, the code is chr(32); a tab character, chr(9); and a newline character, chr(10).)

In the example string mystring, there is:

- A forced space between the substrings "four," and ",five"
- A literal space between "five," and "nine"
- A literal space between "ten:," and "eleven,"
- A forced space between "thirteen," and ",four"

In the following call against mystring, no spaces are specified in delimiters (it is omitted), so the function uses the space character as the string delimiter:

```
<br>
<cfoutput>
         gettoken(mystring, 3) is : #gettoken(mystring, 3)#
</cfoutput><br>
```

The output of this code is as follows:

```
gettoken(mystring, 3) is : nine,zero:;
```

The function finds the third delimiter, and returns the substring just before it that is between the second and third delimiter. This substring is "nine,zero:;".

Example C: Consider the following code:

```
<cfset mystring2 = "four,"
         &#chr(9)# & #chr(10)#
         & ",five,nine,zero:;"
         & #chr(10)#
         & "nine,ten:,eleven:;twelve:;thirteen,"
         & #chr(9)# & #chr(10)# & ",four">
<cfoutput>
         #mystring2#<br>
</cfoutput>
```

The output is as follows:

```
four,
,five,nine,zero:;
nine,ten:,eleven:;twelve:;thirteen,
,four
```

The following is a call against mystring2:

```
<cfoutput>
         gettoken(mystring2, 2) is : #gettoken(mystring2, 2)#
</cfoutput>
```

The output is as follows:

```
gettoken(mystring2, 2) is : ,five,nine,zero:;
```

The function finds the second delimiter, and returns the substring just before it that is between the first and second delimiter. This substring is ",five,nine,zero:;".

Example

```
<h3>GetToken Example</h3>
<cfif IsDefined("FORM.yourString")>
<!--- set delimiter --->
<cfif FORM.yourDelimiter is not "">
     <cfset yourDelimiter = FORM.yourDelimiter>
  <cfelse>
     <cfset yourDelimiter = " ">
</cfif>
<!--- check whether number of elements in list is greater than or
         equal to the element sought to return --->
<cfif ListLen(FORM.yourString, yourDelimiter) GTE FORM.returnElement>
  <cfoutput>
  <p>Element #FORM.ReturnElement# in #FORM.yourString#,
  delimited by "#yourDelimiter#"
  <br>is:#GetToken(FORM.yourString, FORM.returnElement, yourDelimiter)#
  </cfoutput>
...
```

Hash

Description: Converts a variable-length string to a 32-byte, hexadecimal string, using the MD5 algorithm. (It is not possible to convert the hash result back to the source string.)

Return value: 32-byte, hexadecimal string

Category: Conversion functions, Other functions, String functions

Syntax: `Hash(string)`

Parameters

Parameter	Description
string	A string or a variable that contains one.

Usage: The result is useful for comparison and validation. For example, a developer can store the hash of a password in a database without exposing the password. The developer can check the validity of the password with the following code:

```
<cfif hash(form.password) is not myQuery.passwordHash>
  <cflocation url = "unauthenticated.cfm">
</cfif>
```

Example:

```
<!--- How to use Hash for password validation. This assumes that UserID
         value is passed to this page with a URL parameter. --->
<h3>Hash Example</h3>

<cfquery name = "CheckPerson" datasource = "UserData">
   SELECT PasswordHash
   FROM SecureData
   WHERE UserID = <cfqueryparam value = "#UserID#"
      cfsqltype = "CF_SQL_CHARVAR">
</cfquery>

<cfif Hash(form.password) is not checkperson.passwordHash>
  <cflocation url = "unauthenticated.cfm">
<cfelse>
   ...
</cfif>
```

Hour

Description: Gets the current hour of the day.

Return value: Ordinal value of the hour, in the range 0 - 23.

Category: Date and time functions

Syntax: `Hour(date)`

See also: `DatePart`, `Minute`, `Second`

Parameters

Parameter	Description
date	Date/time object, in the range 100 AD–9999 AD. See "How ColdFusion processes two-digit year values" on page 377.

Usage: When passing a date/time value as a string, you must enclose it in quotation marks. Otherwise, it is interpreted as a number representation of a date/time object.

Example:

```
<!--- This example shows the use of Hour, Minute, and Second --->
<h3>Hour Example</h3>
<cfoutput>
The time is currently #TimeFormat(Now())#.
We are in hour #Hour(Now())#, Minute #Minute(Now())#
and Second #Second(Now())# of the day.
</cfoutput>
```

HTMLCodeFormat

Description: Replaces special characters in a string with their HTML-escaped equivalents and inserts <pre> and </pre> tags at the beginning and end of the string.

Return value: HTML-escaped string *string* enclosed in <pre> and </pre> tags. Returns are removed from *string*. Special characters (> < " &) are escaped.

Category: Display and formatting functions

Syntax: `HTMLCodeFormat(string [, version ])`

See also: HTMLEditFormat

Parameters

Parameter	Description
string	A string or a variable that contains one.
version	HTML version to use: • -1: The latest implementation of HTML • 2.0: HTML 2.0 (Default) • 3.2: HTML 3.2

Example

```
<!--- Shows the use of HTMLCodeFormat and HTMLEditFormat --->
<h3>HTMLCodeFormat Example</h3>
<form action = "HTMLcodeformat.cfm">
   Try entering a URL for the tag to return in HTMLCodeFormat and HTMLEditFormat:
   <input type = "Text" size = 25 name = "urladdress"
          value = "http://www.macromedia.com">
   <input type = "Submit" name = "" value = "get page">
</form>

<!--- sets a default value for a url to retrieve --->
<cfparam name = "urladdress" default = "http://localhost/cfdocs/index.htm">

<!--- if we passed a url address in FORM, we display it --->
<cfif IsDefined("FORM.urladdress") is True>

<!--- do simple error check to avoid crashing the tag --->
   <cfif Trim(Form.urladdress) is "" or Trim(Form.urladdress) is "http://">
<!--- if error condition tripped, set alternative --->
     <cfset urlAddress = "http://localhost/cfdocs/index.htm">
     <h4>because you entered no url or an empty string, the tag will
         return this address: http://localhost/cfdocs/index.htm</h4>
   <cfelse>
<!--- otherwise use address passed from form --->
     <cfset urlAddress = FORM.urladdress>
   </cfif>
<!--- use the CFHTTP tag to get the file content represented by urladdress --->
     <CFHTTP URL = "#urladdress#"
       method = "GET"
       RESOLVEURL = YES>
     </CFHTTP>
```

```
<cfelse>
<!--- the first time through, retrieve a URL that we know exists --->
<CFHTTP URL = "http://localhost/cfdocs/index.htm"
   method = "GET"
   RESOLVEURL = YES>
</CFHTTP>
</cfif>

<!--- Now, output the file, including the mimetype and content --->
<h3>Show the file</h3>

<cfoutput>
<p>Here is an example of 255 characters from your file output in HTMLCodeFormat:
<p>#HTMLCodeFormat(Mid(CFHTTP.FileContent,1,255))#

<p>Here is an example of 255 characters from your file output in HTMLEditFormat:
<p>#HTMLEditFormat(Mid(CFHTTP.FileContent,1,255))#
</cfoutput>
```

HTMLEditFormat

Description: Replaces special characters in a string with their HTML-escaped equivalents.

Return value: HTML-escaped string *string*. Returns are removed from *string*. Special characters (for example < " &) are escaped.

Category: Display and formatting functions

Syntax: `HTMLEditFormat(string [, version ])`

See also: `HTMLCodeFormat`

Parameters

Parameter	Description
string	A string or a variable that contains one.
version	HTML version to use. • `-1`: The latest implementation of HTML • `2.0`: HTML 2.0 (Default) • `3.2`: HTML 3.2

Usage: This function increases the length of a string. This can cause unpredictable results when performing certain string functions (`Left`, `Right`, and `Mid`, for example) against the expanded string.

Example:

```
<!--- Shows the use of HTMLCodeFormat and HTMLEditFormat --->
<h3>HTMLEditFormat Example</h3>
<form action = "HTMLeditformat.cfm">
   Try entering a URL for the tag to return in HTMLCodeFormat and HTMLEditFormat:
   <input type = "Text" size = 25 name = "urladdress"
          value = "http://www.macromedia.com">
   <input type = "Submit" name = "" value = "get page">
</form>

<!--- sets a default value for a url to retrieve --->
<cfparam name = "urladdress" default = "http://localhost/cfdocs/index.htm">

<!--- if we have passed a url address in the FORM, we want to
         display the passed address --->
<cfif IsDefined("FORM.urladdress") is True>
<!--- do simple error check to avoid crashing the tag --->
   <cfif Trim(Form.urladdress) is "" or Trim(Form.urladdress) is "http://">
<!--- if error condition tripped, set alternative --->
     <cfset urlAddress = "http://localhost/cfdocs/index.htm">
     <h4>because you entered no url or an empty string, the tag will
         return the following address:
     http://localhost/cfdocs/index.htm</h4>

   <cfelse>
<!--- otherwise use address passed from form --->
     <cfset urlAddress = "#FORM.urladdress#">
   </cfif>
```

```
<!--- now use the CFHTTP tag to get the file content represented by urladdress --->
     <CFHTTP URL = "#urladdress#"
       method = "GET"
       RESOLVEURL = YES>
     </CFHTTP>
<cfelse>
<!--- the first time through, retrieve a URL that we know exists --->

<CFHTTP URL = "http://localhost/cfdocs/index.htm"
   method = "GET"
   RESOLVEURL = YES>
</CFHTTP>
</cfif>

<!--- Now, output the file, including the mimetype and content --->
<h3>Show the file</h3>

<cfoutput>
<p>Here is an example of 255 characters from your file
output in HTMLCodeFormat:
<p>#HTMLCodeFormat(Mid(CFHTTP.FileContent,1,255))#

<p>Here is an example of 255 characters from your file
output in HTMLEditFormat:
<p>#HTMLEditFormat(Mid(CFHTTP.FileContent,1,255))#
</cfoutput>
```

IIf

Description
Evaluates a Boolean conditional dynamic expression. Depending on whether the expression is true or false, dynamically evaluates one of two string expressions and returns the result. This function is convenient for incorporating a `cfif` tag in-line in HTML.

For general conditional processing, see `cfif`. For error handling, see `cftry`. For more information, see *Developing ColdFusion MX Applications with CFML*.

Return value
If result is True, returns the value of `Evaluate(string_expression1)`; otherwise, returns the value of `Evaluate(string_expression2)`.

Category
Decision functions, Dynamic evaluation functions

Syntax
`IIf(condition, string_expression1, string_expression2)`

See also
`DE`, `Evaluate`

Parameters

Parameter	Description
condition	An expression that can be evaluated as a Boolean.
string_expression1	A string or a variable that contains one. Expression to evaluate and return if condition is True.
string_expression2	A string or a variable that contains one. Expression to evaluate and return if condition is False.

Usage
The `IIf` function is a shortcut for the following construct:

```
<cfif condition>
   <cfset result = Evaluate(string_expression1)>
<cfelse>
   <cfset result = Evaluate(string_expression2)>
</cfif>
```

The expressions *string_expression1* and *string_expression2* must be string expressions, so that they are not evaluated immediately as the parameters of `IIf`. For example:

```
IIf(y is 0, DE("Error"), x/y)
```

If `y = 0`, this generates an error, because the third expression is the value of x/0 (invalid expression).

ColdFusion evaluates *string_expression1* and *string_expression2*. To return the string itself, use the `DE` function.

Note: If you use pound signs (#) in string_expression1 or string_expression2, ColdFusion evaluates the part of the expression in pound signs first. If you misuse the pound signs, you can cause unexpected results from the `IIf` function. For example, if you use pound signs around the whole expression in string_expression1, and if there is an undefined variable in string_expression1, the function might fail, with the error 'Error Resolving Parameter,'

If a variable is undefined, ColdFusion throws an error when it processes this function. The following example shows this problem:

```
#IIf(IsDefined("Form.Deliver"), DE(Form.Deliver), DE("no"))#
```

This returns "Error resolving parameter FORM.DELIVER".

To avoid this problem, use the DE and Evaluate functions in code such as the following:

```
#IIf(IsDefined("Form.Deliver"), Evaluate(DE("Form.Deliver")), DE("no"))#
```

This returns "no"; ColdFusion does not throw an error.

In the following example, LocalVar is undefined; however, if you omit pound signs around LocalVar, the code works properly:

```
<cfoutput>
  #IIf(IsDefined("LocalVar"), "LocalVar",
     DE("The variable is not defined."))#
</cfoutput>
```

The output is:

```
The variable is not defined.
```

The pound signs around LocalVar in the following code cause it to fail with the error message 'Error Resolving Parameter', because ColdFusion never evaluates the original condition IsDefined("LocalVar").

Here is another example:

```
<cfoutput>
#IIf(IsDefined("LocalVar"), DE("#LocalVar#"), DE("The variable is not defined."))#
</cfoutput>
```

The error message would be as follows:

```
Error resolving parameter LOCALVAR
```

The DE function has no effect on the evaluation of LocalVar, because the pound signs cause it to be evaluated immediately.

Example

```
<h3>IIf Function Example</h3>
<p>IIf evaluates a condition, and does an Evaluate on string
         expression 1 or string expression 2 depending on the Boolean
         outcome <I>(True: run expression 1; False: run expression 2)</I>.</p>
<p>The result of the expression
IIf( Hour(Now()) GTE 12,
  DE("It is afternoon or evening"),
  DE("It is morning"))
is:<br><b>
<cfoutput>
  #IIf( Hour(Now()) GTE 12,
         DE("It is afternoon or evening"),
         DE("It is morning"))#
</cfoutput>
</b>
```

IncrementValue

Description: Adds one to an integer.

Return value: The integer part of *number*, incremented by one.

Category: Mathematical functions

Syntax: `IncrementValue(number)`

See also: DecrementValue

Parameters

Parameter	Description
number	Number to increment

Example

```
<h3>IncrementValue Example</h3>
<p>Returns the integer part of a number incremented by one.
<p>IncrementValue(0): <cfoutput>#IncrementValue(0)#</cfoutput>
<p>IncrementValue("1"): <cfoutput>#IncrementValue("1")#</cfoutput>
<p>IncrementValue(123.35): <cfoutput>#IncrementValue(123.35)#</cfoutput>
```

InputBaseN

Description Converts *string*, using the base specified by *radix*, to an integer.

Return value A number in the range 2-36, as a string.

Category Mathematical functions

Syntax `InputBaseN(string, radix)`

See also `FormatBaseN`

Parameters

Parameter	Description
string	A string or a variable that contains one. String that represents a number, in the base specified by `radix`.
radix	Base of the number represented by `string`, in the range 2 – 36.

Example

```
<h3>InputBaseN Example</h3>

<p>FormatBaseN converts a number to a string in the base specified by Radix.
<p>
<cfoutput>
<br>FormatBaseN(10,2): #FormatBaseN(10,2)#
<br>FormatBaseN(1024,16): #FormatBaseN(1024,16)#
<br>FormatBaseN(125,10): #FormatBaseN(125,10)#
<br>FormatBaseN(10.75,2): #FormatBaseN(10.75,2)#
</cfoutput>
<h3>InputBaseN Example</h3>
<p>InputBaseN returns the number obtained by converting a string,
         using the base specified by Radix,.
<cfoutput>
<br>InputBaseN("1010",2): #InputBaseN("1010",2)#
<br>InputBaseN("3ff",16): #InputBaseN("3ff",16)#
<br>InputBaseN("125",10): #InputBaseN("125",10)#
<br>InputBaseN(1010,2): #InputBaseN(1010,2)#
</cfoutput>
```

Insert

Description: Inserts a substring in a string after a specified character position. If `position` = 0, prefixes the substring to the string.

Return value: A string.

Category: String functions

Syntax: `Insert(substring, string, position)`

See also: `RemoveChars`, `Len`

Parameters

Parameter	Description
substring	A string or a variable that contains one. String to insert.
string	A string or a variable that contains one. String into which to insert `substring`.
position	Integer or variable; position in string after which to insert `substring`.

Example

```
<h3>Insert Example</h3>

<cfif IsDefined("FORM.myString")>
   <!--- if the position is longer than the length of the string, err --->
   <cfif FORM.insertPosition GT Len(MyString)>
      <cfoutput>
      <p>This string only has #Len(MyString)# characters; therefore,
          you cannot insert the substring #FORM.mySubString# at position
          #FORM.insertPosition#.
      </cfoutput>
   </cfif>
<cfelse>
   <cfoutput>
   <p>You inserted the substring #FORM.MySubstring# into the string
         #FORM.MyString#, resulting in the following string:
         <br>#Insert(FORM.MySubString, FORM.myString,
         FORM.insertposition)#
   </cfoutput>
</cfif>
```

Int

Description Calculates the closest integer that is smaller than *number*.

Return value An integer, as a string.

Category Mathematical functions

Syntax `Int(number)`

See also `Ceiling`, `Fix`, `Round`

Parameters

Parameter	Description
number	Real number to round down to an integer

Example

```
<h3>Int Example</h3>
<p>Int returns the closest integer smaller than a number.

<p>Int(11.7) : <cfoutput>#Int(11.7)#</cfoutput>
<p>Int(-11.7) : <cfoutput>#Int(-11.7)#</cfoutput>
<p>Int(0) : <cfoutput>#Int(0)#</cfoutput>
```

IsArray

Description: Determines whether a value is an array.

Return value: True, if *value* is an array, or a query column object.

Category: Array functions, Decision functions

Syntax: `IsArray(value [, number ])`

See also: Array functions

History: New in ColdFusion MX: if the `value` attribute contains a reference to a query result column, this function returns True. For example: `isArray(MyQuery['Column1'])` returns True. (In earlier releases, it returns False.)

New in ColdFusion MX: this function can be used on XML objects.

Parameters

Parameter	Description
value	Variable or array name
number	Dimension; function tests whether the array has exactly this dimension

Usage: This function evaluates a Java array object, such as vector object, as having one dimension.

Example:

```
<h3>IsArray Example</h3>

<!--- Make an array --->
<cfset MyNewArray = ArrayNew(1)>
<!--- set some elements --->
<cfset MyNewArray[1] = "element one">
<cfset MyNewArray[2] = "element two">
<cfset MyNewArray[3] = "element three">
<!--- is it an array? --->
<cfoutput>
   <p>Is this an array? #IsArray(MyNewArray)#
   <p>It has #ArrayLen(MyNewArray)# elements.
   <p>Contents: #ArrayToList(MyNewArray)#
</cfoutput>
```

IsAuthenticated

Description This function is obsolete. Use the newer security tools; see "Authentication functions" on page 328 and the Application Security chapter in *Developing ColdFusion MX Applications with CFML*.

History New in ColdFusion MX: This function is obsolete. It does not work in ColdFusion MX and later ColdFusion releases.

IsAuthorized

Description This function is obsolete. Use the newer security tools; see "Authentication functions" on page 328 and the Application Security chapter in *Developing ColdFusion MX Applications with CFML*

History New in ColdFusion MX: This function is obsolete. It does not work in ColdFusion MX and later releases.

IsBinary

Description: Determines whether a value is stored as binary data.

Return value: True, if the value is binary; False, otherwise. Returns the value in Base64 format.

Category: Decision functions

Syntax: `IsBinary(value)`

See also: ToBinary, ToBase64, IsNumeric, YesNoFormat

Parameters

Parameter	Description
value	Number or string

Example

```
<!--- This example shows the use of IsBinary. It assumes that another page in a
         syndication program passed this page the value of the variable data_var,
         which can hold character or binary data. This example checks whether
         data_var is binary; if so, it converts the data to Base64 data. --->
<h3>IsBinary Example</h3>

<!--- Check whether syndicated data is in binary form. If so, convert to Base64
         so it can be viewed as printable characters. --->
<cfif IsBinary(data_var)>
   <cfset Base64data = ToBase64(data_var)
</cfif>
```

IsBoolean

Description Determines whether a value can be converted to Boolean.

Return value True, if *value* can be converted to Boolean; False, otherwise.

Category Decision functions

Syntax `IsBoolean(value)`

See also `IsNumeric`, `YesNoFormat`

Parameters

Parameter	Description
value	Number or string

Example

```
<h3>IsBoolean Example</h3>

<cfif IsDefined("FORM.theTestValue")>
   <cfif IsBoolean(FORM.theTestValue)>
      <h3>The expression <cfoutput>#DE(FORM.theTestValue)#</cfoutput> is
          Boolean</h3>
   <cfelse>
      <h3>The expression <cfoutput>#DE(FORM.theTestValue)#</cfoutput> is not
          Boolean</h3>
   </cfif>
</cfif>

<form action = "isBoolean.cfm">
<p>Enter an expression, and discover whether it can be evaluated to a
          Boolean value.

<input type = "Text" name = "TheTestValue" value = "1">
<input type = "Submit" value = "Is it Boolean?" name = "">
</form>
```

IsCustomFunction

Description — Determines whether a function be called as a custom function. Displays information about a user-defined function.

Return value — True, if *name* can be called as a custom function; False, otherwise.

Category — Decision functions

Syntax — `IsCustomFunction("name")`

Parameters

Parameter	Description
name	Name of a custom function, between quotation marks.

Usage — The following example generates the following output:

```
realUDF is a function.
```

Example

```
<h3>IsCustomFunction</h3>
<cfscript>
function realUDF() {
  return 1;
}
</cfscript>
<CFSET X = 1>
<!--- example that  fails completely --->
<CFIF IsDefined("Foo") AND IsCustomFunction("Foo")>
  Foo is a UDF.<br>
</CFIF>

<!--- Example that passes, for X, but fails on IsCustomFunction --->
<CFIF IsDefined("X") AND IsCustomFunction("X")>
  X is a UDF.<br>
</CFIF>
<!--- Example that passes --->
<CFIF IsDefined("realUDF") AND IsCustomFunction("realUDF")>
  realUDF is a function.<br>
</CFIF>
```

IsDate

Description: Determines whether a string or Java object can be converted to a date/time value.

Return value: True, if *string* can be converted to a date/time value; otherwise, False. ColdFusion converts the Boolean return value to its string equivalent, "Yes" or "No."

Category: Date and time functions, Decision functions

Syntax: `IsDate(string)`

See also: `CreateDateTime`, `IsNumericDate`, `LSDateFormat`, `LSIsDate`, `ParseDateTime`

Parameters:

Parameter	Description
string	A string or a variable that contains one.

Usage: This function checks only a U.S. date format. For other date support, see `LSDateFormat`.

A date/time object falls in the range 100 AD–9999 AD. See "How ColdFusion processes two-digit year values" on page 377.

Example:

```
<h3>IsDate Example</h3>
<cfif IsDefined("FORM.theTestValue")>
   <cfif IsDate(FORM.theTestValue)>
   <h3>The string <cfoutput>#DE(FORM.theTestValue)#</cfoutput>
    is a valid date</h3>
   <cfelse>
   <h3>The string <cfoutput>#DE(FORM.theTestValue)#</cfoutput>
    is not a valid date</h3>
   </cfif>
</cfif>
<form action = "isDate.cfm">
<p>Enter a string, find whether it can be evaluated to a date value.
<p><input type = "Text" name = "TheTestValue" value = "<cfoutput>#Now()#
         </cfoutput>">
<input type = "Submit" value = "Is it a Date?" name = "">
</form>
```

IsDebugMode

Description: Determines the setting of debug mode in ColdFusion Administrator.

If the debugging service is enabled and the `showDebugOutput` attribute value is Yes, the `IsDebugMode()` function returns Yes; otherwise, No.

Return value: True, if debugging mode is set in the ColdFusion Administrator; False if debugging mode is disabled.

Category: Decision functions

Syntax:

```
IsDebugMode()
```

Example:

```
<h3>IsDebugMode Example</h3>
<cfif IsDebugMode()>
 <h3>Debugging is set in the ColdFusion Administrator</h3>
<cfelse>
 <h3>Debugging is disabled</h3>
</cfif>
```

IsDefined

Description
Evaluates a string value to determine whether the variable named in it exists.
This function is an alternative to the ParameterExists function, which is deprecated.

Return value
True, if the variable is found, or, for a structure, if its key is defined; False, otherwise.
Returns False for an array or structure element referenced using bracket notation. For example, IsDefined("myArray[3]") always returns False, even if the array element myArray[3] has a value.

Category
Decision functions

Syntax
`IsDefined("variable_name")`

See also
Evaluate

History
New in ColdFusion MX: this function can process only the following constructs:
- a simple variable
- a named variable with dot notation
- a named structure with dot notation (example: `mystruct.key`)

Parameters

Parameter	Description
variable_name	String, enclosed in quotation marks. Name of variable to test for.

Usage
When working with the URL and Form scopes, which are structures, the StructKeyExists function can sometimes be used in place of this function. The following blocks of code are equivalent:

Example
```
<h3>IsDefined Example</h3>

<cfif IsDefined("FORM.myString")>
<p>Because the variable FORM.myString is defined, we can show its contents.
         This lets us put a FORM and its resulting action page in the same page,
         while using IsDefined to control the flow of page execution.
<p>The value of "FORM.myString" is <B><I><cfoutput>#FORM.myString#
 </cfoutput></I></B>
<cfelse>
<p>During the first execution of this page, the variable "FORM.myString" is
         not yet defined, so it is not evaluated.
</cfif>
...
```

IsK2ServerABroker

Description: Determines whether the K2Server version is K2 Broker. For more information, see GetK2ServerDocCountLimit on page 448.

This function is used primarily by the ColdFusion Verity and K2Server Administrator pages. This function uses Verity K2Server Release K2.2.0.

Return value: True, if K2Broker is in configured with ColdFusion; False, otherwise.

Category: Decision functions, Full-text search functions, Query functions

Syntax:

```
IsK2ServerABroker()
```

See also: GetK2ServerDocCountLimit, IsK2ServerDocCountExceeded, IsK2ServerOnline

History: New in ColdFusion MX: This function is new.

Example:

```
<cfoutput>IsK2ServerABroker =
        $*#IsK2ServerABroker()#*$</cfoutput>
```

IsK2ServerDocCountExceeded

Description: Determines whether the number of documents that can be searched by the ColdFusion registered K2 Server exceed the limit. Depends on the K2Server platform limit; see GetK2ServerDocCountLimit on page 448.

This function is used primarily by the ColdFusion Verity and K2Server Administrator pages. This function uses Verity K2Server Release K2.2.0.

Return value: True, if the document count limit is exceeded; False, otherwise.

Category: Decision functions, Full-text search functions, Query functions

Syntax:

```
IsK2ServerDocCountExceeded()
```

See also: GetK2ServerDocCount, GetK2ServerDocCountLimit, IsK2ServerABroker

History: New in ColdFusion 5: this function is new.

Example:

```
<cfoutput>IsK2ServerDocCountExceeded =
        $*#IsK2ServerDocCountExceeded()#*$</cfoutput>
```

IsK2ServerOnline

Description: Determines whether the K2Server is running and available to perform a search.

This function is used primarily by the ColdFusion Verity and K2Server Administrator pages.

This function uses Verity K2Server Release K2.2.0.

Return value: True, if the K2Server is available to perform a search; False, otherwise.

Category: Decision functions, Full-text search functions, Query functions

Syntax:

```
IsK2ServerOnline()
```

See also: IsK2ServerABroker

History: New in ColdFusion MX: This function is new.

Example:

```
<cfoutput>IsK2ServerOnline =
        $*#IsK2ServerOnline()#*$</cfoutput>
```

IsLeapYear

Description: Determines whether a year is a leap year.

Return value: True, if *year* is a leap year; otherwise, False.

Category: Date and time functions, Decision functions

Syntax: `IsLeapYear(year)`

See also: DaysInYear

Parameters

Parameter	Description
year	Number representing a year

Example

```
<h3>IsLeapYear Example</h3>

<cfif IsDefined("FORM.theTestValue")>
   <cfif IsLeapYear(FORM.theTestValue)>
      <h3>The year value <cfoutput>#DE(FORM.theTestValue)#</cfoutput> is a Leap
          Year</h3>
   <cfelse>
      <h3>The year value <cfoutput>#DE(FORM.theTestValue)#</cfoutput> is not a Leap
          Year</h3>
   </cfif>
</cfif>

<form action = "isLeapYear.cfm">
<p>Enter a year value, and find out whether it is a Leap Year.
<p><input type = "Text" name = "TheTestValue" value = "
          <cfoutput>#Year(Now())#</cfoutput>">
<input type = "Submit" value = "Is it a Leap Year?" name = "">
</form>
```

IsNumeric

Description — Determines whether a string can be converted to a numeric value. Supports numbers in U.S. number format. For other number support, use LSIsNumeric.

Return value — True, if *string* can be converted to a number; otherwise, False.

Category — Decision functions

Syntax — `IsNumeric(string)`

See also — IsBinary

Parameters

Parameter	Description
string	A string or a variable that contains one.

Example

```
<h3>IsNumeric Example</h3>
<cfif IsDefined("FORM.theTestValue")>
   <cfif IsNumeric(FORM.theTestValue)>
   <h3>The string <cfoutput>#DE(FORM.theTestValue)#</cfoutput> can be
          converted to a number</h3>
   <cfelse>
   <h3>The string <cfoutput>#DE(FORM.theTestValue)#</cfoutput> cannot be
          converted to a number</h3>
   </cfif>
</cfif>

<form action = "isNumeric.cfm">
<p>Enter a string, and find out whether it can be evaluated to a numeric value.

<p><input type = "Text" name = "TheTestValue" value = "123">
<input type = "Submit" value = "Is it a Number?" name = "">
</form>
```

IsNumericDate

Description Evaluates the "real value" of a date/time object.

Return value True, if number represents "real value" of a date/time object; otherwise, False.

Category Date and time functions, Decision functions

Syntax `IsNumericDate(number)`

See also `IsDate`, `ParseDateTime`

Parameters

Parameter	Description
number	A real number

Example

```
<h3>IsNumericDate Example</h3>
<cfif IsDefined("FORM.theTestValue")>
<!--- test if the value is Numeric or a pre-formatted Date value --->
  <cfif IsNumeric(FORM.theTestValue) or IsDate(FORM.theTestValue)>
<!--- if this value is a numericDate value, then pass --->
     <cfif IsNumericDate(FORM.theTestValue)>
     <h3>The string <cfoutput>#DE(FORM.theTestValue)#</cfoutput> is a
          valid numeric date</h3>
     <cfelse>
     <h3>The string <cfoutput>#DE(FORM.theTestValue)#</cfoutput> is not a
          valid numeric date</h3>
     </cfif>
  <cfelse>
  <h3>The string <cfoutput>#DE(FORM.theTestValue)#</cfoutput> is not a
         valid numeric date</h3>
  </cfif>
</cfif>
<form action = "isNumericDate.cfm">
<p>Enter a string, and discover if it can be evaluated to a date value.
<p><input type = "Text" name = "TheTestValue" value = "<cfoutput>#Now()#
 </cfoutput>">
<input type = "Submit" value = "Is it a Date?" name = "">
</form>
```

IsObject

Description: Determines whether a value is an object.

Return value: True, if the value is an object whose attributes match the specified values; False, otherwise.

The basic ColdFusion types are as follows:

- simple: integer, string, float, date, and so on
- structure
- array
- user defined function

Category: Decision functions

Syntax: `IsObject(value [, type [, … ]])`

See also: `IsDate`, `IsQuery`

History: New in ColdFusion MX: This function is new.

Parameters

Parameter	Description
value	A value
type	String; a type name; data type of the argument. • any • array • binary • boolean • date • guid • numeric • query • string • struct • uuid • variableName • a component name If the value is not a recognized type, ColdFusion processes it as a component name
component-specific parameter	See the Usage section.

Usage: This function can take other, context-specific arguments; for example, `classPath`, `server`. For a component, the parameter name is `componentName`.

This object returns False for query and XML objects.

IsProtected

Description: This function is obsolete. Use the newer security tools; see "Authentication functions" on page 328 and the Application Security chapter in *Developing ColdFusion MX Applications with CFML*

History: New in ColdFusion MX: This function is obsolete. It does not work in ColdFusion MX and later releases.

IsQuery

Description: Determines whether *value* is a query.

Return value: True, if *value* is a query.

Category: Decision functions, Query functions

Syntax: `IsQuery(value)`

See also: QueryAddRow

Parameters

Parameter	Description
value	Query variable

Example

```
<!--- Shows an example of IsQuery and IsSimpleValue --->
<h3>IsQuery Example</h3>
<!--- define a variable called "getEmployees" --->
<CFPARAM name = "getEmployees" DEFAULT = "#Now()#">

<p>Before the query is run, the value of GetEmployees is
         <cfoutput>#getEmployees#</cfoutput>

<cfif IsSimpleValue(getEmployees)>
   <p>getEmployees is currently a simple value
</cfif>
<!--- make a query on the snippets datasource --->
<cfquery name = "getEmployees" datasource = "cfsnippets">
SELECT *
FROM employees
</cfquery>

<p>After the query is run, GetEmployees contains a number of rows
         that look like this (display limited to three rows):
<cfoutput QUERY = "GetEmployees" MaxRows = "3">
<pre>#Emp_ID# #FirstName# #LastName#</pre>
</cfoutput>
<cfif IsQuery(getEmployees)>
   GetEmployees is no longer a simple value, but the name of a query
</cfif>
```

IsSimpleValue

Description Determines the type of a value.

Return value True, if value is a string, number, Boolean, or date/time value; False, otherwise.

Category Decision functions

Syntax `IsSimpleValue(value)`

Parameters

Parameter	Description
value	Variable or expression

Example

```
<!--- Shows an example of IsQuery and IsSimpleValue --->
<h3>IsSimpleValue Example</h3>
<!--- define a variable called "getEmployees" --->
<cfparam name = "getEmployees" default = "#Now()#">

<p>Before the query is run, the value of GetEmployees is
         <cfoutput>#getEmployees#</cfoutput>

<cfif IsSimpleValue(getEmployees)>
   <p>getEmployees is currently a simple value
</cfif>
<!--- make a query on the snippets datasource --->
<cfquery name = "getEmployees" datasource = "cfsnippets">
SELECT *
FROM employees
</cfquery>
<p>After the query is run, GetEmployees contains a number of rows
         that look like this (display limited to three rows):
<cfoutput QUERY = "GetEmployees" MaxRows = "3">
<pre>#Emp_ID# #FirstName# #LastName#</pre>
</cfoutput>

<cfif IsQuery(getEmployees)>
   GetEmployees is no longer a simple value, but the name of a query
</cfif>
```

IsStruct

Description: Determines whether a variable is a structure.

Return value: True, if *variable* is a ColdFusion structure or is a Java object that implements the java.lang.Map interface. Returns False if the object in *variable* is a user-defined function (UDF).

Category: Decision functions, Structure functions

Syntax: `IsStruct(variable )`

See also: Structure functions

History: New in ColdFusion MX: this function can be used on XML objects.

Parameters

Parameter	Description
variable	Variable name

Example

```
<!--- This view-only example shows the use of IsStruct. --->
<p>This file is similar to addemployee.cfm, which is called by StructNew,
          StructClear, and StructDelete. It is an example of a custom tag used to
          add employees. Employee information is passed through the employee
          structure (the EMPINFO attribute). In UNIX, you must also add the Emp_ID.
<!---
<cfswitch expression = "#ThisTag.ExecutionMode#">
  <cfcase value = "start">
   <cfif IsStruct(attributes.EMPINFO)>
   <cfoutput>Error. Invalid data.</cfoutput>
   <cfexit method = "ExitTag">
   <cfelse>
   <cfquery name = "AddEmployee" datasource = "cfsnippets">
   INSERT INTO Employees
   (FirstName, LastName, Email, Phone, Department)
   VALUES
   <cfoutput>
   (
   '#StructFind(attributes.EMPINFO, "firstname")#' ,
   '#StructFind(attributes.EMPINFO, "lastname")#' ,
   '#StructFind(attributes.EMPINFO, "email")#' ,
   '#StructFind(attributes.EMPINFO, "phone")#' ,
   '#StructFind(attributes.EMPINFO, "department")#'
   )
   </cfoutput>
   </cfquery>
</cfif>
<cfoutput><hr>Employee Add Complete</cfoutput>
</cfcase>
</cfswitch> --->
```

IsUserInRole

Description: Determines whether an authenticated user belongs to the specified Role.

Return value: True, if the authenticated user, belongs to the specified Role; False, otherwise.

Category: Authentication functions, Decision functions

Syntax: `IsUserInRole("role_name")`

See also: `GetAuthUser`

History: New in ColdFusion MX: This function is new.

Parameters

Parameter	Description
role_name	Name of a security role

Example

```
<cfif isUserInRole("Admin") >
   <cfoutput>Authenticated user is an administrator</cfoutput>
<cfelse isUserInRole("User") >
   <cfoutput>Authenticated user is a user</cfoutput>
</cfif>
```

IsWDDX

Description: Determines whether a value is a well-formed WDDX packet.

Return value: True, if the value is a well-formed WDDX packet; False, otherwise.

Category: Decision functions, XML functions

Syntax: `IsWDDX(value)`

History: New in ColdFusion MX: if the `value` parameter is not a WDDX packet, ColdFusion returns False. (In earlier releases, ColdFusion threw an error.)

Parameters

Parameter	Description
value	A WDDX packet

Usage: This function processes a WDDX packet with a validating XML parser, which uses the WDDX Document Type Definition (DTD).

To prevent CFWDDX deserialization errors, you can use this function to validate WDDX packets from unknown sources.

Example:

```
<cfset packet="
   <wddxPacket version='1.0'>
   <header></header>
   <data>
      <struct>
      <var name='ARRAY'>
         <array length='3'>
            <string>one</string>
         <string>two</string>
         </array>
      </var>
      <var name='NUMBER'>
         <string>5</string>
      </var>
      <var name='STRING'>
         <string>hello</string>
      </var>
      </struct>
   </data>
   </wddxPacket>"
   >
<hr>
<xmp>
<cfoutput>#packet#
</xmp>
IsWDDX() returns #iswddx(packet)#<br>
</cfoutput>
```

IsXmlDoc

Description: Determines whether a function parameter is an Extended Markup language (XML) document object.

Return value: True, if the function argument is an XML document object; False, otherwise.

Category: Decision functions, XML functions

Syntax: `IsXmlDoc(value)`

See also: `IsXmlElement`, `IsXmlRoot`, `XmlChildPos`, `XmlNew`, `XmlParse`, `XmlSearch`, `XmlTransform`

History: New in ColdFusion MX: this function is new.

Parameters

Parameter	Description
value	name of an XML document object

IsXmlElement

Description: Determines whether a function parameter is an Extended Markup language (XML) document object element.

Return value: True, if the function argument is an XML document object element; False, otherwise.

Category: Decision functions, XML functions

Syntax: `IsXmlElement(value)`

See also: `IsXmlDoc`, `IsXmlRoot`, `XmlChildPos`, `XmlNew`, `XmlParse`, `XmlSearch`, `XmlTransform`

History: New in ColdFusion MX: this function is new.

Parameters

Parameter	Description
value	name of an XML document object element

IsXmlRoot

Description: Determines whether a function parameter is the root element of an Extended Markup language (XML) document object.

Return value: True, if the function argument is the root object of an XML document object; False, otherwise.

Category: Decision functions, XML functions

Syntax:
```
IsXmlRoot(value)
```

See also: IsXmlDoc, IsXmlElement, XmlChildPos, XmlNew, XmlParse, XmlSearch, XmlTransform

History: New in ColdFusion MX: this function is new.

Parameters

Parameter	Description
value	name of an XML document object

JavaCast

Description: Converts the data type of a ColdFusion variable to pass as an argument to an overloaded method of a Java object. Use only for scalar and string arguments.

Return value: The variable, as type *type*.

Category: String functions

Syntax: `JavaCast(type, variable)`

See also: `CreateObject`, `cfobject`

Parameters:

Parameter	Description
type	Data type to which to convert variable: • boolean • int • long • double • string
variable	A ColdFusion variable that holds a scalar or string type

Usage: Use after creating a Java object with the `cfobject` tag, before calling one of its methods. If the method takes more than one overloaded argument, you must call `JavaCast` for each one. Use `JavaCast` only when a method is overloaded (because its arguments can take more than one data type, not because the method can take a variable number of arguments).

`JavaCast` cannot be used to cast between complex objects, nor to cast to a super-class. Use this function's result only on calls to Java objects. Because there is not a one-to-one correspondence between internally stored ColdFusion types and Java scalar types, some conversions cannot be performed.

Example: The fooClass method `fooMethod` takes one overloaded argument, as follows:

```
public void fooMethod(String arg);
public void fooMethod(int arg);
```

Within ColdFusion, you use the following code:

```
<cfobject
   type = java
   CLASS = fooClass name = obj>
   <!--- ColdFusion may treat this as a string or a real number --->
   <cfset x = 33>
   <!--- Perform an explicit cast to an int. --->
   <cfset myInt = JavaCast("int", x)>
   <cfset void = fooMethod(myInt)>
   <!--- Perform an explicit cast to a string. --->
   <cfset myString = javaCast("String", x)>
   <cfset void = fooMethod(myString)>
</cfobject>
```

JSStringFormat

Description Escapes special JavaScript characters, such as single quotation mark, double quotation mark, and newline.

Return value A string that is safe to use with JavaScript.

Category String functions

Syntax `JSStringFormat(string)`

Parameters

Parameter	Description
string	A string or a variable that contains one.

Usage Escapes special JavaScript characters, so you can put arbitrary strings safely into JavaScript.

Example

```
<!--- This example shows the use of the JSStringFormat function. ---->
<h3>JSStringFormat</h3>
<cfset stringValue = "An example string value with a tab chr(8),
         a newline (chr10) and some ""quoted"" 'text'">

<p>This is the string we have created:<br>
<cfoutput>#stringValue#</cfoutput>
</p>
<cfset jsStringValue = JSStringFormat(#stringValue#)>
<!----- Generate an alert from the JavaScript string jsStringValue. ---->
<SCRIPT>
s = "<cfoutput>#jsStringValue#</cfoutput>";
alert(s);
</SCRIPT>
```

LCase

Description: Converts the alphabetic characters in a string to lowercase.

Return value: A string, converted to lowercase.

Category: String functions

Syntax: `LCase(string)`

See also: UCase

Parameters

Parameter	Description
string	A string or a variable that contains one

Example

```
<h3>LCase Example</h3>

<cfif IsDefined("FORM.sampleText")>
   <cfif FORM.sampleText is not "">
   <p>Your text, <cfoutput>#FORM.sampleText#</cfoutput>,
   returned in lowercase is <cfoutput>#LCase(FORM.sampleText)#
    </cfoutput>.
   <cfelse>
   <p>Please enter some text.
   </cfif>
</cfif>

<form action = "lcase.cfm">
<p>Enter your text. Press "submit" to see it returned in lowercase:

<p><input type = "Text" name = "SampleText" value = "SAMPLE">
<input type = "Submit" name = "" value = "submit">
</form>
```

Left

Description Determines the number of characters from the beginning of the `string` parameter to the location specified by the `count` parameter.

Return value Number; count of characters.

Category String functions

Syntax `Left(string, count)`

See also `Right`, `Mid`, `Len`

Parameters

Parameter	Description
string	A string or a variable that contains one.
count	A positive integer or a variable that contains one. Number of characters to return.

Example

```
<h3>Left Example</h3>

<cfif IsDefined("Form.MyText")>
<!--- if len is 0, then err --->
   <cfif Len(FORM.myText) is not 0>
      <cfif Len(FORM.myText) LTE FORM.RemoveChars>
      <p>Your string <cfoutput>#FORM.myText#</cfoutput>
      only has <cfoutput>#Len(FORM.myText)#</cfoutput>
      characters. You cannot output the <cfoutput>#FORM.removeChars#
       </cfoutput>
      leftmost characters of this string because it is not long enough.
      <cfelse>
      <p>Your original string: <cfoutput>#FORM.myText#</cfoutput>
      <p>Your changed string, showing only the
       <cfoutput>#FORM.removeChars#</cfoutput> leftmost characters:
      <cfoutput>#Left(Form.myText, FORM.removeChars)#
      </cfoutput>
      </cfif>
   <cfelse>
   <p>Please enter a string
   </cfif>
</cfif>
```

Len

Description: Determines the length of a string or binary object.

Return value: Number; length of a string or a binary object.

Category: String functions

Syntax: `Len(string or binary object)`

See also: `ToBinary`, `Left`, `Right`, `Mid`

History: New in ColdFusion MX: ColdFusion supports the Java UCS-2 representation of Unicode character values 0–65535. (ColdFusion 5 and earlier releases supported ASCII values 1–255. When calculating a length, some string-processing functions processed the ASCII 0 (NUL) character, but did not process subsequent characters of the string.)

Parameters

Parameter	Description
string	A string, the name of a string, or a binary object

Example

```
<h3>Len Example</h3>

<cfif IsDefined("Form.MyText")>
<!--- if len is 0, then err --->
   <cfif Len(FORM.myText) is not 0>
   <p>Your string, <cfoutput>"#FORM.myText#"</cfoutput>,
   has <cfoutput>#Len(FORM.myText)#</cfoutput> characters.
   <cfelse>
   <p>Please enter a string of more than 0 characters.
   </cfif>
</cfif>

<form action = "len.cfm">
<p>Type in some text to see the length of your string.

<br><input type = "Text" name = "MyText">
<br><input type = "Submit" name = "Remove characters">
</form>
```

ListAppend

Description: Concatenates a list or element to a list.

Return value: A copy of the list, with *value* appended. If `delimiter = ""`, returns a copy of the list, unchanged.

Category: List functions

Syntax: `ListAppend(list, value [, delimiters ])`

See also: `ListPrepend`, `ListInsertAt`, `ListGetAt`, `ListLast`, `ListSetAt`

Parameters

Parameter	Description
list	A list or a variable that contains one.
value	An element or a list of elements.
delimiters	A string or a variable that contains one. Character(s) that separate list elements. Default: comma. If this parameter contains more than one character, ColdFusion uses only the first character.

Usage

ColdFusion inserts a delimiter character before *value*.

To add an element to the beginning or end of a list, Macromedia recommends that you do so with code such as the following, rather than with the `listAppend` or `listPrepend` functions:

```
<cfset MyValue = "another element">

<cfif listLen(myList) is 0>
   <cfset myList = MyValue>
<cfelse>
   <cfset myList = myList & ", " & MyValue>
</cfif>
```

The following table shows examples of `ListAppend` processing:

Statement	Output	Comment
`listAppend('elem1,elem2', '' )`	`elem1,elem2,`	Appended element is empty; delimiter is last character in list; list length is 2
`listAppend('', 'elem1,elem2' )`	`elem1,elem2`	List length is 2
`listAppend ("one___two", "three", "___")`	`"one___two_three"`	Inserted the first character of `delimiters` before "three."

Example

```
<h3>ListAppend Example</h3>
<!--- First, query to get some values for our list elements--->
<cfquery name = "GetParkInfo" datasource = "cfsnippets">
SELECT PARKNAME,CITY,STATE
FROM PARKS WHERE PARKNAME LIKE 'AL%'
</cfquery>
```

```
<cfset temp = ValueList(GetParkInfo.ParkName)>
<cfoutput>
<p>The original list: #temp#
</cfoutput>
<!--- now, append a park name to the list --->
<cfset temp2 = ListAppend(Temp, "ANOTHER PARK")>
...
```

ListChangeDelims

Description Changes a list delimiter.

Return value A copy of the list, with each delimiter character replaced by *new_delimiter*.

Category List functions

Syntax `ListChangeDelims(list, new_delimiter [, delimiters ])`

See also `ListFirst`, `ListQualify`

Parameters

Parameter	Description
list	A list or a variable that contains one.
new_delimiter	Delimiter string or a variable that contains one. Can be an empty string. ColdFusion processes the string as one delimiter.
delimiters	A string or a variable that contains one. Character(s) that separate list elements. Default: comma. If this parameter contains more than one character, ColdFusion processes each occurrence of each character as a delimiter.

Example

```
<h3>ListChangeDelims Example</h3>
<p>ListChangeDelims lets you change the delimiters of a list.
<!--- First, query to get some values for our list elements--->
<CFQUERY NAME="GetParkInfo" DATASOURCE="cfsnippets">
SELECT PARKNAME,CITY,STATE
          FROM Parks
          WHERE PARKNAME LIKE 'BA%'
</CFQUERY>
<CFSET temp = ValueList(GetParkInfo.ParkName)>
<cfoutput>
<p>The original list: <p>#temp#
</cfoutput>
<!--- Change the delimiters in the list --->
<CFSET temp2 = ListChangeDelims(Temp, "|:P|", ",")>
<cfoutput>
<p>After executing the statement
          <strong>ListChangeDelims(Temp, "|:P|", ",")</strong>,
          the updated list: <p>#temp2#
</cfoutput>
```

ListContains

Description: Determines the index of the first list element that contains a specified substring.

Return value: Index of the first list element that contains *substring*. If not found, returns zero.

Category: List functions

Syntax: `ListContains(list, substring [, delimiters ])`

See also: ListContainsNoCase, ListFind

Parameters

Parameter	Description
list	A list or a variable that contains one.
substring	A string or a variable that contains one. The search is case-sensitive.
delimiters	A string or a variable that contains one. Character(s) that separate list elements. Default: comma. If this parameter contains more than one character, ColdFusion processes each occurrence of each character as a delimiter.

Usage: ColdFusion ignores empty list elements; thus, the list "a,b,c,,d" has four elements.

Example:

```
<!--- This example shows differences between ListContains and ListFind --->
<!--- Create a list composed of the elements one, two, three. ---->
<cfset aList = "one">
<cfset aList = ListAppend(aList, "two")>
<cfset aList = ListAppend(aList, "three")>
<p>Here is a list: <cfoutput>#aList#</cfoutput>
<p><strong>ListContains</strong> checks for substring "wo" in the list elements:
<cfoutput>
   <p>   Substring "wo" is in
         <B>element #ListContains(aList, "wo")#</B> of list.
</cfoutput>
<p>ListFind cannot check for a substring within an element; therefore, in the
   code, it does not find substring "wo" (it returns 0):
<cfoutput>
   <p>   Substring "wo" is in <b>element #ListFind(aList, "wo")#
   </b> of the list.</cfoutput>
<p><p>If you specify a string that exactly equals an entire list element, such
   as "two", both ListContains and ListFind find it, in the second element:
   <p>   <strong>ListContains</strong>:
<cfoutput>
The string "two" is in <b>element #ListContains(aList, "two")#</b> of the list.
</cfoutput>
   <p><strong>ListFind</strong>:
<cfoutput>
The string "two" is in <b>element #ListFind(aList, "two")#</b> of the list.
</cfoutput>
```

ListContainsNoCase

Description — Determines the index of the first list element that contains a specified substring.

Return value — Index of the first list element that contains *substring*, regardless of case. If not found, returns zero.

Category — List functions

Syntax — `ListContainsNoCase(list, substring [, delimiters ])`

See also — `ListContains`, `ListFindNoCase`

Parameters

Parameter	Description
list	A list or a variable that contains one.
substring	A string or a variable that contains one. The search is case-insensitive.
delimiters	A string or a variable that contains one. Character(s) that separate list elements. Default: comma. If this parameter contains more than one character, ColdFusion processes each occurrence of each character as a delimiter.

Usage — ColdFusion ignores empty list elements; thus, the list "a,b,c,,d" has four elements.

Example

```
<h3>ListContainsNoCase Example</h3>

<cfif IsDefined("FORM.letter")>
   <!--- First, query to get some values for our list elements--->
   <cfquery name = "GetParkInfo" datasource = "cfsnippets">
   SELECT PARKNAME,CITY,STATE
   FROM PARKS
   WHERE PARKNAME LIKE '#FORM.letter#%'
   </cfquery>
   <cfset tempList = ValueList(GetParkInfo.City)>
   <cfif ListContainsNoCase(tempList, FORM.yourCity) is not 0 OR
    FORM.yourCity is "">
   <p><cfif FORM.yourCity is "">The list of parks for the letter
    <cfoutput>#FORM.Letter#</cfoutput>
```

ListDeleteAt

Description: Deletes an element from a list.

Return value: A copy of the list, without the specified element.

Category: List functions

Syntax: `ListDeleteAt(list, position [, delimiters ])`

See also: ListGetAt, ListSetAt, ListLen

Parameters

Parameter	Description
list	A list or a variable that contains one.
position	A positive integer or a variable that contains one. Position at which to delete element. The first list position is 1.
delimiters	A string or a variable that contains one. Character(s) that separate list elements. Default: comma. If this parameter contains more than one character, ColdFusion processes each occurrence of each character as a delimiter.

Usage

To use this and other functions with the default delimiter (comma), you can code as follows:

```
<cfset temp2 = ListDeleteAt(temp, "3")>
```

To specify another delimiter, you code as follows:

```
<cfset temp2 = ListDeleteAt(temp, "3", ";")>
```

ColdFusion ignores empty list elements; thus, the list "a,b,c,,d" has four elements.

Example

```
<!--- First, query to get some values for our list elements--->
<CFQUERY NAME="GetParkInfo" DATASOURCE="cfsnippets">
   SELECT PARKNAME,CITY,STATE
   FROM Parks
   WHERE PARKNAME LIKE 'CHI%'
</CFQUERY>
<CFSET temp = ValueList(GetParkInfo.ParkName)>
<CFSET deleted_element = ListGetAt(temp, "3", ",")>
<cfoutput>
<p>The original list: #temp#
</cfoutput>
<!--- Delete the third element from the list --->
<CFSET temp2 = ListDeleteAt(Temp, "3")>
<cfoutput>
<p>The changed list: #temp2#
<p><I>This list element:<br>#deleted_element#<br> is no longer present
         at position three of the list.</I> </cfoutput>
```

ListFind

Description: Determines the index of the first list element in which a specified value occurs. Case-sensitive.

Return value: Index of the first list element that contains *value*, with matching case. If not found, returns zero. The search is case-sensitive.

Category: List functions

Syntax: `ListFind(list, value [, delimiters ])`

See also: `ListContains`, `ListFindNoCase`

Parameters

Parameter	Description
list	A list or a variable that contains one
value	A string, a number, or a variable that contains one. Item for which to search. The search is case-sensitive.
delimiters	A string or a variable that contains one. Character(s) that separate list elements. Default: comma. If this parameter contains more than one character, ColdFusion processes each occurrence of each character as a delimiter.

Usage: ColdFusion ignores empty list elements; thus, the list "a,b,c,,,d" has four elements.

Example:

```
<!--- Uses ListFind and ListFindNoCase to see if a substring exists in a list --->
<form action="./listfind.cfm" method="POST">
   <p>Try changing the case in Leary's last name:
   <br><input type="Text" size="25" name="myString" value="Leary">
   <p>Pick a search type:
      <select name="type">
         <option value="ListFind" selected>Case-Sensitive
         <option value="ListFindNoCase">Case-Insensitive
      </select>
   <input type="Submit" name="" value="Search Employee List">
</form>

<!--- wait to have a string for searching defined --->
<cfif IsDefined("form.myString") and IsDefined("form.type")>

<cfquery name="SearchEmpLastName" datasource="cfsnippets">
   SELECT  FirstName, RTrim(LastName) AS LName, Phone, Department
   FROM    Employees
</cfquery>

<cfset myList = ValueList(SearchEmpLastName.LName)>
<!--- Is this case-sensitive or case-insensitive searching --->
<cfif form.type is "ListFind">
   <cfset temp = ListFind(myList, form.myString)>
      <cfif temp is 0>
         <h3>An employee with that exact last name was not found</h3>
```

```
      <cfelse>
        <cfoutput>
        <p>Employee #ListGetAt(ValueList(SearchEmpLastName.FirstName), temp)#
        #ListGetAt(ValueList(SearchEmpLastName.LName), temp)#, of the
        #ListGetAt(ValueList(SearchEmpLastName.Department), temp)# Department,
        can be reached at #ListGetAt(ValueList(SearchEmpLastName.Phone), temp)#.
        <p>This was the first employee found under this case-sensitive last name
        search.
        </cfoutput>
      </cfif>
      <cfelse>
        <cfset temp = ListFindNoCase(myList, form.myString)>
        <cfif temp is 0>
          <h3>An employee with that exact last name was not found</h3>
        <cfelse>
          <cfoutput>
          <p>Employee #ListGetAt(ValueList(SearchEmpLastName.FirstName), temp)#
          #ListGetAt(ValueList(SearchEmpLastName.LName), temp)#, of the
          #ListGetAt(ValueList(SearchEmpLastName.Department), temp)# Department,
          can be reached at #ListGetAt(ValueList(SearchEmpLastName.Phone), temp)#.
          <p>This was the first employee found under this case-insensitive last
          name search.
        </cfoutput>
      </cfif>
    </cfif>
</cfif>
```

ListFindNoCase

Description: Determines the index of the first list element in which a specified value occurs.

Return value: Index of the first list element that contains *value*. If not found, returns zero. The search is case-insensitive.

Category: List functions

Syntax: `ListFindNoCase(list, value [, delimiters ])`

See also: ListContains, ListFind

Parameters

Parameter	Description
list	A list or a variable that contains one.
value	Number or string for which to search. The search is case-insensitive.
delimiters	A string or a variable that contains one. Character(s) that separate list elements. Default: comma. If this parameter contains more than one character, ColdFusion processes each occurrence of each character as a delimiter.

Usage: ColdFusion ignores empty list elements; thus, the list "a,b,c,,,d" has four elements.

Example

```
<!--- Uses ListFind and ListFindNoCase to see if a substring exists in a list --->
<form action="./listfind.cfm" method="POST">
   <p>Try changing the case in Leary's last name:
   <br><input type="Text" size="25" name="myString" value="Leary">
   <p>Pick a search type:
      <select name="type">
         <option value="ListFind" selected>Case-Sensitive
         <option value="ListFindNoCase">Case-Insensitive
      </select>
   <input type="Submit" name="" value="Search Employee List">
</form>

<!--- wait to have a string for searching defined --->
<cfif IsDefined("form.myString") and IsDefined("form.type")>

<cfquery name="SearchEmpLastName" datasource="cfsnippets">
   SELECT  FirstName, RTrim(LastName) AS LName, Phone, Department
   FROM    Employees
</cfquery>

<cfset myList = ValueList(SearchEmpLastName.LName)>
<!--- Is this case-sensitive or case-insensitive searching --->
<cfif form.type is "ListFind">
   <cfset temp = ListFind(myList, form.myString)>
      <cfif temp is 0>
         <h3>An employee with that exact last name was not found</h3>
```

```
      <cfelse>
        <cfoutput>
        <p>Employee #ListGetAt(ValueList(SearchEmpLastName.FirstName), temp)#
        #ListGetAt(ValueList(SearchEmpLastName.LName), temp)#, of the
        #ListGetAt(ValueList(SearchEmpLastName.Department), temp)# Department,
        can be reached at #ListGetAt(ValueList(SearchEmpLastName.Phone), temp)#.
        <p>This was the first employee found under this case-sensitive last name
        search.
        </cfoutput>
      </cfif>
      <cfelse>
        <cfset temp = ListFindNoCase(myList, form.myString)>
        <cfif temp is 0>
          <h3>An employee with that exact last name was not found</h3>
        <cfelse>
          <cfoutput>
          <p>Employee #ListGetAt(ValueList(SearchEmpLastName.FirstName), temp)#
          #ListGetAt(ValueList(SearchEmpLastName.LName), temp)#, of the
          #ListGetAt(ValueList(SearchEmpLastName.Department), temp)# Department,
          can be reached at #ListGetAt(ValueList(SearchEmpLastName.Phone), temp)#.
          <p>This was the first employee found under this case-insensitive last
          name search.
        </cfoutput>
      </cfif>
    </cfif>
  </cfif>
```

ListFirst

Description: Gets the first element of a list.

Return value: The first element of a list. If the list is empty, returns an empty string.

Category: List functions

Syntax: `ListFirst(list [, delimiters ])`

See also: ListGetAt, ListLast, ListQualify

Parameters

Parameter	Description
list	A list or a variable that contains one.
delimiters	A string or a variable that contains one. Character(s) that separate list elements. Default: comma. If this parameter contains more than one character, ColdFusion processes each occurrence of each character as a delimiter.

Usage: ColdFusion ignores empty list elements; thus, the list "a,b,c,,d" has four elements.

Example:

```
<h3>ListFirst Example</h3>
<!--- Find a list of users who wrote messages --->
<cfquery name = "GetMessageUser" datasource = "cfsnippets">
   SELECT Username, Subject, Posted
   FROM  Messages
</cfquery>

<cfset temp = ValueList(GetMessageUser.Username)>
<p>Before editing the list, it is: 
<cfoutput>#ValueList(GetMessageUser.Username)#</cfoutput>.
<p>(Users who posted more than once are listed more than once.)
<!--- Show the first user in the list --->
<p>The first user in the list is:
<cfoutput>#ListFirst(temp)#</cfoutput>
<p>The rest of the list is: <cfoutput>#ListRest(temp)#</cfoutput>.
<p>(Users who posted more than once are listed more than once.)
<p>The last user in the list is: <cfoutput>#ListLast(temp)#</cfoutput>
```

ListGetAt

Description: Gets a list element at a specified position.

Return value: Index of the list element at position *position*.

Category: List functions

Syntax: `ListGetAt(list, position [, delimiters ])`

See also: ListFirst, ListLast, ListQualify, ListSetAt

Parameters

Parameter	Description
list	A list or a variable that contains one.
position	A positive integer or a variable that contains one. Position at which to get element. The first list position is 1.
delimiters	A string or a variable that contains one. Character(s) that separate list elements. Default: comma. If this parameter contains more than one character, ColdFusion processes each occurrence of each character as a delimiter.

Usage

If you use list functions on strings that are delimited by a delimiter character and a space, a returned list element might contain a leading space; you use the `trim` function to remove such spaces from a returned element. For example, consider this list:

```
<cfset myList = "one hundred, two hundred, three hundred">
```

To get a value from this list, use the `trim` function to remove the space before the returned value:

```
<cfset MyValue = #trim(listGetAt(myList, 2))#>
```

With this usage, `MyValue` = "`two hundred`", not " `two hundred`", and spaces within a list element are preserved.

ColdFusion ignores empty list elements; thus, the list "a,b,c,,d" has four elements.

Example

```
<h3>ListGetAt Example</h3>
<!--- Find a list of users who wrote messages --->
<cfquery name = "GetMessageUser" datasource = "cfsnippets">
   SELECT Username, Subject, Posted
   FROM  Messages
</cfquery>
<cfset temp = ValueList(GetMessageUser.Username)>
<!--- loop through the list and show it with ListGetAt --->
<h3>This list of usernames who have posted messages numbers
<cfoutput>#ListLen(temp)#</cfoutput> users.</h3>
<ul>
<cfloop From = "1" To = "#ListLen(temp)#" index = "Counter">
   <cfoutput><li>Username #Counter#: #ListGetAt(temp, Counter)# </cfoutput>
</cfloop>
</ul>
```

ListInsertAt

Description Inserts an element in a list.

Return value A copy of the list, with *value* inserted at the specified position.

Category List functions

Syntax `ListInsertAt(list, position, value [, delimiters ])`

See also `ListDeleteAt`, `ListAppend`, `ListPrepend`, `ListSetAt`

Parameters

Parameter	Description
list	A list or a variable that contains one.
position	A positive integer or a variable that contains one. Position at which to insert element. The first list position is 1.
value	An element or a list of elements.
delimiters	A string or a variable that contains one. Character(s) that separate list elements. Default: comma. If this parameter contains more than one character, ColdFusion processes each occurrence of each character as a delimiter.

Usage When inserting an element, ColdFusion inserts a delimiter. If `delimiters` contains more than one delimiter, ColdFusion uses the first delimiter in the string; if `delimiters` is omitted, ColdFusion uses a comma.

ColdFusion ignores empty list elements; thus, the list "a,b,c,,,d" has four elements.

Example

```
<!--- This example shows ListInsertAt --->
<cfquery name = "GetParkInfo" datasource = "cfsnippets">
SELECT PARKNAME,CITY,STATE
FROM PARKS
WHERE PARKNAME LIKE 'DE%'
</cfquery>
<cfset temp = ValueList(GetParkInfo.ParkName)>
<cfset insert_at_this_element = ListGetAt(temp, "3", ",")>
<cfoutput>
<p>The original list: #temp#
</cfoutput>
<cfset temp2 = ListInsertAt(Temp, "3", "my Inserted Value")>
```

ListLast

Description: Gets the last element of a list.

Return value: The last element of the list.

Category: List functions

Syntax: `ListLast(list [, delimiters ])`

See also: ListGetAt, ListFirst

Parameters

Parameter	Description
list	A list or a variable that contains one.
delimiters	A string or a variable that contains one. Character(s) that separate list elements. Default: comma. If this parameter contains more than one character, ColdFusion processes each occurrence of each character as a delimiter.

Usage

If you use list functions on strings that are delimited by a delimiter character and a space, a returned list element might contain a leading space; you use the `trim` function to remove such spaces from a returned element. For example, consider this list:

```
<cfset myList = "one hundred, two hundred, three hundred">
```

To get a value from this list, use the `trim` function to remove the space before the returned value:

```
<cfset MyValue = #trim(listGetAt(myList, 2))#>
```

With this usage, `MyValue = "two hundred"`, not `" two hundred"`, and spaces within a list element are preserved.

ColdFusion ignores empty list elements; thus, the list "a,b,c,,d" has four elements.

Example

```
<h3>ListFirst, ListLast, and ListRest Example</h3>
<!--- Find a list of users who wrote messages --->
<cfquery name = "GetMessageUser" datasource = "cfsnippets">
   SELECT Username, Subject, Posted
   FROM  Messages
</cfquery>
<cfset temp = ValueList(GetMessageUser.Username)>
<p>Before editing the list, it is: 
<cfoutput>#ValueList(GetMessageUser.Username)#</cfoutput>.
<p>(Users who posted more than once are listed more than once.)
<!--- Show the first user in the list --->
<p>The first user in the list is: <cfoutput>#ListFirst(temp)#</cfoutput>
<p>The rest of the list is: <cfoutput>#ListRest(temp)#</cfoutput>.
<p>(Users who posted more than once are listed more than once.)
<p>The last user in the list is: <cfoutput>#ListLast(temp)#</cfoutput>
```

ListLen

Description

Determines the number of elements in a list.

Integer; the number of elements in a list.

Category

List functions

Syntax

`ListLen(list [, delimiters ])`

See also

`ListAppend`, `ListDeleteAt`, `ListInsertAt`, `ListPrepend`

Parameters

Parameter	Description
list	A list or a variable that contains one.
delimiters	A string or a variable that contains one. Character(s) that separate list elements. Default: comma. If this parameter contains more than one character, ColdFusion processes each occurrence of each character as a delimiter.

Usage

ColdFusion ignores empty list elements; thus, the list "a,b,c,,d" has four elements.

Here are some examples of `ListLen` processing:

Statement	Output	Comment
`ListLen('a,b, c,,,d')`	4	Third element is `" c"`
`ListLen('a,b, c,,,d','')`	4	Fourth element is `"d'"`
`ListLen('elem_1___elem_2___elem_3')`	1	
`ListLen('elem*1***elem*2***elem*3')`	1	
`ListLen('elem_1___elem_2___elem_3','_')`	6	

Example

```
<h3>ListLen Example</h3>
<!--- Find a list of users who wrote messages --->
<cfquery name = "GetMessageUser" datasource = "cfsnippets">
   SELECT Username, Subject, Posted
   FROM  Messages
</cfquery>
<cfset temp = ValueList(GetMessageUser.Username)>
<!--- loop through the list and show it with ListGetAt --->
<h3>This is a list of usernames who have posted messages
<cfoutput>#ListLen(temp)#</cfoutput> users.</h3>
<ul>
<cfloop From = "1" TO = "#ListLen(temp)#" INDEX = "Counter">
   <cfoutput><li>Username #Counter#:
    #ListGetAt(temp, Counter)#</cfoutput>
</cfloop>
</ul>
```

ListPrepend

Description: Inserts an element at the beginning of a list.

Return value: A copy of the list, with *value* inserted at the first position.

Category: List functions

Syntax: `ListPrepend(list, value [, delimiters ])`

See also: `ListAppend`, `ListInsertAt`, `ListSetAt`

Parameters

Parameter	Description
list	A list or a variable that contains one.
value	An element or a list of elements.
delimiters	A string or a variable that contains one. Character(s) that separate list elements. Default: comma. If this parameter contains more than one character, ColdFusion processes each occurrence of each character as a delimiter.

Usage

When prepending an element to a list, ColdFusion inserts a delimiter. If `delimiters` *contains more* than one delimiter character, ColdFusion uses the first delimiter in the string; if `delimiters` is omitted, ColdFusion uses a comma.

ColdFusion ignores empty list elements; thus, the list "a,b,c,,d" has four elements.

If the `delimiters` parameter is empty (""), ColdFusion replaces the list with the contents of `value`.

To add an element to the beginning or end of a list, Macromedia recommends that you do so with code such as the following, rather than with the `listAppend` or `listPrepend` functions:

```
<cfset MyValue = "another element">

<cfif listLen(myList) is 0>
   <cfset myList = MyValue>
<cfelse>
   <cfset myList = myList & ", " & MyValue>
</cfif>
```

Example

```
<!--- This example shows ListPrepend --->
<cfquery name = "GetParkInfo" datasource = "cfsnippets">
   SELECT PARKNAME,CITY,STATE
   FROM PARKS
   WHERE PARKNAME LIKE 'DE%'
</cfquery>
<cfset temp = ValueList(GetParkInfo.ParkName)>
<cfset first_element = ListFirst(temp)>
<cfoutput><p>The original list: #temp#</cfoutput>
<!--- now, insert an element at position 1--->
<cfset temp2 = ListPrepend(Temp, "my Inserted Value")>
```

ListQualify

Description: Inserts a string at the beginning and end of list elements.

Return value: A copy of the list, with *qualifier* before and after the specified element(s).

Category: List functions

Syntax: `ListQualify(list, qualifier [, delimiters ] [, elements ])`

History: New in ColdFusion MX: As the `elements` parameter value, you must specify `"all"` or `"char"`; otherwise, ColdFusion throws an exception. (In earlier releases, the function ignored an invalid value, and used `"all"`; this was inconsistent with other functions.)

Parameters

Parameter	Description
list	A list or a variable that contains one.
qualifier	A string or a variable that contains one. Character or string to insert before and after the list elements specified in the `elements` attribute.
delimiters	A string or a variable that contains one. Character(s) that separate list elements. Default: comma. If this parameter contains more than one character, ColdFusion processes each occurrence of each character as a delimiter.
elements	• all: all elements • char: elements that are composed of alphabetic characters

Usage: The new list might not preserve all of the delimiters in the list.

ColdFusion ignores empty list elements; thus, the list "a,b,c,,,d" has four elements.

Example:

```
<cfquery name = "GetEmployeeNames" datasource = "cfsnippets">
SELECT FirstName, LastName
FROM Employees
</cfquery>

<h3>ListQualify Example</h3>
<p>This example uses ListQualify to put the full names of the
         employees in the query within quotation marks.</p>
<cfset myArray = ArrayNew(1)>

<!--- loop through query; append these names successively to the last element --->
<cfloop query = "GetEmployeeNames">
   <cfset temp = ArrayAppend(myArray, "#FirstName# #LastName#")>
</cfloop>

<!--- sort that array descending alphabetically --->
<cfset myAlphaArray = ArraySort(myArray, "textnocase")>

<!--- show the resulting array as a list --->
<cfset myList = ArrayToList(myArray, ",")>
```

```
<cfoutput>
   <p>The contents of the unqualified list are as follows:</p>
   #myList#
</cfoutput>

<!--- show the resulting alphabetized array as a qualified list with
         single quotes around each full name.    --->
<cfset qualifiedList1 = ListQualify(myList,"'",",","CHAR")>

<!--- output the array as a list --->
<cfoutput>
   <p>The contents of the qualified list are as follows:</p>
   <p>#qualifiedList1#</p>
</cfoutput>

<!--- show the resulting alphabetized array as a qualified list with quotation
         marks around each full name. We use &quot; to denote quotation marks
         because the quotation mark character is a control character. --->
<cfset qualifiedList2 = ListQualify(myList,"&quot;",",","CHAR")>

<!--- output the array as a list --->
<cfoutput>
   <p>The contents of the second qualified list are:</p>
   <p>#qualifiedList2#</p>
</cfoutput>
```

ListRest

Description: Gets a list, without its first element.

Return value: A copy of *list*, without the first element. If *list* has one element, returns an empty list.

Category: List functions

Syntax: `ListRest(list [, delimiters ])`

See also: ListFirst, ListGetAt, ListLast

Parameters

Parameter	Description
list	A list or a variable that contains one.
delimiters	A string or a variable that contains one. Character(s) that separate list elements. Default: comma. If this parameter contains more than one character, ColdFusion processes each occurrence of each character as a delimiter.

Usage: If the list begins with one or more empty entries, this function drops them, as well as the first element.

ColdFusion ignores empty list elements; thus, the list "a,b,c,,d" has four elements.

Example:

```
<h3>ListFirst, ListLast, and ListRest Example</h3>
<!--- Find a list of users who wrote messages --->
<cfquery name = "GetMessageUser" datasource = "cfsnippets">
   SELECT Username, Subject, Posted
   FROM  Messages
</cfquery>

<cfset temp = ValueList(GetMessageUser.Username)>
<p>Before editing the list, it is: 
<cfoutput>#ValueList(GetMessageUser.Username)#</cfoutput>.
<p>(Users who posted more than once are listed more than once.)
<p>The first user in the list is:
<cfoutput>#ListFirst(temp)# </cfoutput>
<p>The rest of the list is: <cfoutput>#ListRest(temp)#</cfoutput>.
<p>(Users who posted more than once are listed more than once.)
<p>The last user in the list is: <cfoutput>#ListLast(temp)#</cfoutput>
```

ListSetAt

Description: Assigns a value to a list element.

Return value: A copy of a list, with a value assigned to the element at a specified position.

Category: List functions

Syntax: `ListSetAt(list, position, value [, delimiters ])`

See also: `ListDeleteAt`, `ListGetAt`, `ListInsertAt`

History: New in ColdFusion MX: ColdFusion MX does not modify delimiters in the list. (In earlier releases, in some cases, replaced delimiters with the first character in the `delimiters` parameter.)

Parameters

Parameter	Description
list	A list or a variable that contains one.
position	A positive integer or a variable that contains one. Position at which to set a value. The first list position is 1.
value	An element or a list of elements.
delimiters	A string or a variable that contains one. Character(s) that separate list elements. Default: comma. If this parameter contains more than one character, ColdFusion processes each occurrence of each character as a delimiter.

Usage: When assigning an element to a list, ColdFusion inserts a delimiter. If `delimiters` contains more than one delimiter, ColdFusion uses the first delimiter in the string, or, if `delimiters` was omitted, a comma.

ColdFusion ignores empty list elements; thus, the list "a,b,c,,,d" has four elements.

Example:

```
<h3>ListSetAt Example</h3>
<!--- Find a list of users who wrote messages --->
<cfquery name = "GetMessageUser" datasource = "cfsnippets">
SELECT Username, Subject, Posted
FROM  Messages
</cfquery>

<cfset temp = ValueList(GetMessageUser.Subject)>

<!--- loop through the list and show it with ListGetAt --->
<h3>This is a list of <cfoutput>#ListLen(temp)#</cfoutput>
subjects posted in messages.</h3>

<cfset ChangedElement = ListGetAt(temp, 2)>
<cfset TempToo = ListSetAt(temp, 2, "I changed this subject", ",")>
<ul>
<cfloop From = "1" To = "#ListLen(temptoo)#" INDEX = "Counter">
   <cfoutput><li>(#Counter#) SUBJECT: #ListGetAt(temptoo, Counter)#
   </cfoutput>
```

```
</cfloop>
</ul>
<p>Note that element 2, "<cfoutput>#changedElement#</cfoutput>",
          has been altered to "I changed this subject" using ListSetAt.
```

ListSort

Description: Sorts list elements according to a sort type and sort order.

Return value: A copy of a list, sorted.

Category: List functions

Syntax: `ListSort(list, sort_type [, sort_order] [, delimiters ])`

History: New in ColdFusion MX: in a `textnocase`, descending sort, this function might return elements in a different sort order than in earlier releases. If `sort_type = "textnocase"` and `sort_order = "desc"`, ColdFusion MX processes elements that *differ only in case* differently from earlier releases. ColdFusion MX outputs the elements in the reverse of the ascending order. Earlier releases do not change order of elements that differ only in case. Both operations are correct. The new operation ensures that an ascending and descending sort output elements in exactly reverse order.

For example, in a `textnocase`, `desc` sort of `d,a,a,b,A`, the following occurs:

- ColdFusion MX returns `d,b,A,a,a`
- Earlier ColdFusion releases return `d,b,a,a,A`

(In a `textnocase`, *`asc`* sort, all ColdFusion releases return `a,a,A,b,d`.)

Parameters

Parameter	Description
list	A list or a variable that contains one.
sort_type	• numeric: sorts numbers • text: sorts text alphabetically, taking case into account (also known as case sensitive). All letters of one case precede the first letter of the other case: - aabzABZ, if `sort_order = "asc"` (ascending sort) - ZBAzbaa, if `sort_order = "desc"` (descending sort) • textnocase: sorts text alphabetically, without regard to case (also known as case-insensitive). A letter in varying cases precedes the next letter: - aAaBbBzzZ, in an ascending sort; preserves original intra-letter order - ZzzBbBaAa, in a descending sort; reverses original intra-letter order
sort_order	• asc - ascending sort order. Default. - aabzABZ or aAaBbBzzZ, depending on value of `sort_type`, for letters - from smaller to larger, for numbers • desc - descending sort order. - ZBAzbaa or ZzzBbBaAa, depending on value of `sort_type`, for letters - from larger to smaller, for numbers
delimiters	A string or a variable that contains one. Character(s) that separate list elements. Default: comma. If this parameter contains more than one character, ColdFusion processes each occurrence of each character as a delimiter.

Usage: ColdFusion ignores empty list elements; thus, the list "a,b,c,,,d" has four elements.

Example

```
<h3>ListSort Example</h3>

<!--- Find a list of users who wrote messages --->
<cfquery name = "GetMessageUser" datasource = "cfsnippets">
SELECT Username, Subject, Posted
FROM Messages
</cfquery>

<cfset myList = ValueList(GetMessageUser.UserName)>
<p>Here is the unsorted list. </p>
<cfoutput>#myList#
  </cfoutput>
<p>Here is the list sorted alphabetically:</p>
<cfset sortedList = ListSort(myList, "Text")>
<cfoutput>#sortedList#
  </cfoutput>

<p>Here is a numeric list that is to be sorted in descending order.</p>
<cfset sortedNums = ListSort("12,23,107,19,1,65","Numeric", "Desc")>
<cfoutput>#sortedNums# </cfoutput>

<p>Here is a list that must be sorted numerically, since it contains
         negative and positive numbers, and decimal numbers. </p>
<cfset sortedNums2 = ListSort("23.75;-34,471:100,-9745","Numeric", "ASC", ";,:")>
<cfoutput>#sortedNums2# </cfoutput>

<p>Here is a list to be sorted alphabetically without consideration of case.</p>
<cfset sortedMix =
  ListSort("hello;123,HELLO:jeans,-345,887;ColdFusion:coldfusion",
     "TextNoCase", "ASC", ";,:")>
<cfoutput>#sortedMix# </cfoutput>
```

ListToArray

Description: Copies the elements of a list to an array.

Return value: An array

Category: Array functions, Conversion functions, List functions

Syntax: `ListToArray(list [, delimiters ])`

See also: ArrayToList

Parameters

Parameter	Description
list	A list or a variable that contains one. You define a list variable with a `cfset` statement.
delimiters	A string or a variable that contains one. Character(s) that separate list elements. Default: comma. If this parameter contains more than one character, ColdFusion processes each occurrence of each character as a delimiter.

Usage: ColdFusion ignores empty list elements; thus, the list "a,b,c,,d" has four elements.

Example:

```
<h3>ListToArray Example</h3>
<!--- Find a list of users who wrote messages --->
<cfquery name = "GetMessageUser" datasource = "cfsnippets">
SELECT Username, Subject, Posted
FROM  Messages
</cfquery>
<cfset myList = ValueList(GetMessageUser.UserName)>
<p>My list is a list with <cfoutput>#ListLen(myList)#</cfoutput>
 elements.
<cfset myArrayList = ListToArray(myList)>
<p>My array list is an array with <cfoutput>#ArrayLen(myArrayList)#
 </cfoutput> elements.
```

ListValueCount

Description: Counts instances of a specified value in a list. The search is case-sensitive.

Return value: The number of instances of *value* in the list.

Category: List functions, String functions

Syntax: `ListValueCount(list, value [, delimiters ])`

See also: ListValueCountNoCase

Parameters

Parameter	Description
list	A list or a variable that contains one.
value	String or number or a variable that contains one. Item for which to search. The search is case-sensitive.
delimiters	A string or a variable that contains one. Character(s) that separate list elements. Default: comma. If this parameter contains more than one character, ColdFusion processes each occurrence of each character as a delimiter.

Example

```
<cfquery name = "SearchByDepartment" datasource = "cfsnippets">
SELECT Department
FROM Employees
</cfquery>
<h3>ListValueCount Example</h3>
<p>This example uses ListValueCount to count employees in a department.

<form action = "listvaluecount.cfm">
<p>Select a department:</p>
   <select name = "departmentName">
         <option value = "Accounting">
            Accounting
         </OPTION>
         <option value = "Administration">
            Administration
         </OPTION>
         <option value = "Engineering">
            Engineering
         </OPTION>
         <option value = "Sales">
            Sales
         </OPTION>
   </select>
<input type = "Submit" name = "Submit" value = "Search Employee List">
</form>

<!--- wait to have a string for searching defined --->
<cfif IsDefined("FORM.Submit") and IsDefined("FORM.departmentName")>
   <cfset myList = ValueList(SearchByDepartment.Department)>
   <cfset numberInDepartment = ListValueCount(myList, FORM.departmentName)>
```

```
    <cfif numberInDepartment is 0>
      <h3>There are no employees in <cfoutput>#FORM.departmentName#</cfoutput></h3>
    <cfelseIf numberInDepartment is 1>
      <cfoutput><p>There is only one person in #FORM.departmentName#.
      </cfoutput>
    <cfelse>
      <cfoutput><p>There are #numberInDepartment# people in #FORM.departmentName#.
      </cfoutput>
    </cfif>
  </cfif>
```

ListValueCountNoCase

Description: Counts instances of a specified value in a list. The search is case-insensitive.

Return value: The number of instances of *value* in the list.

Category: List functions

Syntax: `ListValueCountNoCase(list, value [, delimiters ])`

See also: ListValueCount

Parameters

Parameter	Description
list	A list or a variable that contains one.
value	String or number or a variable that contains one. Item for which to search. The search is case-insensitive.
delimiters	A string or a variable that contains one. Character(s) that separate list elements. Default: comma. If this parameter contains more than one character, ColdFusion processes each occurrence of each character as a delimiter.

Example

```
<cfquery name = "SearchByDepartment" datasource = "cfsnippets">
SELECT Department
FROM Employees
</cfquery>

<h3>ListValueCountNoCase Example</h3>
<p>This example uses ListValueCountNoCase to count employees in a department.

<form action = "listvaluecountnocase.cfm">
<p>Select a department:</p>
   <select name = "departmentName">
         <option value = "Accounting">
            Accounting
         </OPTION>
         <option value = "Administration">
            Administration
         </OPTION>
         <option value = "Engineering">
            Engineering
         </OPTION>
         <option value = "Sales">
            Sales
         </OPTION>
   </select>
   </select>
<input type = "Submit" name = "Submit" value = "Search Employee List">
</form>
```

```
<!--- wait to have a string for searching defined --->
<cfif IsDefined("FORM.Submit") and IsDefined("FORM.departmentName")>
   <cfset myList = ValueList(SearchByDepartment.Department)>
   <cfset numberInDepartment = ListValueCountNoCase(myList,
      FORM.departmentName)>

   <cfif numberInDepartment is 0>
      <h3>There are no employees in <cfoutput>#FORM.departmentName#</cfoutput></h3>
   <cfelseIf numberInDepartment is 1>
      <cfoutput><p>There is only one person in #FORM.departmentName#.
      </cfoutput>
   <cfelse>
      <cfoutput><p>There are #numberInDepartment# people in #FORM.departmentName#.
      </cfoutput>
   </cfif>
</cfif>
```

LJustify

Description Left justifies characters in a string of a specified length.

Return value A copy of a string, left-justified.

Category Display and formatting functions, String functions

Syntax `LJustify(string, length)`

See also `CJustify`, `RJustify`

Parameters

Parameter	Description
string	A string or a variable that contains one
length	Length of field in which to justify string

Example

```
<!--- This example shows how to use LJustify --->
<cfparam name = "jstring" default = "">

<cfif IsDefined("FORM.justifyString")>
   <cfset jstring = Ljustify(FORM.justifyString, 35)>
</cfif>
<html>
<head>
   <title>LJustify Example</title>
</head>
<body>

<h3>LJustify Function</h3>
<p>Enter a string, and it will be left justified within the sample field

<form action = "ljustify.cfm">
<p><input type = "Text" value = "<cfoutput>#jString#</cfoutput>"
 size = 35 name = "justifyString">

<p><input type = "Submit" name = ""> <input type = "RESET">
</form>
```

Log

Description Calculates the natural logarithm of a number. Natural logarithms are based on the constant e (2.71828182845904).

Return value The natural logarithm of a number.

Category Mathematical functions

Syntax `Log(number)`

See also `Exp`, `Log10`

Parameters

Parameter	Description
number	Positive real number for which to calculate the natural logarithm

Example

```
<h3>Log Example</h3>

<cfif IsDefined("FORM.number")>
   <cfoutput>
      <p>Your number, #FORM.number#
      <br>#FORM.number# raised to the E power: #exp(FORM.number)#
      <cfif FORM.number LTE 0><br>Enter a positive real number to get its
          natural logarithm
        <cfelse><br>The natural logarithm of #FORM.number#: #log(FORM.number)#
      </cfif>
      <cfif FORM.number LTE 0><br>Enter a positive real number to get its
          logarithm to base 10
        <cfelse><br>The logarithm of #FORM.number# to base 10: #log10(FORM.number)#
      </cfif>
   </cfoutput>
</cfif>
<cfform action = "log.cfm">
Enter a number to see its value raised to the E power, its natural logarithm,
          and the logarithm of number to base 10.
<cfinput type = "Text" name = "number" message = "You must enter a number"
          validate = "float" required = "No">
<input type = "Submit" name = "">
</cfform>
```

Log10

Description: Calculates the logarithm of *number*, to base 10.

Return value: Number; the logarithm of *number*, to base 10.

Category: Mathematical functions

Syntax: `Log10(number)`

See also: `Exp`, `Log`

Parameters

Parameter	Description
number	Positive real number for which to calculate the logarithm

Example

```
<h3>Log10 Example</h3>
<cfif IsDefined("FORM.number")>
   <cfoutput>
      <p>Your number, #FORM.number#
      <br>#FORM.number# raised to the E power: #exp(FORM.number)#
      <cfif FORM.number LTE 0><br>You must enter a positive real number to
          see the natural logarithm of that number
      <cfelse><br>The natural logarithm of #FORM.number#: #log(FORM.number)#
      </cfif>
      <cfif #FORM.number# LTE 0><br>You must enter a positive real number to
          see the logarithm of that number to base 10
      <cfelse><br>The logarithm of #FORM.number# to base 10: #log10(FORM.number)#
      </cfif>
   </cfoutput>
</cfif>
<cfform action = "log10.cfm">
Enter a number to find its value raised to the E power, its natural
          logarithm, and the logarithm of number to base 10.
<cfinput type = "Text" name = "number" message = "You must enter a number"
          validate = "float" required = "No">
<input type = "Submit" name = "">
</cfform>
```

LSCurrencyFormat

Description: Formats a number in a locale-specific currency format. To manage euro currency values, use the `LSEuroCurrencyFormat` function.

Return value: A formatted currency value.

Category: Display and formatting functions, International functions

Syntax: `LSCurrencyFormat(number [, type ])`

See also: `LSEuroCurrencyFormat`, `SetLocale`

History: New in ColdFusion MX: This function might return different formatting than in earlier releases. If a negative number is passed to it, it returns a negative number. If `type = "local"`, it returns the value in the current locale's standard format. If `type = "international"`, it returns the value in the current locale's international standard format. This function uses Java standard locale formatting rules on all platforms.

Parameters

Parameter	Description
number	Currency value
type	• local: the currency format and currency symbol used in the locale. - With JDK 1.3, the default for Euro Zone countries is their local currency. - With JDK 1.4, the default for Euro Zone countries is the euro. • international: the international standard currency format and currency symbol of the locale. • none: the currency format used in the locale; no currency symbol

Usage: This function uses Java standard locale formatting rules on all platforms.

Currency output

The following table shows sample currency output:

Locale	Type = Local	Type = International	Type = None
Dutch (Belgian)	100.000,00 BF	BEF100.000,00	100.000,00
Dutch (Standard)	fl 100.000,00	NLG100.000,00	100.000,00
English (Australian)	$100,000.00	AUD100,000.00	100,000.00
English (Canadian)	$100,000.00	CAD100,000.00	100,000.00
English (New Zealand)	$100,000.00	NZD100,000.00	100,000.00
English (UK)	£100,000.00	GBP100,000.00	100,000.00
English (US)	$100,000.00	USD100,000.00	100,000.00
French (Belgian)	100.000,00 FB	BEF100.000,00	100.000,00
French (Canadian)	100 000,00 $	CAD100 000,00	100 000,00
French (Standard)	100 000,00 F	FRF100 000,00	100 000,00

Locale	Type = Local	Type = International	Type = None
French (Swiss)	SFr. 100'000.00	CHF100'000.00	100'000.00
German (Austrian)	öS 100.000,00	ATS100.000,00	100.000,00
German (Standard)	100.000,00 DM	DEM100.000,00	100.000,00
German (Swiss)	SFr. 100'000.00	CHF100'000.00	100'000.00
Italian (Standard)	L. 10.000.000	ITL10.000.000	10.000.000
Italian (Swiss)	SFr. 100'000.00	CHF100'000.00	100'000.00
Norwegian (Bokmal)	kr 100 000,00	NOK100 000,00	100 000,00
Norwegian (Nynorsk)	kr 100 000,00	NOK100 000,00	100 000,00
Portuguese (Brazilian)	R$100.000,00	BRC100.000,00	100.000,00
Portuguese (Standard)	R$100.000,00	BRC100.000,00	100.000,00
Spanish (Mexican)	$100,000.00	MXN100,000.00	100,000.00
Spanish (Modern)	10.000.000 Pts	ESP10.000.000	10.000.000
Spanish (Standard)	10.000.000 Pts	ESP10.000.000	10.000.000
Swedish	100.000,00 kr	SEK100.000,00	100.000,00

Note: ColdFusion maps Spanish (Modern) to the Spanish (Standard) format.

To set the default display format of date, time, number, and currency values, use the `SetLocale` function.

Example

```
<h3>LSCurrencyFormat Example</h3>
<p>LSCurrencyFormat returns a currency value using the locale
        convention. Default value is "local."
<!--- loop through list of locales; show currency values for 100,000 units --->
<cfloop LIST = "#Server.Coldfusion.SupportedLocales#"
index = "locale" delimiters = ",">
  <cfset oldlocale = SetLocale(locale)>
  <cfoutput><p><b><I>#locale#</I></b><br>
    Local: #LSCurrencyFormat(100000, "local")#<br>
    International: #LSCurrencyFormat(100000, "international")#<br>
    None: #LSCurrencyFormat(100000, "none")#<br>
    <hr noshade>
  </cfoutput>
</cfloop>
```

LSDateFormat

Description: Formats the date part of a date/time value in a locale-specific format.

Return value: A formatted date/time value. If no mask is specified, the value is formatted according to the locale setting of the client computer.

Category: Date and time functions, Display and formatting functions, International functions

Syntax: `LSDateFormat(date [, mask ])`

History: New in ColdFusion MX:

- This function might return different formatting than in earlier releases. This function uses Java standard locale formatting rules on all platforms.
- This function supports the `short`, `medium`, `long`, and `full mask` attribute options.

Parameters

Parameter	Description
date	A date/time object, in the range 100 AD–9999 AD. See "How ColdFusion processes two-digit year values" on page 377.
mask	Characters that show how ColdFusion displays the date: • d: Day of month. Digits; no leading zero for single-digit days • dd: Day of month. Digits; leading zero for single-digit days • ddd: Day of week. Three-letter abbreviation • dddd: Day of week. Full name • m: Month. Digits; no leading zero for single-digit months • mm: Month. Digits; leading zero for single-digit months • mmm: Month. Three-letter abbreviation • mmmm: Month. Full name • y: Year. Last two digits; no leading zero for years less than 10 • yy: Year. Last two digits; leading zero for years less than 10 • yyyy: Year. Four digits • gg: Period/era string. Not processed. Reserved for future use • short • medium • long • full Default: the Java standard Medium format for the locale. For more information on formats, see LSParseDateTime on page 553.

Usage: This function uses Java standard locale formatting rules on all platforms.

When passing date/time value as a string, enclose it in quotation marks. Otherwise, it is interpreted as a number representation of a date/time object.

To calculate a difference between time zones, use the `GetTimeZoneInfo` function.

To set the default display format of date, time, number, and currency values, use the `SetLocale` function.

Example

```
<h3>LSDateFormat Example</h3>
<p>LSDateFormat formats the date part of a date/time value using the
        locale convention.
<!--- loop through a list of locales; show date values for Now()--->
<cfloop list = "#Server.Coldfusion.SupportedLocales#"
index = "locale" delimiters = ",">
  <cfset oldlocale = SetLocale(locale)>

  <cfoutput><p><B><I>#locale#</I></B><br>
    #LSDateFormat(Now(), "mmm-dd-yyyy")#<br>
    #LSDateFormat(Now(), "mmmm d, yyyy")#<br>
    #LSDateFormat(Now(), "mm/dd/yyyy")#<br>
    #LSDateFormat(Now(), "d-mmm-yyyy")#<br>
    #LSDateFormat(Now(), "ddd, mmmm dd, yyyy")#<br>
    #LSDateFormat(Now(), "d/m/yy")#<br>
    #LSDateFormat(Now())#<br>
    <hr noshade>
  </cfoutput>
</cfloop>
```

LSEuroCurrencyFormat

Description: Formats a number in a locale-specific currency format.

Return value: A formatted value.

- If the country of the current locale belongs to the Euro Zone (whose members have converted to the euro) the euro currency symbol displays in the formatted output. If the value is negative, the format includes a negative sign before the value or parentheses around the value, according to the formatting rules of the current locale.
- If the country of the current locale is not in the Euro Zone, the currency symbol of the current locale displays. If the value is negative, the format includes a negative sign before the value or parentheses around the value, according to the formatting rules of the current locale.

Category: Display and formatting functions, International functions

Syntax: `LSEuroCurrencyFormat(currency-number [, type ])`

See also: `LSParseEuroCurrency`, `LSCurrencyFormat`, `SetLocale`

History: New in ColdFusion MX: This function might return different formatting than in earlier releases. This function uses Java standard locale formatting rules on all platforms. This function determines whether the current locale's country belongs to the Euro Zone (whose members have converted to the euro); if so, it displays the value in the locale's euro format. (Earlier releases formatted output with the euro symbol regardless of the locale or Euro Zone membership.)

Parameters:

Parameter	Description
currency-number	Currency value.
type	• local: the currency format used in the locale. (Default.) • international: the international standard currency format of the locale. For example, EUR10.00 • none: the currency format used in the locale; no currency symbol

Usage: This function uses Java standard locale formatting rules on all platforms.

For a list of the locale options that ColdFusion supports, see SetLocale on page 607.

To set the default display format of date, time, number, and currency values, use the `SetLocale` function.

The following example shows differences between earlier ColdFusion releases and ColdFusion MX in accepting input formats and displaying output:

```
<cfset oldlocale = SetLocale("German (Standard)")>
<cfoutput>
   New Locale = #getLocale()#
   <br>
   <cfset z= LSEuroCurrencyFormat(1234.56, "local")>
   <cfset x= LSParseEuroCurrency("EUR1.234,56")>
   <cfset y= LSIsCurrency("DEM1.234,56")>
   <cfset k= LSIsCurrency("EUR1.234,56")>
   <br>
   Euro CurrFormat  : #z# <br>
   Parsed Euro Value: #x#<br>
   IsCurrency(DEM):   #y#<br>
   IsCurrency(EUR):   #k#<br>
</cfoutput>
```

ColdFusion MX produces the following output:

```
New Locale = German (Standard)
Euro CurrFormat : 1.234,56 €
Parsed Euro Value: 1234.56
IsCurrency(DEM): NO
IsCurrency(EUR): YES
```

Earlier ColdFusion releases produce the following output:

```
New Locale = German (Standard)
Euro CurrFormat : 1.234,56 €
Parsed Euro Value: 1234.56
IsCurrency(DEM): YES
IsCurrency(EUR): NO
```

Currency output

The following table shows examples of currency output:

Locale	Type = local	Type = international	Type = none
Dutch (Belgian)	100.000,00	EUR100.000,00	100.000,00
Dutch (Standard)	€100.000,00	EUR100.000,00	100.000,00
English (Australian)	€100,000.00	EUR100,000.00	100,000.00
English (Canadian)	€100,000.00	EUR100,000.00	100,000.00
English (New Zealand)	€100,000.00	EUR100,000.00	100,000.00
English (UK)	€100,000.00	EUR100,000.00	100,000.00
English (US)	€100,000.00	EUR100,000.00	100,000.00
French (Belgian)	100.000,00 €	EUR100.000,00	100.000,00
French (Canadian)	100 000,00 €	EUR100 000,00	100 000,00
French (Standard)	100 000,00 €	EUR100 000,00	100 000,00
French (Swiss)	€100'000.00	EUR100'000.00	100'000.00

Locale	Type = local	Type = international	Type = none
German (Austrian)	€100.000,00	EUR100.000,00	100.000,00
German (Standard)	100.000,00 €	EUR100.000,00	100.000,00
German (Swiss)	€100'000.00	EUR100'000.00	100'000.00
Italian (Standard)	€10.000.000	EUR10.000.000	10.000.000
Italian (Swiss)	€100'000.00	EUR100'000.00	100'000.00
Norwegian (Bokmal)	€100 000,00	EUR100 000,00	100 000,00
Norwegian (Nynorsk)	€100 000,00	EUR100 000,00	100 000,00
Portuguese (Brazilian)	€100.000,00	EUR100.000,00	100.000,00
Portuguese (Standard)	€100.000,00	EUR100.000,00	100.000,00
Spanish (Mexican)	€100,000.00	EUR100,000.00	100,000.00
Spanish (Modern)	10.000.000 €	EUR10.000.000	10.000.000
Spanish (Standard)	10.000.000 €	EUR10.000.000	10.000.000
Swedish	100.000,00 €	EUR100.000,00	100.000,00

Note: ColdFusion uses the Spanish (Standard) formats for Spanish (Modern) and Spanish (Standard).

The following example shows how the function formats negative values. The format includes a negative sign before the value, or parentheses around the value, according to the formatting rules of the current locale.

Input value	Output if locale = French (Standard)	Output if locale = English (US)
-1234.56	-1 234,56 €	($1,234.56)

Example

```
<h3>LSEuroCurrencyFormat Example</h3>
<p>LSEuroCurrencyFormat returns a currency value using the locale
         convention. Default value is "local."
<!--- loop through list of locales,show currency values for 100,000 units --->
<cfloop list = "#Server.Coldfusion.SupportedLocales#"
index = "locale" delimiters = ",">
   <cfset oldlocale = SetLocale(locale)>
   <cfoutput><p><B><I>#locale#</I></B><br>
      Local: #LSEuroCurrencyFormat(100000, "local")#<br>
      International: #LSEuroCurrencyFormat(100000, "international")#<br>
      None: #LSEuroCurrencyFormat(100000, "none")#<br>
      <Hr noshade>
   </cfoutput>
</cfloop>
```

LSIsCurrency

Description: Determines whether a string is formatted as a locale-specific currency string.

Return value: True, if the current locale setting is that of a Euro Zone country, and the string is valid in the Euro Zone or non-Euro zone currency format; False, otherwise.

Category: Display and formatting functions, Decision functions, International functions

Syntax: `LSIsCurrency(string)`

See also: `GetLocale`, `SetLocale`

History: New in ColdFusion MX: This function might return a different result than in earlier releases. This function uses Java standard locale formatting rules on all platforms. This function determines whether the current locale's country belongs to the Euro Zone, whose members have converted to the euro; if so, it displays the currency value in euros.

Parameters

Parameter	Description
string	A currency string or a variable that contains one.

Usage: For examples of ColdFusion code and output that shows differences between earlier ColdFusion releases and ColdFusion MX in accepting input formats and displaying output, see LSEuroCurrencyFormat on page 542.

To set the default display format of date, time, number, and currency values, use the `SetLocale` function.

Example:

```
<h3>LSIsCurrency Example</h3>

<cfif IsDefined("FORM.locale")>
<!--- if locale is defined, set locale to that entry --->
<cfset NewLocale = SetLocale(FORM.locale)>

<p>Is the value "<cfoutput>#FORM.myValue#</cfoutput>"
         a proper currency value for <cfoutput>#GetLocale()#</cfoutput>?
<p>Answer: <cfoutput>#LSIsCurrency(FORM.myValue)#</cfoutput>
</cfif>

<p><form action = "LSIsCurrency.cfm">
<p>Select a locale for which you would like to check a currency value:
<!--- check the current locale for server --->
<cfset serverLocale = GetLocale()>
```

LSIsDate

Description: Determines whether a string is a date/time value that can be formatted in a locale-specific format.

Return value: True, if the string can be formatted as a date/time value in the current locale; False, otherwise.

Category: Date and time functions, Display and formatting functions, International functions

Syntax: `LSIsDate(string)`

See also: `CreateDateTime`, `GetLocale`, `IsNumericDate`, `LSDateFormat`, `ParseDateTime`, `SetLocale`

History: New in ColdFusion MX:

- This function might return a different result than in earlier releases. This function uses Java standard locale formatting rules on all platforms.
- This function accepts a dash or hyphen character only in the Dutch(Standard) and Portuguese (Standard) locales. If called this way (for example, LsIsDate("3-1-2002") in any other locale, this function returns False. (Earlier releases returned True.)
- When using the SUN JRE 1.3.1 on an English(UK) locale, this function returns False for a date that has a one-digit month or day (for example, 1/1/01). To work around this, insert a zero in a one-digit month or day (for example, 01/01/01).

Parameters

Parameter	Description
string	A string or a variable that contains one

Usage: A date/time object is in the range 100 AD–9999 AD. See "How ColdFusion processes two-digit year values" on page 377.

To set the default display format of date, time, number, and currency values, use the `SetLocale` function.

Example:

```
<h3>LSIsDate Example</h3>
<cfif IsDefined("FORM.locale")>
   <!--- if locale is defined, set locale to that entry --->
   <cfset NewLocale = SetLocale(FORM.locale)>
   <p>Is the value "<cfoutput>#FORM.myValue#</cFOUTPUT>" a proper date
         value for <cfoutput>#GetLocale()#</cfoutput>?
   <p>Answer: <cfoutput>#LSIsDate(FORM.myValue)#</cfoutput>
</cfif>
<p><form action = "LSIsDate.cfm">
<p>Select a locale for which you would like to check a date value:
<!--- check the current locale for server --->
<cfset serverLocale = GetLocale()>
```

LSIsNumeric

Description: Determines whether a string can be formatted in a locale-specific format.

Return value: True, if the string can be formatted in the current locale format; False, otherwise.

Category: Decision functions, International functions, String functions

Syntax: `LSIsNumeric(string)`

See also: `GetLocale`, `SetLocale`

History: New in ColdFusion MX: This function might return a different result than in earlier releases. This function uses Java standard locale formatting rules on all platforms.

Parameters

Parameter	Description
string	A string or a variable that contains one

Usage: To set the default display format of date, time, number, and currency values, use the `SetLocale` function.

Example:

```
<h3>LSIsNumeric Example</h3>

<cfif IsDefined("FORM.locale")>
<!--- if locale is defined, set locale to that entry --->
<cfset NewLocale = SetLocale(FORM.locale)>

<p>Is the value "<cfoutput>#FORM.myValue#</cFOUTPUT>"
a proper numeric value for <cfoutput>#GetLocale()#</cfoutput>?

<p>Answer: <cfoutput>#LSIsNumeric(FORM.myValue)#</cfoutput>
</cfif>

<p><form action = "LSIsNumeric.cfm">

<p>Select a locale for which to check a numeric value:
...
```

LSNumberFormat

Description: Formats a number in a locale-specific format.

Return value: A formatted number.

- If no mask is specified, it returns the number formatted as an integer
- If no mask is specified, truncates the decimal part; for example, it truncates 34.57 to 35
- If the specified mask cannot correctly mask a number, it returns the number unchanged
- If the parameter value is "" (an empty string), it returns 0.

Category: Display and formatting functions, International functions

Syntax: `LSNumberFormat(number [, mask ])`

See also: `GetLocale`, `SetLocale`

History: New in ColdFusion MX: If the specified mask format cannot correctly mask a number, this function returns the number unchanged. (In earlier releases, it truncated the number or threw an error.) (If no mask is specified, ColdFusion MX truncates the decimal part as ColdFusion 5 does. For example, it truncates 1234.567 to 1235.)

New in ColdFusion MX: This function might return different formatting than in earlier releases. This function uses Java standard locale formatting rules on all platforms.

Parameters

Parameter	Description
number	Number to format
mask	LSNumberFormat mask characters apply, except: dollar sign, comma, and dot are mapped to their locale-specific equivalents.

The following table lists the `LSNumberFormat` mask characters.

Character	Meaning
_	(Underscore.) Digit placeholder.
9	Digit placeholder. (Shows decimal places more clearly than _ .)
.	Location of a mandatory decimal point (or locale-appropriate symbol).
0	Located to the left or right of a mandatory decimal point. Pads with zeros.
( )	If number is less than zero, puts parentheses around the mask.
+	Puts plus sign before positive number; minus sign before negative number.
-	Puts space before positive number; minus sign before negative number.
,	Separates every third decimal place with a comma (or locale-appropriate symbol).
L,C	Left-justifies or center-justifies number within width of mask column. First character of mask must be L or C. Default: right-justified.

Character	Meaning
$	Puts a dollar sign (or locale-appropriate symbol) before formatted number. First character of mask must be the dollar sign ($).
^	Separates left and right formatting.

Note: If you do not specify a sign for the mask, positive and negative numbers do not align in columns. To put a plus sign or space before positive numbers and a minus sign before negative numbers, use the plus or hyphen mask character, respectively.

Usage

This function uses Java standard locale formatting rules on all platforms.

The position of symbols in format masks determines where the codes take effect. For example, if you put a dollar sign at the far left of a format mask, ColdFusion displays a dollar sign at the left edge of the formatted number. If you separate the dollar sign on the left edge of the format mask by at least one underscore, ColdFusion displays the dollar sign just to the left of the digits in the formatted number.

These examples show how symbols determine formats:

Number	Mask	Result
4.37	$____.__	"$ 4.37"
4.37	_$___.__	" $4.37"

The positioning can also show where to put a minus sign for negative numbers:

Number	Mask	Result
-4.37	-____.__	"- 4.37"
-4.37	_-___.__	" -4.37"

The positions for a symbol are: far left, near left, near right, and far right. The left and right positions are determined by the side of the decimal point on which the code character is shown. For formats that do not have a fixed number of decimal places, you can use a caret (^) to separate the left fields from the right.

An underscore determines whether the code is placed in the far or near position. Most code characters' effect is determined by the field in which they are located. This example shows how to specify where to put parentheses to display negative numbers:

Number	Mask	Result
3.21	C(__^__)	"(3.21)"
3.21	C__(^__)	" (3.21)"
3.21	C(__^)__	"(3.21) "
3.21	C__(^)__	" (3.21) "

To set the default display format of date, time, number, and currency values, use the SetLocale function.

When converting from string to double, to prevent rounding errors, this function adds a rounding factor of 1.5543122344752E-014 to the converted number. For example, without adding the rounding factor, converting the string value 1.275 to double with two digits of precision results in a value of 1.27499999999999999, which would be rounded up to 1.27. By adding the rounding factor, the conversion correctly results in a value of 1.28.

If you round off a double, such as 1.99499999999999999999999999999, where the last decimal is 10E-14, the rounding factor can cause an incorrect result.

Example

```
<h3>LSNumberFormat Example</h3>
<p>LSNumberFormat returns a number value using the locale convention.
<!--- loop through a list of locales and show number values --->
<cfloop LIST = "#Server.Coldfusion.SupportedLocales#"
index = "locale" delimiters = ",">
  <cfset oldlocale = SetLocale(locale)>
  <cfoutput><p><b><i>#locale#</i></b><br>
    #LSNumberFormat(-1234.5678, "_________")#<br>
    #LSNumberFormat(-1234.5678, "_________.___")#<br>
    #LSNumberFormat(1234.5678, "_________")#<br>
    #LSNumberFormat(1234.5678, "_________.___")#<br>
    #LSNumberFormat(1234.5678, "$_(_________.___)")#<br>
    #LSNumberFormat(-1234.5678, "$_(_________.___)")#<br>
    #LSNumberFormat(1234.5678, "+_________.___")#<br>
    #LSNumberFormat(1234.5678, "-_________.___")#<br>
  </cfoutput>
</cfloop>
```

LSParseCurrency

Description: Formats a locale-specific currency string as a number. Attempts conversion through each of the default currency formats (none, local, international).

Return value: A formatted number that matches the value of the string.

Category: International functions, String functions

Syntax: `LSParseCurrency(string)`

See also: `LSCurrencyFormat`, `LSParseEuroCurrency`

History: New in ColdFusion MX: This function might return different formatting than in earlier releases. This function uses Java standard locale formatting rules on all platforms.

Parameters

Parameter	Description
string	A locale-specific string a variable that contains one

Usage: This function uses Java standard locale formatting rules on all platforms.

This function is similar to `LSParseEuroCurrency`, but it parses a non-euro currency string for a locale.

Currency output

The following table shows examples of currency output:

Locale	Format = local	Format = international	Format = none
Dutch (Belgian)	100.000,00 BF	BEF100.000,00	100.000,00
Dutch (Standard)	fl 100.000,00	NLG100.000,00	100.000,00
English (Australian)	$100,000.00	AUD100,000.00	100,000.00
English (Canadian)	$100,000.00	CAD100,000.00	100,000.00
English (New Zealand)	$100,000.00	NZD100,000.00	100,000.00
English (UK)	£100,000.00	GBP100,000.00	100,000.00
English (US)	$100,000.00	USD100,000.00	100,000.00
French (Belgian)	100.000,00 FB	BEF100.000,00	100.000,00
French (Canadian)	100 000,00 $	CAD100 000,00	100 000,00
French (Standard)	100 000,00 F	FRF100 000,00	100 000,00
French (Swiss)	SFr. 100'000.00	CHF100'000.00	100'000.00
German (Austrian)	öS 100.000,00	ATS100.000,00	100.000,00
German (Standard)	100.000,00 DM	DEM100.000,00	100.000,00
German (Swiss)	SFr. 100'000.00	CHF100'000.00	100'000.00

Locale	Format = local	Format = international	Format = none
Italian (Standard)	L. 10.000.000	ITL10.000.000	10.000.000
Italian (Swiss)	SFr. 100'000.00	CHF100'000.00	100'000.00
Norwegian (Bokmal)	kr 100 000,00	NOK100 000,00	100 000,00
Norwegian (Nynorsk)	kr 100 000,00	NOK100 000,00	100 000,00
Portuguese (Brazilian)	R$100.000,00	BRC100.000,00	100.000,00
Portuguese (Standard)	R$100.000,00	BRC100.000,00	100.000,00
Spanish (Mexican)	$100,000.00	MXN100,000.00	100,000.00
Spanish (Modern)	10.000.000 Pts	ESP10.000.000	10.000.000
Spanish (Standard)	10.000.000 Pts	ESP10.000.000	10.000.000
Swedish	100.000,00 kr	SEK100.000,00	100.000,00

To set the default display format of date, time, number, and currency values, use the SetLocale function.

Example

```
<h3>LSParseCurrency Example</h3>
<p>LSParseCurrency coverts a locale-specific currency string to a number.
         Attempts conversion through each of the three default currency formats.
<!--- loop through a list of locales; show currency values for 123,456 units --->
<cfloop LIST = "#Server.Coldfusion.SupportedLocales#"
index = "locale" delimiters = ",">
   <cfset oldlocale = SetLocale(locale)>
   <cfoutput><p><B><I>#locale#</I></B><br>
      Local: #LSCurrencyFormat(123456, "local")#<br>
      Currency: #LSParseCurrency(LSCurrencyFormat(123456,"local"))#<br>
      International: #LSCurrencyFormat(123456, "international")#<br>
      None: #LSCurrencyFormat(123456, "none")#<br>
      <hr noshade>
   </cfoutput>
</cfloop>
```

LSParseDateTime

Description: Formats a date/time string in a locale-specific format. Does not parse POP date/time objects.

Return value: A formatted date/time value.

Category: Date and time functions, Display and formatting functions, International functions, String functions

Syntax: `LSParseDateTime(date/time-string)`

See also: `LSDateFormat`, `ParseDateTime`, `SetLocale`

History: New in ColdFusion MX: This function might return different formatting than in earlier releases. This function uses Java standard locale formatting rules on all platforms.

New in ColdFusion MX: ColdFusion processes the `date/time string` parameter value differently than in earlier releases:

- If a time zone specified in the `date/time string` parameter is different from the time zone setting of the ColdFusion Server computer, ColdFusion adjusts the time value to its equivalent in the time zone of the ColdFusion Server computer.
- If a time zone is not specified in the `date/time string` parameter, ColdFusion does not adjust the time value.

Macromedia recommends that, when developing an application, you evaluate the entered time string for a time zone setting; ensure that your application adjusts the value, if necessary. (To do this, you can use a `cfcatch` tag.)

Parameters

Parameter	Description
date/time-string	A string a variable that contains one, in a format that is readable in the current locale. Default: English (US) locale.

Usage: This function uses Java standard locale formatting rules on all platforms.

ColdFusion uses the following date/time string length formats:

- Short: all numeric. For example: "12/25/01", "3:30pm"
- Medium: abbreviated month name. For example: "Jan 12, 1952"
- Long: month spelled out. For example: "January 12, 1952", "3:30:32pm"
- Full: Month and day spelled out. For example: "Tuesday, April 12, 1952 AD", "3:30:42pm PST"

You can pass a date/time value to this function that is in the English (US) locale, in any of the following date or date/time formats:

Mask	Format	Example
Jan dd, yyyy	Date, Default	Jan 30, 2002
m/dd/yy	Date, Short	1/30/02
m dd, yyyy	Date, Medium	Jan 30, 2002
mmmm dd, yyyy	Date, Long	January 30, 2002
Wednesday, mmmm dd, yyyy	Date, Full	Wednesday, January 30, 2002
Jan dd, yyyy 7:mm:ss AM	Date/time, Default	Jan 30, 2002 7:02:12 AM
m/dd/yy 7:mm AM	Date/time, Short	1/30/02 7:02 AM
Jan dd, yyyy 7:mm:ss AM	Date/time, Medium	Jan 30, 2002 7:02:12 AM
mmmm dd, yyyy 7:mm:ss AM PST	Date/time, Long	January 30, 2002 7:02:12 AM PST
Wednesday, mmmm dd, yyyy 7:mm:ss AM PST	Date/time, Full	Wednesday, January 30, 2002 7:02:12 AM PST
{ts 'yyyy-07-06 hh:mm:ss'}	Date/time, None	{ts '2003-07-06 00:00:00'}

A date/time object is in the range 100 AD–9999 AD. See "How ColdFusion processes two-digit year values" on page 377.

To set the default display format of date, time, number, and currency values, use the `SetLocale` function.

Note: This function does not accept POP dates.

Example

```
<h3>LSParseDateTime Example - returns a locale-specific date/time object</h3>
<!--- loop through a list of locales and show date values for Now()--->
<cfloop LIST = "#Server.Coldfusion.SupportedLocales#"
index = "locale" delimiters = ",">
   <cfset oldlocale = SetLocale(locale)>
   <cfoutput><p><B><I>#locale#</I></B><br>
      <p>Locale-specific formats:
      <br>#LSDateFormat(Now(), "mmm-dd-yyyy")# #LSTimeFormat(Now())#<br>
      #LSDateFormat(Now(), "mmmm d, yyyy")# #LSTimeFormat(Now())#<br>
      #LSDateFormat(Now(), "mm/dd/yyyy")# #LSTimeFormat(Now())#<br>
      #LSDateFormat(Now(), "d-mmm-yyyy")# #LSTimeFormat(Now())#<br>
      #LSDateFormat(Now(), "ddd, mmmm dd, yyyy")# #LSTimeFormat(Now())#<br>
      #LSDateFormat(Now(), "d/m/yy")# #LSTimeFormat(Now())#<br>
      #LSDateFormat(Now())# #LSTimeFormat(Now())#<br>
   <p>Standard Date/Time:
   #LSParseDateTime("#LSDateFormat(Now())# #LSTimeFormat(Now())#")#<br>
   </cfoutput>
</cfloop>
```

LSParseEuroCurrency

Description: Formats a locale-specific currency string that contains the euro symbol (€) or sign (EUR) as a number. Attempts conversion through each of the default currency formats (none, local, international).

Return value: A formatted number that matches the value of the string.

Category: International functions, String functions

Syntax: `LSParseEuroCurrency(currency-string)`

See also: `LSParseCurrency`, `LSEuroCurrencyFormat`, `SetLocale`

History: New in ColdFusion MX: This function might return different formatting than in earlier releases. This function uses Java standard locale formatting rules on all platforms. This function determines whether the current locale's country belongs to the Euro Zone, whose members have converted to the euro; if so, it displays the currency value in euros.

Parameters

Parameter	Description
currency-string	Locale-specific string or a variable that contains one.

Usage: This function uses Java standard locale formatting rules on all platforms.

For example ColdFusion code and output that shows differences between earlier ColdFusion releases and ColdFusion MX in accepting input formats and displaying output, see LSEuroCurrencyFormat on page 542.

This function is similar to `LSParseCurrency`, but it parses only euro currency.

For a list of the locale options that ColdFusion supports, see SetLocale on page 607.

To set the default display format of date, time, number, and currency values, use the `SetLocale` function.

Example:

```
<h3>LSParseEuroCurrency Example</h3>
<p>LSParseEuroCurrency converts locale-specific currency string to
        number. Attempts conversion through each default currency format.
<!--- Loop through a list of locales. Show currency values for 123,456 units. --->
<cfloop LIST = "#Server.Coldfusion.SupportedLocales#"
index = "locale" delimiters = ",">
   <cfset oldlocale = SetLocale("#locale#")>
   <cfoutput><p><B><I>#locale#</I></B><br>
      Local: #LSEuroCurrencyFormat(123456, "local")#<br>
      Currency Number:
      #LSParseEuroCurrency("EUR123456")#<br>
      International: #LSEuroCurrencyFormat(123456, "international")#<br>
      None: #LSEuroCurrencyFormat(123456, "none")#<br>
      <hr noshade>
   </cfoutput>
</cfloop>
```

LSParseNumber

Description: Formats a locale-specific string as a number.

Return value: A number that matches the value of the string.

Category: International functions, String functions

Syntax: `LSParseNumber(string)`

See also: `LSParseDateTime`, `SetLocale`

History: New in ColdFusion MX: This function might return different formatting than in earlier releases. This function uses Java standard locale formatting rules on all platforms.

Parameters

Parameter	Description
string	A string or a variable that contains one

Usage: This function uses Java standard locale formatting rules on all platforms.

To set the default display format of date, time, number, and currency values, use the `SetLocale` function.

Example

```
<h3>LSParseNumber Example</h3>
<p>LSParseNumber converts a locale-specific string to a number.
         Returns the number matching the value of string.
<!--- loop through a list of locales and show number values --->
<cfloop LIST = "#Server.Coldfusion.SupportedLocales#"
index = "locale" delimiters = ",">
  <cfset oldlocale = SetLocale(locale)>

  <cfoutput><p><B><I>#locale#</I></B><br>
    #LSNumberFormat(-1234.5678, "_________")#<br>
    #LSNumberFormat(-1234.5678, "_________.___")#<br>
    #LSNumberFormat(1234.5678, "_________")#<br>
    #LSNumberFormat(1234.5678, "_________.___")#<br>
    #LSNumberFormat(1234.5678, "$_(_________.___)")#<br>
    #LSNumberFormat(-1234.5678, "$_(_________.___)")#<br>
    #LSNumberFormat(1234.5678, "+_________.___")#<br>
    #LSNumberFormat(1234.5678, "-_________.___")#<br>
    The actual number:
      #LSParseNumber(LSNumberFormat(1234.5678, "_________"))#<br>
    <hr noshade>
  </cfoutput>
</cfloop>
```

LSTimeFormat

Description: Formats the time part of a date/time string in a locale-specific format.

Return value: A formatted time value.

Category: Date and time functions, Display and formatting functions, International functions

Syntax: `LSTimeFormat(time [, mask ])`

See also: `LSParseDateTime`

History: New in ColdFusion MX:

- This function might return different formatting than in earlier releases. This function uses Java standard locale formatting rules on all platforms.
- This function supports the `short`, `medium`, `long`, and `full mask` attribute options.

Parameters

Parameter	Description
string	• A date/time value • A string that is convertible to a time value A date/time object is in the range 100 AD–9999 AD. See "How ColdFusion processes two-digit year values" on page 377.
mask	Masking characters that determine the format: • h: Hours; no leading zero for single-digit hours (12-hour clock) • hh: Hours; leading zero for single-digit hours. (12-hour clock) • H: Hours; no leading zero for single-digit hours (24-hour clock) • HH: Hours; leading zero for single-digit hours (24-hour clock) • m: Minutes; no leading zero for single-digit minutes • mm: Minutes; leading zero for single-digit minutes • s: Seconds; no leading zero for single-digit seconds • ss: Seconds; leading zero for single-digit seconds • t: One-character time marker string, such as A or P. • tt: Multiple-character time marker string, such as AM or PM • short • medium • long • full

Usage: This function uses Java standard locale formatting rules on all platforms.

When passing date/time value as a string, enclose it in quotation marks. Otherwise, it is interpreted as a number representation of a date/time object.

To calculate a difference between time zones, use the `GetTimeZoneInfo` function.

To set the default display format of date, time, number, and currency values, use the `SetLocale` function.

If no seconds value is passed to this function, and the mask value is `s`, the default output seconds format is one zero; for example, `lstimeformat(6:39, "h:m:s")` returns `6:39:0`. If the mask value is `ss`, it returns `6:39:00`.

Example

```
<h3>LSTimeFormat Example</h3>

<p>LSTimeFormat returns a time value using the locale convention.

<!--- loop through a list of locales and show time values --->
<cfloop LIST = "#Server.Coldfusion.SupportedLocales#"
index = "locale" delimiters = ",">
   <cfset oldlocale = SetLocale(locale)>

   <cfoutput><p><B><I>#locale#</I></B><br>
   #LSTimeFormat(Now())#<br>
   #LSTimeFormat(Now(), 'hh:mm:ss')#<br>
   #LSTimeFormat(Now(), 'hh:mm:sst')#<br>
   #LSTimeFormat(Now(), 'hh:mm:sstt')#<br>
   #LSTimeFormat(Now(), 'HH:mm:ss')#<br>
      <hr noshade>
   </cfoutput>

</cfloop>
```

LTrim

Description Removes leading spaces from a string.

Return value A copy of the string, without leading spaces.

Category Display and formatting functions, String functions

Syntax **LTrim**(*string*)

See also RTrim, ToBase64

Parameters

Parameter	Description
string	A string or a variable that contains one

Example

```
<h3>LTrim Example</h3>

<cfif IsDefined("FORM.myText")>
<cfoutput>
<pre>
Your string:"#FORM.myText#"
Your string:"#Ltrim(FORM.myText)#"
(left trimmed)
</pre>
</cfoutput>
</cfif>

<form action = "ltrim.cfm">
<p>Type in some text, and it will be modified by Ltrim to remove
          leading spaces from the left
<p><input type = "Text" name = "myText" value = "   TEST">

<p><input type = "Submit" name = "">
</form>
```

Max

Description Determines the greater of two numbers.

Return value The greater of two numbers.

Category Mathematical functions

Syntax `Max(number1, number2)`

See also `Min`

Parameters

Parameter	Description
number1, number2	Numbers

Example

```
<h3>Max Example</h3>
<cfif IsDefined("FORM.myNum1")>
   <cfif IsNumeric(FORM.myNum1) and IsNumeric(FORM.myNum2)>
   <p>The maximum of the two numbers is <cfoutput>#Max(FORM.myNum1,
    FORM.myNum2)#</cfoutput>
   <p>The minimum of the two numbers is <cfoutput>#Min(FORM.myNum1,
    FORM.myNum2)#</cfoutput>
   <cfelse>
   <p>Please enter two numbers
   </cfif>
</cfif>

<form action = "max.cfm">
<h3>Enter two numbers, see the maximum and minimum of them</h3>

Number 1 <input type = "Text" name = "MyNum1">
<br>Number 2 <input type = "Text" name = "MyNum2">

<br><input type = "Submit" name = "" value = "See results">
</form>
```

Mid

Description Extracts a substring from a string.

Return value A string; the set of characters from *string*, beginning at *start*, of length *count*.

Category String functions

Syntax `Mid(string, start, count)`

See also `Left`, `Len`, `Right`

Parameters

Parameter	Description
string	A string or a variable that contains one. Must be single-quote or double-quote delimited.
start	A positive integer or a variable that contains one. Position at which to start count.
count	A positive integer or a variable that contains one. Number of characters to return. (0 is not valid, but it does not throw an error.)

Example

```
<h3>Mid Example</h3>
<cfif IsDefined("Form.MyText")>
<!--- if len is 0, then err --->
  <cfif Len(FORM.myText) is not 0>
    <cfif Len(FORM.myText) LTE FORM.RemoveChars>
    <p>Your string <cfoutput>#FORM.myText#</cFOUTPUT> only has
        <cfoutput>#Len(FORM.myText)#</cfoutput> characters. You cannot output
        the <cfoutput>#FORM.removeChars# </cfoutput> middle characters of this
        string because it is not long enough
    <cfelse>

    <p>Your original string: <cfoutput>#FORM.myText#</cfoutput>
    <p>Your changed string, showing only the <cfoutput>#FORM.removeChars#
        </cfoutput> middle characters:
    <cfoutput>#Mid(Form.myText, FORM.removeChars, Form.countChars)#</cfoutput>
    </cfif>
```

Min

Description Determines the lesser of two numbers.

Return value The lesser of two numbers.

Category Mathematical functions

Syntax `Min(number1, number2)`

See also `Max`

Parameters

Parameter	Description
number1, number2	Numbers

Example

```
<h3>Min Example</h3>
<cfif IsDefined("FORM.myNum1")>
   <cfif IsNumeric(FORM.myNum1) and IsNumeric(FORM.myNum2)>
   <p>The maximum of the two numbers is <cfoutput>#Max(FORM.myNum1,
    FORM.myNum2)#</cfoutput>
   <p>The minimum of the two numbers is <cfoutput>#Min(FORM.myNum1,
    FORM.myNum2)#</cfoutput>
   <cfelse>
   <p>Please enter two numbers
   </cfif>
</cfif>

<form action = "min.cfm">
<h3>Enter two numbers, and see the maximum and minimum of the two numbers</h3>

Number 1 <input type = "Text" name = "MyNum1">
<br>Number 2 <input type = "Text" name = "MyNum2">

<br><input type = "Submit" name = "" value = "See results">
</form>
```

Minute

Description Extracts the minute value from a date/time object.

Return value The ordinal value of the minute, in the range 0–59.

Category Date and time functions

Syntax `Minute(date)`

See also `DatePart`, `Hash`, `Second`

Parameters

Parameter	Description
date	• A date/time object, in the range 100 AD–9999 AD. See "How ColdFusion processes two-digit year values" on page 377.

Usage When passing a date/time value as a string, you must enclose it in quotation marks. Otherwise, it is interpreted as a number representation of a date/time object.

Example

```
<h3>Minute Example</h3>

<cfoutput>
The time is currently #TimeFormat(Now())#.
We are in hour #Hour(Now())#, Minute #Minute(Now())#
and Second #Second(Now())# of the day.
</cfoutput>
```

Month

Description: Extracts the month value from a date/time object.

Return value: The ordinal value of the month, in the range 1 (January) – 12 (December).

Category: Date and time functions

Syntax: `Month(date)`

See also: `DatePart`, `MonthAsString`, `Quarter`

Parameters:

Parameter	Description
date	Date/time object, in the range 100 AD–9999 AD. See "How ColdFusion processes two-digit year values" on page 377.

Usage: When passing a date/time value as a string, you must enclose it in quotation marks. Otherwise, it is interpreted as a number representation of a date/time object.

Note: You can pass the `CreateDate` function or the `Now` function as the `date` parameter of this function; for example: `#Month(CreateDate(2001, 3, 3))#`

Example:

```
<h3>Month Example</h3>
<cfif IsDefined("FORM.year")>
More information about your date:
<cfset yourDate = CreateDate(FORM.year, FORM.month, FORM.day)>
<cfoutput>
   <p>Your date, #DateFormat(yourDate)#.
   <br>It is #DayofWeekAsString(DayOfWeek(yourDate))#, day
    #DayOfWeek(yourDate)# in the week.
   <br>This is day #Day(yourDate)# in the month of
    #MonthAsString(Month(yourDate))#, which has
    #DaysInMonth(yourDate)# days.
   <br>We are in week #Week(yourDate)# of #Year(yourDate)#
    (day #DayofYear(yourDate)# of #DaysinYear(yourDate)#). <br>
   <cfif IsLeapYear(Year(yourDat"))>This is a leap year
      <cfelse>This is not a leap year
   </cfif>
</cfoutput>
</cfif>
```

MonthAsString

Description: Determines the name of the month that corresponds to *month_number*.

Return value: A string; the name of a month.

Category: Date and time functions, String functions

Syntax: `MonthAsString(month_number)`

See also: DatePart, Month, Quarter

Parameters

Parameter	Description
month_number	An integer in the range 1 – 12.

Example

```
<h3>MonthAsString Example</h3>

<cfif IsDefined("FORM.year")>
<p>More information about your date:
<cfset yourDate = CreateDate(FORM.year, FORM.month, FORM.day)>

<cfoutput>
<p>Your date, #DateFormat(yourDate)#.
<br>It is #DayofWeekAsString(DayOfWeek(yourDate))#, day
 #DayOfWeek(yourDate)# in the week.
<br>This is day #Day(YourDate)# in the month of
 #MonthAsString(Month(yourDate))#, which has
  #DaysInMonth(yourDate)# days.
<br>We are in week #Week(yourDate)# of #Year(yourDate)#
   (day #DayofYear(yourDate)# of #DaysinYear(yourDate)#). <br>
   <cfif IsLeapYear(Year(yourDate))>This is a leap year
      <cfelse>This is not a leap year
   </cfif>
</cfoutput>
</cfif>
```

Now

Description — Gets the current date and time of the computer running the ColdFusion server. The return value can be passed as a parameter to date functions such as `DaysInYear` or `FirstDayOfMonth`.

Return value — A date/time object; the current date and time of the computer running the ColdFusion server.

Category — Date and time functions

Syntax — `Now()`

See also — `CreateDateTime`, `DatePart`

Example —

```
<h3>Now Example</h3>
<p>Now returns the current date and time as a valid date/time object.

<p>The current date/time value is <cfoutput>#Now()#</cfoutput>
<p>You can also represent this as <cfoutput>#DateFormat(Now())#,
 #TimeFormat(Now())#</cfoutput>
```

NumberFormat

Description: Creates a custom-formatted number value. Supports the numeric formatting used in the U.S. For international number formatting, see `LSNumberFormat`.

Return value: A formatted number value.

- If no mask is specified, returns the value as an integer with a thousands separator
- If the parameter value is "" (an empty string), it returns 0.

Category: Display and formatting functions

Syntax: `NumberFormat(number [, mask ])`

See also: `DecimalFormat`, `DollarFormat`, `IsNumeric`, `LSNumberFormat`

History: New in ColdFusion MX: If the mask format cannot correctly mask a number, this function returns the number unchanged. (It does not truncate the number nor throw an error.) (If no mask is selected, ColdFusion MX truncates the decimal part as ColdFusion 5 does. For example, it truncates 34.567 to 35.)

Parameters

Parameter	Description
number	A number.
mask	A string or a variable that contains one. Set of characters that determine how ColdFusion displays the number

The following table explains mask characters:

Mask character	Meaning
_ (underscore)	Optional. Digit placeholder.
9	Optional. Digit placeholder. (Shows decimal places more clearly than _.)
.	Location of a mandatory decimal point.
0	Located to the left or right of a mandatory decimal point. Pads with zeros.
( )	If number is less than zero, puts parentheses around the mask.
+	Puts plus sign before positive number; minus sign before negative number.
-	Puts a space before positive number; minus sign before negative number.
,	Separates every third decimal place with a comma.
L,C	Left-justifies or center-justifies number within width of mask column. First character of mask must be L or C. Default: right-justified.
$	Puts a dollar sign before formatted number. First character of mask must be the dollar sign ($).
^	Separates left and right formatting.

Note: If you do not specify a sign for the mask, positive and negative numbers do not align in columns. To put a plus sign or space before positive numbers and a minus sign before negative numbers, use the plus or minus sign, respectively.

Usage This function uses Java standard locale formatting rules on all platforms.

The position of symbols in format masks determines where the codes take effect. For example, if you put a dollar sign at the far left of a format mask, ColdFusion displays a dollar sign at the left edge of the formatted number. If you separate the dollar sign on the left edge of the format mask by at least one underscore, ColdFusion displays the dollar sign just to the left of the digits in the formatted number.

These examples show how symbols determine formats:

Number	Mask	Result
4.37	$____.__	"$ 4.37"
4.37	_$___.__	" $4.37"

The positioning can also show where to place the minus sign for negative numbers:

Number	Mask	Result
-4.37	-____.__	"- 4.37"
-4.37	_-___.__	" -4.37"

The positions for a symbol are: far left, near left, near right, and far right. The left and right positions are determined by the side of the decimal point on which the code character is shown. For formats that do not have a fixed number of decimal places, you can use a caret (^) to separate the left fields from the right.

An underscore determines whether the code is placed in the far or near position. Most code characters' effect is determined by the field in which they are located. This example shows how to specify where to put parentheses to display negative numbers:

Number	Mask	Result
3.21	C(__^__)	"(3.21)"
3.21	C__(^__)	" (3.21)"
3.21	C(__^)__	"(3.21) "
3.21	C__(^)__	" (3.21) "

When converting from string to double, to prevent rounding errors, this function adds a rounding factor of 1.5543122344752E-014 to the converted number. For example, without adding the rounding factor, converting the string value 1.275 to double with two digits of precision results in a value of 1.27499999999999999, which would be rounded up to 1.27. By adding the rounding factor, the conversion correctly results in a value of 1.28.

If you round off a double such as 1.99499999999999999999999999999, where the last decimal is 10E-14, the rounding factor can cause an incorrect result.

To set the default display format of date, time, number, and currency values, use the `SetLocale` function.

Example

```
<h3>NumberFormat Example</h3>

<cfloop FROM = 1000 TO = 1020 INDEX = "counter">
<cfset CounterRoot2 = Evaluate(counter * sqr(2))>

<!--- Show result in default format, adding comma for thousands place;
         and in custom format, displaying to two decimal places --->
<cfoutput>
<pre>#counter# * Square Root of 2: #NumberFormat(CounterRoot2,
  '_____.__')#</pre>
<pre>#counter# * Square Root of 2: #NumberFormat(CounterRoot2)#</pre>
</cfoutput>
</cfloop>
```

ParagraphFormat

Description
Replaces characters in a string:
- Single newline characters (CR/LF sequences) with spaces
- Double newline characters with HTML paragraph tags (<p>)

Return value
A copy of the string, with characters converted.

Category
Display and formatting functions, String functions

Syntax
`ParagraphFormat(string)`

See also
`StripCR`

Parameters

Parameter	Description
string	A string or a variable that contains one

Usage
This function is useful for displaying data entered in `textarea` fields.

Example
```
<h3>ParagraphFormat Example</h3>
<p>Enter text into this textarea, and see it returned as HTML.
<cfif IsDefined("FORM.myTextArea")>
   <p>Your text area, formatted
   <p><cfoutput>#ParagraphFormat(FORM.myTextArea)#</cfoutput>
</cfif>
<!--- use #Chr(10)##Chr(13)# to simulate a line feed/carriage
return combination; i.e, a return --->
<form action = "paragraphformat.cfm">
<textArea name = "MyTextArea" cols = "35" ROWS = 8>
This is sample text and you see how it scrolls
          <cfoutput>#Chr(10)##Chr(13)#</cfoutput>
          From one line
          <cfoutput>#Chr(10)##Chr(13)##Chr(10)##Chr(13)#</cfoutput>
          to the next
</textArea>
<input type = "Submit" name = "Show me the HTML version">
</form>
```

ParameterExists

Description This function is deprecated. Do not use it in new applications. Use the `IsDefined` function.

Determines whether a parameter exists. ColdFusion does not evaluate the argument.

History New in ColdFusion MX: This function is deprecated. Do not use it in new applications. It might not work, and might cause an error, in later releases.

ParseDateTime

Description: Formats a date/time string according to the English (U.S.) locale convention. (To format a date/time string for other locales, use the `LSParseDateTime` function.)

Return value: An English (U.S.)-formatted date/time value.

Category: Date and time functions, Display and formatting functions

Syntax: `ParseDateTime(date/time-string [, pop-conversion ] )`

See also: `IsDate`, `IsNumericDate`, `SetLocale`

Parameters

Parameter	Description
date/time string	A date/time string or a variable that contains one. A date/time object is in the range 100 AD–9999 AD. See "How ColdFusion processes two-digit year values" on page 377.
pop-conversion	• pop: formats a date/time string to UTC, for the English (US) locale • standard: function does no conversion. See the Usage section.

Usage: This function is similar to `CreateDateTime`, but it takes a string instead of enumerated date/time values. These functions are provided primarily to increase the readability of code in compound expressions.

When passing a date/time value as a string, you must enclose it in quotation marks. Otherwise, it is interpreted as a number representation of a date/time object.

To calculate a difference between time zones, use the `GetTimeZoneInfo` function.

To set the default display format of date, time, number, and currency values, use the `SetLocale` function.

Note: This function does not accept POP dates.

Example:

```
<h3>ParseDateTime Example</h3>
<cfif IsDefined("FORM.theTestValue")>
   <cfif IsDate(FORM.theTestValue)>
   <h3>The expression <cfoutput>#DE(FORM.theTestValue)#</cfoutput> is a valid
          date</h3>
   <p>The date, parsed for use as a date/time value:
    <cfoutput>#ParseDateTime(FORM.theTestValue)#
</cfoutput>
```

Pi

Description: Gets the mathematical constant π, accurate to 15 digits.

Return value: The number 3.14159265358979.

Category: Mathematical functions

Syntax:

```
Pi()
```

See also: ASin, Cos, Sin, Tan

Example:

```
<body>
<h3>Pi Example</h3>

Returns the number
<cfoutput>
#NumberFormat(Pi(), '_.________________')#
</cfoutput>, the mathematical constant pi, accurate to 15 digits.
```

PreserveSingleQuotes

Description: Prevents ColdFusion from automatically escaping single quotation mark characters that are contained in a variable. ColdFusion does not evaluate the argument.

Return value: (None)

Category: Other functions

Syntax: `PreserveSingleQuotes(variable)`

History: New in ColdFusion MX: ColdFusion automatically escapes simple-variable, array-variable, and structure-variable references within a `cfQuery` tag body or block. (Earlier releases did not automatically escape array-variable references.)

Parameters

Parameter	Description
variable	Variable that contains a string in which to preserve single quotation marks.

Usage: This function is useful in SQL statements to defer evaluation of a variable reference until runtime. This prevents errors that result from the evaluation of a single-quote or apostrophe data character (for example, "Joe's Diner") as a delimiter.

Example A: Consider this code:

```
<cfset mystring = "'Newton's Law', 'Fermat's Theorem'">
preservesinglequotes(#mystring#) is
<cfoutput>
         #preservesinglequotes(mystring)#
</cfoutput>
```

The output is as follows:

```
preservesinglequotes(#mystring#) is 'Newton's Law', 'Fermat's Theorem'
```

Example B: Consider this code:

```
<cfset list0 = " '1','2''3' ">
<cfquery sql = "select * from foo where bar in (#list0#)">
```

ColdFusion escapes the single-quote characters in the list as follows:

```
""1"", ""2"", ""3""
```

The `cfquery` tag throws an error.

You code this function correctly as follows:

```
<cfquery sql = "select * from foo where bar in (#preserveSingleQuotes(list0)#)">
```

This function ensures that ColdFusion evaluates the code as follows:

```
'1', '2', '3'
```

Example

```
<h3>PreserveSingleQuotes Example</h3><p>This is a useful function for
   creating lists of information to return from a query. In this example,
   we pick the list of Centers in Suisun, San Francisco, and San Diego,
   using the SQL grammar IN to modify a WHERE clause, rather than looping
   through the result set after the query is run.
<cfset List = "'Suisun', 'San Francisco', 'San Diego'">
```

```
<cfquery name = "GetCenters" datasource = "cfsnippets">
   SELECT Name, Address1, Address2, City, Phone
   FROM Centers
   WHERE City IN (#PreserveSingleQuotes(List)#)
</cfquery>
<p>We found <cfoutput>#GetCenters.RecordCount#</cfoutput> records.
<cfoutput query = "GetCenters">
<p>#Name#<br>
#Address1#<br>
<cfif Address2 is not "">#Address2#
         </cfif>
#City#<br>
#Phone#<br>
</cfoutput>
```

Quarter

Description: Calculates the quarter of the year in which a date falls.

Return value: An integer, 1–4.

Category: Date and time functions

Syntax: `Quarter(date)`

See also: `DatePart`, `Month`

Parameters

Parameter	Description
date	A date/time object in the range 100 AD–9999 AD. See "How ColdFusion processes two-digit year values" on page 377.

Usage: When passing a date/time value as a string, you must enclose it in quotation marks. Otherwise, it is interpreted as a number representation of a date/time object.

Example:

```
<h3>Quarter Example</h3>

Today, <cfoutput>#DateFormat(Now())#</cfoutput>,
is in Quarter <cfoutput>#Quarter(Now())#</cfoutput>.
```

QueryAddColumn

Description: Adds a column to a query and populates its rows with the contents of a one-dimensional array. Pads query columns, if necessary, to ensure that all columns have the same number of rows.

Return value: The number of the column that was added.

Category: Query functions

Syntax: `QueryAddColumn(query, column-name, array-name)`

See also: `QueryNew`, `QueryAddRow`, `QuerySetCell`

History: New in ColdFusion MX: If a user attempts to add a column whose name is invalid, ColdFusion throws an error. (In earlier releases, ColdFusion permitted the add operation, but the user could not reference the column after adding it.)

Parameters

Parameter	Description
query	Name of a query that was created with `QueryNew`.
column-name	Name of the new column.
array-name	Name of an array whose elements are to populate the new column.

Usage: You can add columns to query objects such as queries retrieved with the `cfquery` tag or queries created with the `QueryNew` function. You cannot us the `QueryAddColumn` function on a cached query.

Useful for generating a query object from the arrays of output parameters that Oracle stored procedures can generate.

Example:

```
<h3>QueryAddColumn Example</h3>
<p>This example adds three columns to a query object and then populates
         the columns with the contents of three arrays.</p>
<p>After populating the query, the example shows, in tabular format,
         the contents of the columns.</p>
<!--- make a query --->
<cfset myQuery = QueryNew("")>

<!--- create an array --->
<cfset FastFoodArray = ArrayNew(1)>
<cfset FastFoodArray[1] = "French Fries">
<cfset FastFoodArray[2] = "Hot Dogs">
<cfset FastFoodArray[3] = "Fried Clams">
<cfset FastFoodArray[4] = "Thick Shakes">

<!--- add a column to the query --->
<cfset nColumnNumber = QueryAddColumn(myQuery, "FastFood", FastFoodArray)>
<!--- create a second array --->
<cfset FineCuisineArray = ArrayNew(1)>
<cfset FineCuisineArray[1] = "Lobster">
<cfset FineCuisineArray[2] = "Flambe">
```

```
<!--- add a second column to the query --->
<cfset nColumnNumber2 = QueryAddColumn(myQuery, "FineCuisine", FineCuisineArray)>

<!--- create a third array --->
<cfset HealthFoodArray = ArrayNew(1)>
<cfset HealthFoodArray[1] = "Bean Curd">
<cfset HealthFoodArray[2] = "Yogurt">
<cfset HealthFoodArray[3] = "Tofu">

<!--- add a third column to the query --->
<cfset nColumnNumber3 = QueryAddColumn(myQuery, "HealthFood", HealthFoodArray)>

<table cellspacing = "2" cellpadding = "2" border = "0">
<tr>
  <th align = "left">Fast Food</th>
  <th align = "left">Fine Cuisine</th>
  <th align = "left">Health Food</th>
</tr>
<cfoutput query = "myQuery">
<tr>
  <td>#FastFood#</td>
  <td>#FineCuisine#</td>
  <td>#HealthFood#</td>
</tr>
</cfoutput>
</table>
<p><B>Note:</B> Because there are fewer elements in the Fine Cuisine
          and Health Food arrays, QueryAddColumn added padding to the
          corresponding columns in the query.</p>
```

QueryAddRow

Description: Adds a specified number of empty rows to a query.

Return value: The number of rows in a query

Category: Query functions

Syntax: `QueryAddRow(query [, number ])`

See also: QueryAddColumn, QueryAddRow, QuerySetCell, QueryNew

Parameters

Parameter	Description
query	Name of an executed query.
number	Number of rows to add to the query. Default: 1.

Example

```
<h3>QueryAddRow Example</h3>

<!--- start by making a query --->
<cfquery name = "GetCourses" datasource = "cfsnippets">
   SELECT Course_ID, Descript
   FROM Courses
</cfquery>
<p>The Query "GetCourses" has <cfoutput>#GetCourses.RecordCount#</cfoutput> rows.

<cfset CountVar = 0>
<cfloop CONDITION = "CountVar LT 15">
   <cfset temp = QueryAddRow(GetCourses)>
   <cfset CountVar = CountVar + 1>
   <cfset Temp = QuerySetCell(GetCourses, "Number", 100*CountVar)>
   <cfset CountVar = CountVar + 1>
   <cfset Temp = QuerySetCell(GetCourses, "Descript",
          "Description of variable #Countvar#")>
</cfloop>

<P>After the QueryAddRow action, the query has
   <CFOUTPUT>#GetCourses.RecordCount#</CFOUTPUT>
   records.
   <CFOUTPUT query="GetCourses">
   <PRE>#Course_ID# #Course_Number# #Descript#</pre> </cfoutput>
```

QueryNew

Description: Creates an empty query.

Return value: An empty query with a set of columns, or an empty query.

Category: Query functions

Syntax: `QueryNew(columnlist)`

See also: QueryAddColumn, QueryAddRow, QuerySetCell

Parameters

Parameter	Description
columnlist	A string or a variable that contains one. Delimited list of columns, or an empty string.

Usage: If you specify an empty string, you can add a column to the query and populate its rows with the contents of a one-dimensional array using `QueryAddColumn`.

Example:

```
<h3>QueryNew Example</h3>
<p>We will construct a new query with two rows:
<cfset myQuery = QueryNew("name, address, phone")>
<!--- make some rows in the query --->
<cfset newRow  = QueryAddRow(MyQuery, 2)>
<!--- set the cells in the query --->
<cfset temp = QuerySetCell(myQuery, "name", "Fred", 1)>
<cfset temp = QuerySetCell(myQuery, "address", "9 Any Lane", 1)>
<cfset temp = QuerySetCell(myQuery, "phone", "555-1212", 1)>
<cfset temp = QuerySetCell(myQuery, "name", "Jane", 2)>
<cfset temp = QuerySetCell(myQuery, "address", "14 My Street", 2)>
<cfset temp = QuerySetCell(myQuery, "phone", "555-1444", 2)>
<!--- output the query --->
<cfoutput query = "myQuery">
<pre>#name##address##phone#</pre>
</cfoutput>
To get any element in the query, we can output it individually
<cfoutput>
   <p>#MyQuery.name[2]#'s phone number: #MyQuery.phone[2]#
</cfoutput>
```

QuerySetCell

Description Sets a cell to a value. If no row number is specified, the cell on the last row is set.

Return value True, if successful; False, otherwise.

Category Query functions

Syntax `QuerySetCell(query, column_name, value [, row_number ])`

See also `QueryAddColumn`, `QueryAddRow`, `QueryNew`

Parameters

Parameter	Description
query	Name of an executed query
column_name	Name of a column in the query
value	Value to set in the cell
row_number	Row number. Default: last row.

Example

```
<!--- This example shows the use of QueryAddRow and QuerySetCell --->

<!--- start by making a query --->
<cfquery name = "GetCourses" datasource = "cfsnippets">
   SELECT Course_ID, Descript
   FROM Courses
</cfquery>
<p>The Query "GetCourses" has <cfoutput>#GetCourses.RecordCount#</cfoutput> rows.

<cfset CountVar = 0>
<cfloop CONDITION = "CountVar LT 15">
   <cfset temp = QueryAddRow(GetCourses)>
   <cfset CountVar = CountVar + 1>
   <cfset Temp = QuerySetCell(GetCourses, "Number", 100*CountVar)>
   <cfset CountVar = CountVar + 1>
   <cfset Temp = QuerySetCell(GetCourses, "Descript",
          "Description of variable #Countvar#")>
</cfloop>

<P>After the QueryAddRow action, the query has
   <CFOUTPUT>#GetCourses.RecordCount#</CFOUTPUT>
   records.
   <CFOUTPUT query="GetCourses">
   <PRE>#Course_ID# #Course_Number# #Descript#</pre> </cfoutput>
```

QuotedValueList

Description — Gets the values of each record returned from an executed query. ColdFusion does not evaluate the arguments.

Return value — A delimited list of the values of each record returned from an executed query. Each value is enclosed in single quotation marks.

Category — Query functions

Syntax — `QuotedValueList(query.column [, delimiter ])`

See also — `ValueList`

Parameters

Parameter	Description
query.column	Name of an executed query and column. Separate query name and column name with a period.
delimiter	A string or a variable that contains one. Character(s) that separate column data.

Example

```
<h3>QuotedValueList Example</h3>
<!--- use the contents of one query to create another dynamically --->
<cfset List = "'BIOL', 'CHEM'">
<!--- first, get the department IDs in our list --->
<cfquery name = "GetDepartments" datasource = "cfsnippets">
   SELECT Dept_ID FROM Departments
   WHERE Dept_ID IN (#PreserveSingleQuotes(List)#)
</cfquery>
<!--- now, select the courses for that department based on the
quotedValueList produced from our previous query --->
<cfquery name = "GetCourseList" datasource = "cfsnippets">
   SELECT *
   FROM CourseList
   WHERE Dept_ID IN (#QuotedValueList(GetDepartments.Dept_ID)#)
</cfquery>
<!--- now, output the results --->
<cfoutput QUERY = "GetCourseList" >
<pre>#Course_ID##Dept_ID##CorNumber##CorName#</pre>
</cfoutput>
```

Rand

Description: Generates a random number.

Return value: A random decimal number, in the range 0 – 1.

Category: Mathematical functions

Syntax: `Rand()`

See also: `Randomize`, `RandRange`

Usage: To ensure greater randomness, call the `Randomize` function before calling `Rand`.

Example:

```
<h3>Rand Example</h3>
<p>Rand() returns a random number in the range 0 to 1.
<p>Rand() returned: <cfoutput>#Rand()#</cfoutput>
<p><A HREF = "rand.cfm">Try again</A>
```

Randomize

Description: Seeds the ColdFusion random number generator with an integer number. Seeding the generator helps ensure that the `Rand` function generates highly random numbers.

Return value: A non-random decimal number, in the range 0 – 1.

Category: Mathematical functions

Syntax: `Randomize(number)`

See also: `Rand`, `RandRange`

Parameters:

Parameter	Description
number	A number

Usage: Call this function before calling `Rand`. Although this function returns a decimal number, it is not a random number.

Example:

```
<h3>Randomize Example</h3>
<p>Call Randomize to seed the random number generator. This helps
         to ensure the randomness of numbers generated by Rand.
<cfif IsDefined("FORM.myRandomInt")>
   <cfif IsNumeric(FORM.myRandomInt)>
     <cfoutput><p><b>Seed value is #FORM.myRandomInt#</b>
      </cfoutput><br>
    <cfset r = Randomize(FORM.myRandomInt)>
     <cfloop index = "i" from = "1" to = "10" step = "1">
       <cfoutput>Next random number is #Rand()#</cfoutput><br>
     </cfloop><br>
   <cfelse>
     <p>Please enter a number.
   </cfif>
</cfif>
<form action = "randomize.cfm">
<p>Enter a number to seed the randomizer:
<input type = "Text" name = "MyRandomInt">
<p><input type = "Submit" name = "">
</form>
```

RandRange

Description: Generates a random integer between two specified numbers. Requests for random integers that are greater than 100,000,000 result in non-random numbers, to prevent overflow during internal computations.

Return value: A random integer

Category: Mathematical functions

Syntax: `RandRange(number1, number2)`

See also: `Rand`, `Randomize`

Parameters:

Parameter	Description
number1, number2	Integer numbers less than 100,000,000

Example:

```
<h3>RandRange Example</h3>
<p>RandRange returns an integer between two specified integers.
<cfif IsDefined("FORM.myInt")>
   <p>RandRange returned:
      <cfoutput>#RandRange(FORM.myInt, FORM.myInt2)#</cfoutput>
</cfif>

<cfform action = "randRange.cfm">
<p>Enter a number to seed the randomizer:
<cfinput type = "Text" name = "MyInt" value = "1" RANGE = "1,100000000"
 message = "Please enter a value between 1 and 100,000,000"
  validate = "integer" required = "Yes">
<cfinput type = "Text" name = "MyInt2" value = "500" RANGE = "1,100000000"
 message = "Please enter a value between 1 and 100,000,000"
  validate = "integer" required = "Yes">
<p><input type = "Submit" name = "">
</cfform>
```

REFind

Description

Uses a regular expression (RE) to search a string for a pattern. The search is case sensitive.

For more information on regular expressions, including escape sequences, anchors, and modifiers, see *Developing ColdFusion MX Applications with CFML*.

Return value

Depends on the value of the `returnsubexpressions` parameter:

- If `returnsubexpressions = "False"`:
  - The position in the string where the match begins
  - 0, if the regular expression is not matched in the string
- If `returnsubexpressions = "True"`: a structure that contains two arrays, `len` and `pos`. The array elements are as follows:
  - If the regular expression is found in the string, the first element of the `len` and `pos` arrays contains the length and position, respectively, of the first match of the entire regular expression.
    If the regular expression contains parentheses that group subexpressions, each subsequent array element contains the length and position, respectively, of the first occurrence of each group.
  - If the regular expression is not found in the string, the first element of the `len` and `pos` arrays contains 0.

Category

String functions

Syntax

`REFind(reg_expression, string [, start ] [, returnsubexpressions ] )`

See also

`Find`, `FindNoCase`, `REFindNoCase`, `REReplace`, `REReplaceNoCase`

Parameters

Parameter	Description
reg_expression	Regular expression for which to search. Case-sensitive.
string	A string, or a variable that contains one, in which to search.
start	Optional. A positive integer, or a variable that contains one. Position in the string at which to start search. Default: 1.
returnsubexpressions	Optional. Boolean. Whether to return substrings of reg_expression, in arrays named `len` and `pos`: • True: if the regular expression is found, the first array element contains the length and position, respectively, of the first match. If the regular expression contains parentheses that group subexpressions, each subsequent array element contains the length and position, respectively, of the first occurrence of each group. If the regular expression is not found, the arrays each contain one element with the value 0. • False: the function returns the position in the string where the match begins. Default.

Usage This function finds the first occurrence of a regular expression in a string. To find the second and subsequent instances of the expression or of subexpressions in it, you call this function more than once, each time with a different start position. To determine the next start position, use the `returnsubexpressions` parameter, and add the value returned in the first element of the length array to the value in the first element of the position array.

Example
```
<h3>REFind Example</h3>
<p>This example shows the use of the REFind function with and without the
         <i>returnsubexpressions</i> parameter set to True.
         If you do not use the <i>returnsubexpressions</i> parameter,
         REFind returns the position of the first occurrence of a regular
         expression in a string starting from the specified position.
         Returns 0 if no occurrences are found.</p>

<p>REFind("a+c+", "abcaaccdd"):
<cfoutput>#REFind("a+c+", "abcaaccdd")#</cfoutput></p>
<p>REFind("a+c*", "abcaaccdd"):
<cfoutput>#REFind("a+c*", "abcaaccdd")#</cfoutput></p>
<p>REFind("[[:upper:]]", "abcaacCDD"):
<cfoutput>#REFind("[[:upper:]]", "abcaacCDD")#</cfoutput></p>
<p>REFind("[\?&]rep = ", "report.cfm?rep = 1234&u = 5"):
  <cfoutput>#REFind("[\?&]rep = ", "report.cfm?rep = 1234&u = 5")#
  </cfoutput>
</p>
<!--- Set startPos to one; returnMatchedSubexpressions = TRUE --->
<hr size = "2" color = "#0000A0">
<p>If you use the <i>returnssubexpression</i> parameter, REFind returns the
         position and length of the first occurrence of a regular expression
         in a string starting from the specified position. The position and
         length variables are stored in a structure. To access position and length
         information, use the keys <i>pos</i> and <i>len</i>, respectively.</p>
<cfset teststring = "The cat in the hat hat came back!">
<p>The string in which the function is to search is:
<cfoutput><b>#teststring#</b></cfoutput>.</p>
<p>The first call to REFind to search this string is:
     <b>REFind("[A-Za-z]+",testString,1,"TRUE")</b></p>
<p>This function returns a structure that contains two arrays: pos and len.</p>
<p>To create this structure you can use a CFSET statement, for example: </p>
&lt;CFSET st = REFind("[[:alpha:]]",testString,1,"TRUE")&gt;
<cfset st = REFind("[[:alpha:]]",testString,1,"TRUE")>
<p>
  <cfoutput>
  The number of elements in each array: #ArrayLen(st.pos)#.
  </cfoutput></p>
<p><b>The number of elements in the pos and len arrays is always one
         if you do not use parentheses in the regular expression.</b></p>
<p>The value of st.pos[1] is: <cfoutput>#st.pos[1]#.</cfoutput></p>
<p>The value of st.len[1] is: <cfoutput>#st.len[1]#.</cfoutput></p>
<p>
<cfoutput>
  Substring is <b>[#Mid(testString,st.pos[1],st.len[1])#]</B>
  </cfoutput></p>
<hr size = "2" color = "#0000A0">
```

```
<p>However, if you use parentheses in the regular expression, the first
         element contains the position and length of the first instance
         of the whole expression. The position and length of the first instance
         of each parenthesized subexpression within is included in additional
         array elements.</p>
<p>For example:
&lt;CFSET st1 = REFind("([[:alpha:]])[ ]+(\1)",testString,1,"TRUE")&gt;</p>
<cfset st1 = REFind("([[:alpha:]]+)[ ]+(\1)",testString,1,"TRUE")>
<p>The number of elements in each array is <cfoutput>#ArrayLen(st1.pos)#
         </cfoutput>.</p>
<p>First whole expression match; position is
   <cfoutput>#st1.pos[1]#;
      length is #st1.len[1]#; whole expression match is
      <B>[#Mid(testString,st1.pos[1],st1.len[1])#]</B>
   </cfoutput></p>
<p>Subsequent elements of the arrays provide the position and length of
         the first instance of each parenthesized subexpression therein.</p>
 <cfloop index = "i" from = "2" to = "#ArrayLen(st1.pos)#">
   <p><cfoutput>Position is #st1.pos[i]#; Length is #st1.len[i]#;
         Substring is <B>[#Mid(testString,st1.pos[i],st1.len[i])#]
         </B></cfoutput></p>
</cfloop><br>
```

REFindNoCase

Description

Uses a regular expression (RE) to search a string for a pattern, starting from a specified position. The search is case-insensitive.

For more information on regular expressions, including escape sequences, anchors, and modifiers, see *Developing ColdFusion MX Applications with CFML*.

Return value

Depends on the value of the `returnsubexpressions` parameter:

- If `returnsubexpressions = "False"`:
  - The position in the string where the match begins
  - 0, if the regular expression is not matched in the string
- If `returnsubexpressions = "True"`: a structure that contains two arrays, `len` and `pos`. The array elements are as follows:
  - If the regular expression is found in the string, the first element of the `len` and `pos` arrays contains the length and position, respectively, of the first match of the entire regular expression.
    If the regular expression contains parentheses that group subexpressions, each subsequent array element contains the length and position, respectively, of the first occurrence of each group.
  - If the regular expression is not found in the string, the first element of the `len` and `pos` arrays contains 0.

Category

String functions

Syntax

`REFindNoCase(`*reg_expression*`, `*string*` [, `*start*`] [, `*returnsubexpressions*`] )`

See also

`Find`, `FindNoCase`, `REFind`, `REReplace`, `REReplaceNoCase`

Parameters

Parameter	Description
reg_expression	Regular expression for which to search. Case-insensitive. For more information, see Developing ColdFusion MX Applications with CFML.
string	A string or a variable that contains one. String in which to search.
start	Optional. A positive integer or a variable that contains one. Position at which to start search. Default: 1.
returnsubexpressions	Optional. Boolean. Whether to return substrings of reg_expression, in arrays named `len` and `pos`: • True: if the regular expression is found, the first array element contains the length and position, respectively, of the first match. If the regular expression contains parentheses that group subexpressions, each subsequent array element contains the length and position, respectively, of the first occurrence of each group. If the regular expression is not found, the arrays each contain one element with the value 0. • False: the function returns the position in the string where the match begins. Default.

Usage This function finds the first occurrence of a regular expression in a string. To find the second and subsequent instances of the expression or of subexpressions in it, you call this function more than once, each time with a different start position. To determine the next start position, use the `returnsubexpressions` parameter, and add the value returned in the first element of the length array to the value in the first element of the position array.

Example
```
<h3>REFindNoCase Example</h3>
<p>This example demonstrates the use of the REFindNoCase function with and
        without the <i>returnsubexpressions</i> parameter set to True.</p>
<p>If you do not use the <i>returnsubexpressions</i> parameter, REFindNoCase
        returns the position of the first occurrence of a regular expression
        in a string starting from the specified position. Returns 0 if no
        occurrences are found. </p>
<p>REFindNoCase("a+c+", "abcaaccdd"):
<cfoutput>#REFindNoCase("a+c+", "abcaaccdd")#</cfoutput></p>
<p>REFindNoCase("a+c*", "abcaaccdd"):
<cfoutput>#REFindNoCase("a+c*", "abcaaccdd")#</cfoutput></p>
<p>REFindNoCase("[[:alpha:]]+", "abcaacCDD"):
<cfoutput>#REFindNoCase("[[:alpha:]]+", "abcaacCDD")#</cfoutput></p>
<p>REFindNoCase("[\?&]rep = ", "report.cfm?rep = 1234&u = 5"):
<cfoutput>#REFindNoCase("[\?&]rep = ", "report.cfm?rep = 1234&u = 5")#
        </cfoutput></p>
<!--- Set startPos to one; returnMatchedSubexpressions = True --->
<hr size = "2" color = "#0000A0">
<p>If you do use the <i>returnssubexpression</i> parameter, REFindNoCase returns
        the position and length of the first occurrence of a regular expression
        in a string starting from the specified position. The position and length
        variables are stored in a structure. To access position and length
        information, use the keys <i>pos</i> and <i>len</i>, respectively.</p>

<cfset teststring = "The cat in the hat hat came back!">
<p>The string in which the function is to search is:
        <cfoutput><b>#teststring#</b></cfoutput>.</p>
<p>The first call to REFindNoCase to search this string is:
<b>REFindNoCase("[[:alpha:]]+",testString,1,"True")</b></p>
<p>This function returns a structure that contains two arrays: pos and len.</p>
<p>To create this structure you can use a CFSET statement,
        for example:</p>
&lt;CFSET st = REFindNoCase("[[:alpha:]]+",testString,1,"True")&gt;
<cfset st = REFindNoCase("[[:alpha:]]+",testString,1,"True")>
<p>
  <cfoutput>
  The number of elements in each array: #ArrayLen(st.pos)#.
  </cfoutput></p>
<p><b>The number of elements in the pos and len arrays will always be one,
        if you do not use parentheses to denote subexpressions in the regular
        expression.</b></p>
<p>The value of st.pos[1] is: <cfoutput>#st.pos[1]#.</cfoutput></p>
<p>The value of st.len[1] is: <cfoutput>#st.len[1]#.</cfoutput></p>
<p>
  <cfoutput>
  Substring is <b>[#Mid(testString,st.pos[1],st.len[1])#]</B>
  </cfoutput></p>
<hr size = "2" color = "#0000A0">
```

```
<p>However, if you use parentheses to denote subexpressions in the regular
         expression, the first element contains the position and length of
         the first instance of the whole expression. The position and length
         of the first instance of each subexpression within will be included
         in additional array elements.</p>
<p>For example:
&lt;CFSET st1 = REFindNoCase("([[:alpha:]]+)[ ]+(\1)",testString,1,"True")&gt;</p>
<cfset st1 = REFindNoCase("([[:alpha:]]+)[ ]+(\1)",testString,1,"True")>

<p>The number of elements in each array is
<cfoutput>
  #ArrayLen(st1.pos)#
</cfoutput>.</p>

<p>First whole expression match; position is
<cfoutput>
  #st1.pos[1]#; length is #st1.len[1]#;
  whole expression match is <B>[#Mid(testString,st1.pos[1],st1.len[1])#]</B>
</cfoutput></p>

<p>Subsequent elements of the arrays provide the position and length of the
         first instance of each parenthesized subexpression therein.</p>
 <cfloop index = "i" from = "2" to = "#ArrayLen(st1.pos)#">
  <p><cfoutput>Position is #st1.pos[i]#; Length is #st1.len[i]#;
  Substring is <B>[#Mid(testString,st1.pos[i],st1.len[i])#]</B>
  </cfoutput></p>
</cfloop><br>
```

RemoveChars

Description Removes characters from a string.

Return value A copy of the string, with *count* characters removed from the specified start position. If no characters are found, returns zero.

Category String functions

Syntax `RemoveChars(string, start, count)`

See also `Insert, Len`

Parameters

Parameter	Description
string	A string or a variable that contains one. String in which to search.
start	A positive integer or a variable that contains one. Position at which to start search.
count	Number of characters to remove

Example

```
<h3>RemoveChars Example</h3>
Returns a string with <I>count</I> characters removed from the
         start position. Returns 0 if no characters are found.

<cfif IsDefined("FORM.myString")>
   <cfif Evaluate(FORM.numChars + FORM.start)
    GT Len(FORM.myString)>
      <p>Your string is only <cfoutput>#Len(FORM.myString)#
       </cfoutput> characters long.
      Please enter a longer string, select fewer characters to remove or
      begin earlier in the string.
   <cfelse>
      <cfoutput>
      <p>Your original string: #FORM.myString#
      <p>Your modified string:#RemoveChars(FORM.myString,
       FORM.numChars, FORM.start)#
      </cfoutput>
```

RepeatString

Description: Creates a string that contains a specified number of repetitions of the specified string.

Return value: A string.

Category: String functions

Syntax: `RepeatString(string, count)`

See also: CJustify, LJustify, RJustify

Parameters

Parameter	Description
string	A string or a variable that contains one
count	Number of repeats

Example

```
<h3>RepeatString Example</h3>
<p>RepeatString returns a string created from <I>string</I>, repeated
        a specified number of times.
<ul>
  <li>RepeatString("-", 10): <cfoutput>#RepeatString("-", 10)#</cfoutput>
  <li>RepeatString("&lt;BR&gt;", 3): <cfoutput>#RepeatString("<br>", 3)#
       </cfoutput>
  <li>RepeatString("", 5): <cfoutput>#RepeatString("", 5)#</cfoutput>
  <li>RepeatString("abc", 0): <cfoutput>#RepeatString("abc", 0)#</cfoutput>
  <li>RepeatString("Lorem Ipsum", 2):
   <cfoutput>#RepeatString("Lorem Ipsum", 2)#</cfoutput>
</ul>
```

Replace

Description: Replaces occurrences of *substring1* in a string with *substring2*, in a specified scope. The search is case-sensitive.

Return value: The string, after making replacements.

Category: String functions

Syntax: `Replace(string, substring1, substring2 [, scope ])`

See also: `Find`, `REFind`, `ReplaceNoCase`, `ReplaceList`, `REReplace`

Parameters

Parameter	Description
string	A string or a variable that contains one. String in which to search
substring1	A string or a variable that contains one. String for which to search
substring2	String that replaces `substring1`
scope	• one: replace the first occurrence (default) • all: replace all occurrences

Example

```
<h3>Replace Example</h3>

<p>The Replace function returns <I>string</I> with <I>substring1</I>
         replaced by <I>substring2</I> in the specified scope. This
         is a case-sensitive search.

<cfif IsDefined("FORM.MyString")>
<p>Your original string, <cfoutput>#FORM.MyString#</cfoutput>
<p>You wanted to replace the substring <cfoutput>#FORM.MySubstring1#
         </cfoutput>
         with the substring <cfoutput>#FORM.MySubstring2#</cfoutput>.
<p>The result: <cfoutput>#Replace(FORM.myString,
         FORM.MySubstring1, FORM.mySubString2)#</cfoutput>
</cfif>
```

ReplaceList

Description Replaces occurrences of the elements from a delimited list in a string with corresponding elements from another delimited list. The search is case-sensitive.

Return value A copy of the string, after making replacements.

Category List functions, String functions

Syntax `ReplaceList(string, list1, list2)`

See also `Find`, `REFind`, `Replace`, `REReplace`

Parameters

Parameter	Description
string	A string, or a variable that contains one, within which to replace substring
list1	Comma-delimited list of substrings for which to search
list2	Comma-delimited list of replacement substrings

Usage The list of substrings to replace is processed sequentially. If a *list1* element is contained in *list2* elements, recursive replacement might occur. The second example shows this.

Example

```
<p>The Replacelist function returns <I>string</I> with
<I>substringlist1</I> (e.g. "a,b") replaced by <I>substringlist2</I>
         (e.g. "c,d") in the specified scope.
<cfif IsDefined("FORM.MyString")>
<p>Your original string, <cfoutput>#FORM.MyString#</cfoutput>
<p>You wanted to replace the substring <cfoutput>#FORM.MySubstring1#
 </cfoutput>
with the substring <cfoutput>#FORM.MySubstring2#</cfoutput>.
<p>The result: <cfoutput>#Replacelist(FORM.myString,
FORM.MySubstring1, FORM.mySubString2)#</cfoutput>
</cfif>
<form action = "replacelist.cfm" method="post">
<p>String 1
<br><input type = "Text" value = "My Test String" name = "MyString">
<p>Substring 1 (find this list of substrings)
<br><input type = "Text" value = "Test, String" name = "MySubstring1">
<p>Substring 2 (replace with this list of substrings)
<br><input type = "Text" value = "Replaced, Sentence" name = "MySubstring2">
<p><input type = "Submit" value = "Replace and display" name = "">
</form>

<h3>Replacelist Example Two</h3>
<cfset stringtoreplace = "The quick brown fox jumped over the lazy dog.">
<cfoutput>
  #replacelist(stringtoreplace,"dog,brown,fox,black", "cow,black,ferret,white")#
</cfoutput>
```

ReplaceNoCase

Description: Replaces occurrences of *substring1* with *substring2*, in the specified scope. The search is case-insensitive.

Return value: A copy of the string, after making replacements.

Category: String functions

Syntax: `ReplaceNoCase(string, substring1, substring2 [, scope ])`

See also: Find, REFind, Replace, ReplaceList, REReplace

Parameters

Parameter	Description
string	A string within which to replace substring
substring1	String, or a variable that contains one, to replace.
substring2	String or a variable that contains one, that replaces `substring1`
scope	• one: Replace the first occurrence (default) • all: Replace all occurrences

Example

```
<h3>ReplaceNoCase Example</h3>
<p>The ReplaceNoCase function returns <I>string</I> with <I>substring1</I>
         replaced by <I>substring2</I> in the specified scope.
         The search/replace is case-insensitive.

<cfif IsDefined("FORM.MyString")>
<p>Your original string, <cfoutput>#FORM.MyString#</cfoutput>
<p>You wanted to replace the substring <cfoutput>#FORM.MySubstring1#
 </cfoutput>
with the substring <cfoutput>#FORM.MySubstring2#</cfoutput>.
<p>The result: <cfoutput>#ReplaceNoCase(FORM.myString,
FORM.MySubstring1, FORM.mySubString2)#</cfoutput>
</cfif>
```

REReplace

Description: Uses a regular expression (RE) to search a string for a string pattern and replace it with another. The search is case-sensitive.

Return value: If the `scope` attribute is set to `one`, returns a string with the first occurrence of the regular expression replaced by the value of *substring*.

If the `scope` attribute is set to `all`, returns a string with all occurrences of the regular expression replaced by the value of *substring*.

If the function finds no matches, it returns a copy of the string unchanged.

Category: String functions

Syntax: `REReplace(string, reg_expression, substring [, scope ])`

See also: `REFind`, `Replace`, `ReplaceList`, `REReplaceNoCase`

History: New in ColdFusion MX: this function supports the following special codes in a substring, to control case conversion:

- \u - uppercase the next character
- \l - lowercase the next character
- \U - uppercase until \E
- \L - lowercase until \E
- \E - end \U or \L

As in ColdFusion 5, the characters \1, \2, and so on, are backreference codes. To include these literal characters in a substring, you must escape them, by inserting a backslash before them (for example, change "\u" to "\\u").

To include a backslash followed by a backreference or case conversion code, you must escape the backslash, by prefixing another backslash (for example, "\\\1").

For more information on new features, see `REFind`.

Parameters:

Parameter	Description
string	A string or a variable that contains one. String within which to search.
reg_expression	Regular expression to replace. The search is case-sensitive.
substring	A string or a variable that contains one. Replaces `reg_expression`.
scope	• one: replace the first occurrence (default) • all: replace all occurrences

Usage: For more information, see *Developing ColdFusion MX Applications with CFML*.

Example:

```
<p>The REReplace function returns <i>string</i> with a regular expression replaced
        with <i>substring</i> in the specified scope. Case-sensitive search.
<p>REReplace("CABARET","C|B","G","ALL"):
<cfoutput>#REReplace("CABARET","C|B","G","ALL")#</cfoutput>
<p>REReplace("CABARET","[A-Z]","G","ALL"):
<cfoutput>#REReplace("CABARET","[A-Z]","G","ALL")#</cfoutput>
```

```
<p>REReplace("I love jellies","jell(y|ies)","cookies"):
<cfoutput>#REReplace("I love jellies","jell(y|ies)","cookies")#
 </cfoutput>
<p>REReplace("I love jelly","jell(y|ies)","cookies"):
<cfoutput>#REReplace("I love jelly","jell(y|ies)","cookies")#</cfoutput>
```

REReplaceNoCase

Description
Uses a regular expression to search a string for a string pattern and replace it with another. The search is case-insensitive.

Return value
- If `scope = "one"`: returns a string with the first occurrence of the regular expression replaced by the value of *substring*.
- If `scope = "all"`: returns a string with all occurrences of the regular expression replaced by the value of *substring*.
- If the function finds no matches: returns a copy of the string, unchanged.

Category
String functions

Syntax
`REReplaceNoCase(string, reg_expression, substring [, scope ])`

See also
`REFind`, `REFindNoCase`, `Replace`, `ReplaceList`

History
New in ColdFusion MX: this function inserts the following special characters in regular expression replacement strings, to control case conversion: \u, \U, \l, \L, and \E. If any of these strings is present in a ColdFusion 5 application, you must insert a backslash before it (for example, change "\u" to "\\u").

Parameters

Parameter	Description
string	A string or a variable that contains one.
reg_expression	Regular expression to replace. For more information, see Developing ColdFusion MX Applications with CFML.
substring	A string or a variable that contains one. Replaces *reg_expression*.
scope	• `one`: Replace the first occurrence of the regular expression. Default. • `all`: Replace all occurrences of the regular expression.

Usage
To include a backslash character in a parameter of this function, you must precede it with another backslash, as an escape character.

Example
```
<p>The REReplaceNoCase function returns <i>string</i> with a regular
         expression replaced with <i>substring</i> in the specified scope.
         This is a case-insensitive search.
<p>REReplaceNoCase("cabaret","C|B","G","ALL"):
<cfoutput>#REReplaceNoCase("cabaret","C|B","G","ALL")#</cfoutput>
<p>REReplaceNoCase("cabaret","[A-Z]","G","ALL"):
<cfoutput>#REReplaceNoCase("cabaret","[A-Z]","G","ALL")#</cfoutput>
<p>REReplaceNoCase("I LOVE JELLIES","jell(y|ies)","cookies"):
<cfoutput>#REReplaceNoCase("I LOVE JELLIES","jell(y|ies)","cookies")#
</cfoutput>
<p>REReplaceNoCase("I LOVE JELLY","jell(y|ies)","cookies"):
<cfoutput>#REReplaceNoCase("I LOVE JELLY","jell(y|ies)","cookies")#
         </cfoutput>
```

Reverse

Description — Reverses the order of items such as the characters in a string, the digits in a number, or the elements in an array.

Return value — A copy of *string*, with the characters in reverse order.

Category — String functions

Syntax — `Reverse(string)`

See also — `Left`, `Mid`, `Right`

Parameters

Parameter	Description
string	A string or a variable that contains one

Usage — You can call this function on a number with code such as the following:

```
<cfoutput>reverse(6*2) equals #reverse(6*2)#<cfoutput>
```

This code outputs the following:

```
reverse(6*2) equals 21
```

Example

```
<h3>Reverse Example</h3>

<p>Reverse returns your string with the positions of the characters reversed.
<cfif IsDefined("FORM.myString")>
   <cfif FORM.myString is not "">
      <p>Reverse returned:
      <cfoutput>#Reverse(FORM.myString)#</cfoutput>
   <cfelse>
      <p>Please enter a string to be reversed.
   </cfif>
</cfif>

<form action = "reverse.cfm">
<p>Enter a string to be reversed:
<input type = "Text" name = "MyString">
<p><input type = "Submit" name = "">
</form>
```

Right

Description Gets a specified number of characters from a string, beginning at the right.

Return value
- If the length of the string is greater than or equal to *count*, the rightmost *count* characters of the string
- If *count* is greater than the length of the string, the whole string
- If *count* is greater than 1, and the string is empty, an empty string

Category String functions

Syntax `Right(string, count)`

See also `Mid`, `Left`, `Reverse`

Parameters

Parameter	Description
string	A string or a variable that contains one
count	A positive integer or a variable that contains one. Number of characters to return.

Example

```
<h3>Right Example</h3>
<cfif IsDefined("Form.MyText")>
<!--- if len is 0, then err --->
   <cfif Len(FORM.myText) is not 0>
      <cfif Len(FORM.myText) LTE FORM.RemoveChars>
      <p>Your string <cfoutput>#FORM.myText#</cfoutput>
      only has <cfoutput>#Len(FORM.myText)#</cfoutput>
      characters. You cannot output the <cfoutput>#FORM.removeChars#
      </cfoutput>
      rightmost characters of this string because it is not long enough
      <cfelse>
      <p>Your original string: <cfoutput>#FORM.myText#</cfoutput>
      <p>Your changed string, showing only the
          <cfoutput>#FORM.removeChars#</cfoutput> rightmost characters:
      <cfoutput>#right(Form.myText, FORM.removeChars)#
       </cfoutput>
      </cfif>
   <cfelse>
   <p>Please enter a string
   </cfif>
</cfif>
```

RJustify

Description: Right justifies characters of a string.

Return value: A copy of a string, right-justified in the specified field length.

Category: Display and formatting functions, String functions

Syntax: `RJustify(string, length)`

See also: CJustify, LJustify

Parameters

Parameter	Description
string	A string enclosed in quotation marks, or a variable that contains one.
length	A positive integer or a variable that contains one. Length of field in which to justify string.

Example

```
<!--- This example shows how to use RJustify --->
<cfparam name = "jstring" default = "">

<cfif IsDefined("FORM.justifyString")>
   <cfset jstring = rjustify(FORM.justifyString, 35)>
</cfif>
<html>
<head>
<title>RJustify Example</title>
</head>
<body>
<h3>RJustify Function</h3>
<p>Enter a string. It will be right justified within the sample field

<form action = "rjustify.cfm">
<p><input type = "Text" value = "<cfoutput>#jString#</cfoutput>"
          size = 35 name = "justifyString">

<p><input type = "Submit" name = ""> <input type = "reset">
</form>
```

Round

Description: Rounds a number to the closest integer.

Return value: An integer.

Category: Mathematical functions

Syntax: `Round(number)`

See also: `Ceiling`, `Fix`, `Int`

Parameters

Parameter	Description
number	Number to round

Example

```
<h3>Round Example</h3>
<p>This function rounds a number to the closest integer.
<ul>
   <li>Round(7.49) : <cfoutput>#Round(7.49)#</cfoutput>
   <li>Round(7.5) : <cfoutput>#Round(7.5)#</cfoutput>
   <li>Round(-10.775) : <cfoutput>#Round(-10.775)#</cfoutput>
   <li>Round(1.2345*100)/100 :
   <cfoutput>#Evaluate(Round(1.2345*100)/100)#</cfoutput>
</ul>
```

RTrim

Description: Removes spaces from the end of a string.

Return value: A copy of *string*, after removing trailing spaces.

Category: String functions

Syntax: `RTrim(string)`

See also: `LTrim`, `Trim`

Parameters

Parameter	Description
string	A string or a variable that contains one

Example

```
<h3>RTrim Example</h3>

<cfif IsDefined("FORM.myText")>
<cfoutput>
<pre>
Your string:"#FORM.myText#"
Your string:"#Rtrim(FORM.myText)#"
(right trimmed)
</pre>
</cfoutput>
</cfif>

<form action = "Rtrim.cfm" method="post">
<p>Enter some text. It will be modified by Rtrim to remove spaces from the right.
<p><input type = "Text" name = "myText" value = "TEST   ">

<p><input type = "Submit" name = "">
</form>
```

Second

Description Extracts the ordinal for the second from a date/time object.

Return value An integer in the range 0–59.

Category Date and time functions

Syntax `Second(date)`

See also `DatePart`, `Hash`, `Minute`

Parameters

Parameter	Description
date	A date/time object

Usage When passing a date/time object as a string, you must enclose it in quotation marks. Otherwise, it is interpreted as a number representation of a date/time object.

Example

```
<!--- This example shows the use of Hour, Minute, and Second --->
<h3>Second Example</h3>
<cfoutput>
The time is currently #TimeFormat(Now())#.
We are in hour #Hour(Now())#, Minute #Minute(Now())#
and Second #Second(Now())# of the day.
</cfoutput>
```

SetEncoding

Description: Sets the character encoding of Form and URL scope variable values; used when the character set of the input to a form, or the character set of a URL, is not in Latin-1 encoding (ISO-8859-1).

Return value: None

Category: Display and formatting functions, System functions

Syntax: `SetEncoding(scope_name,charset)`

See also: `cfcontent`, `cfprocessingdirective`, `URLDecode`, `URLEncodedFormat`

Parameters

Parameter	Description
scope_name	• url • form
charset	A Java character set name for the file contents. The following values are typically used: • EUC-JP • EUC-K • ISO-8859-1 • SHIFT-JIS • UTF-8 (Default) • UTF-16 • US-ASCII • UTF-16BE • UTF-16LE For a list of character sets, see: http://www.w3.org/International/O-charset-lang.html

Usage: Use this tag when the character set of the input to a form or the character set of a URL is not in Latin-1 encoding. For example, Traditional Chinese characters are in Big5 encoding. Before getting URL or FORM parameters, call this function (typically, in the application.cfml page) to set the encoding and avoid getting incorrect parameter values.

For more information, see: http://www.iana.org/assignments/character-sets.

The following example shows how you could process form data such as the following:

```
<form action='process.cfm' method='get'>
<input name='xxx' type='text'>
<input name='yyy' type='text'>
</form>
```

Example

```
<cfoutput>
   setEncoding("url", "big5");
   writeoutput(URL.xxx);
   writeoutput(URL.yyy);
</cfoutput>
```

SetLocale

Description
Sets a country/language locale option for a ColdFusion application user; the setting persists for the current ColdFusion application session. The locale value encapsulates a set of attributes that determine the default display format of date, time, number, and currency values, according to language and regional conventions.

For information about determining the locale value, see GetLocale on page 449.

Return value
An element from the list of locales that ColdFusion supports, as a string.

Category
International functions, System functions

Syntax
```
SetLocale(new_locale)
```

See also
GetHttpTimeString, GetLocale

History
New in ColdFusion MX:

- This function might return a different value than in earlier releases. This function uses Java standard locale determination and formatting rules on all platforms.
- The Spanish (Mexican) locale option is deprecated. Do not use it in new applications. It might not work, and it might cause an error, in later releases. (The Spanish (Modern) option now maps to Spanish (Standard).)

Parameters

Parameter	Description
new_locale	The name of a locale; for example, "en_US" (English, United States)

Usage
This function uses Java standard locale formatting rules on all platforms.

A locale value can be explicitly set in the following ways:

- From the command line; for example: java -Duser.language=de -Duser.region=DE.
- In JSP code; for example: `response.setLocale ( new Locale( "de", "DE") )`
- With the SetLocale function; for example: `SetLocale ("German (Standard)"`

The locale value persistence and usage hierarchy is as follows:

- When the locale is set explicitly, its value persists until the ColdFusion server is stopped. ColdFusion accesses this value.
- The ColdFusion Server operating system default locale value persists permanently. If the value has not been explicitly set and stored in the JSP request object, ColdFusion uses this value.
- If the operating system does not have a default value or the value is not one that ColdFusion supports, ColdFusion uses the value English (US).

ColdFusion supports the following locale options. You can use either the ColdFusion locale name or the Java standard locale string.

ColdFusion locale name	Java standard locale string
Chinese (China)	
Chinese (Hong Kong)	
Chinese (Taiwan)	
Dutch (Belgian)	nl_be
Dutch (Standard)	nl_NL
English (Australian)	en_AU
English (Canadian)	en_CA
English (New Zealand)	en_NZ
English (UK)	en_GB
English (US)	en_US
French (Belgian)	fr_BE
French (Canadian)	fr_CA
French (Standard)	fr_FR
French (Swiss)	fr_CH
German (Austrian)	de_AT
German (Standard)	de_DE
German (Swiss)	de_CH
Italian (Standard)	it_IT
Italian (Swiss)	it_CH
Japanese	ja_JP
Korean	ko_KR
Norwegian (Bokmal)	no_NO
Norwegian (Nynorsk)	no_NO_nynorsk
Portuguese (Brazilian)	pt_BR
Portuguese (Standard)	pt_PT
Spanish (Modern)	es_ES
Spanish (Standard)	es_ES
Swedish	sv_SE

Note: ColdFusion uses the Spanish (Standard) formats for Spanish (Modern) and Spanish (Standard).

This function can restore a locale setting by referencing the session variable that stores a locale, as follows:

```
<cfset oldlocale = SetLocale("localename")>
```

Note: The variable `server.ColdFusion.SupportedLocales` is initialized at startup with a comma-delimited list of the locales that ColdFusion and the operating system support. If it is called with a locale that is not in the list, `SetLocale` fails.

Example

```
<h3>SetLocale Example</h3>
<p>SetLocale sets the locale to the specified new locale for the current session.
<p>A locale encapsulates the set of attributes that govern the display and
         formatting of date, time, number, and currency values.
<p>The locale for this system is <cfoutput>#GetLocale()#</cfoutput>
<p><cfoutput><I>the old locale was #SetLocale("English (UK)")#</I>
<p>The locale is now #GetLocale()#</cfoutput>
```

SetProfileString

Description Sets the value of a profile entry in an initialization file.

Return value An empty string, upon successful execution; otherwise, an error message.

Category System functions

Syntax `SetProfileString(iniPath, section, entry, value)`

See also `GetProfileSections`, `GetProfileString`, `SetProfileString`

Parameters

Parameter	Description
iniPath	Absolute path of initialization file
section	Section of the initialization file in which the entry is to be set
entry	Name of the entry to set
value	Value to which to set the entry

Example

```
<h3>SetProfileString Example</h3>
This example uses SetProfileString to set the timeout value in an
         initialization file. Enter the full path of your initialization
         file, specify the timeout value, and submit the form.
<!--- This section checks whether the form was submitted. If so, this
         section sets the initialization path and timeout value to the
         path and timeout value specified in the form --->
<cfif Isdefined("Form.Submit")>

   <cfset IniPath = FORM.iniPath>
   <cfset Section = "boot loader">
   <cfset MyTimeout = FORM.MyTimeout>
   <cfset timeout = GetProfileString(IniPath, Section, "timeout")>

   <cfif timeout Is Not MyTimeout>
    <cfif MyTimeout Greater Than 0>
       <hr size = "2" color = "#0000A0">
      <p>Setting the timeout value to <cfoutput>#MyTimeout#</cfoutput>
      </p>
         <cfset code = SetProfileString(IniPath,
            Section, "timeout", MyTimeout)>
      <p>Value returned from SetProfileString:
           <cfoutput>#code#</cfoutput></p>
      <cfelse>
         <hr size = "2" color = "red">
         <p>Timeout value should be greater than zero in order to
            provide time for user response.</p>
         <hr size = "2" color = "red">
      </cfif>
   <cfelse>
      <p>The timeout value in your initialization file is already
        <cfoutput>#MyTimeout#</cfoutput>.</p>
   </cfif>
```

```
   <cfset timeout = GetProfileString(IniPath, Section, "timeout")>
   <cfset default = GetProfileString(IniPath, Section, "default")>

   <h4>Boot Loader</h4>
   <p>Timeout is set to: <cfoutput>#timeout#</cfoutput>.</p>
   <p>Default directory is: <cfoutput>#default#</cfoutput>.</p>

</cfif>

<form action = "setprofilestring.cfm">
<hr size = "2" color = "#0000A0">
<table cellspacing = "2" cellpadding = "2" border = "0">
<tr>
  <td>Full Path of Init File</td>
  <td><input type = "Text" name = "IniPath"
           value = "C:\myboot.ini"></td>
</tr>
<tr>
  <td>Timeout</td>
  <td><input type = "Text" name = "MyTimeout" value = "30"></td>
</tr>
<tr>
  <td><input type = "Submit" name = "Submit" value = "Submit"></td>
  <td></td>
</tr>
</table>
</form>
```

SetVariable

Description
This function is no longer required in well-formed ColdFusion pages.
Sets a variable in the `name` parameter to the value of the `value` parameter.

Return value
The new value of the variable.

Category
Dynamic evaluation functions

Syntax
`SetVariable(name, value)`

See also
`DE`, `Evaluate`, `IIf`

Parameters

Parameter	Description
name	Variable name
value	A string, the name of a string, or a number

Usage
Before this function is called, the client variable must exist, and the `cfapplication` tag `ClientManagement` attribute must be set to `"Yes"`.

Note: If you concatenate string elements to form the `name` parameter, you can improve performance using the `cfset` tag, instead.

For example:

```
<cfset "myVar#i#" = myVal>
```

is faster than:

```
SetVariable("myVar" & i, myVal)
```

For more information, see *Developing ColdFusion MX Applications with CFML*.

Example

```
<h3>SetVariable Example</h3>

<cfif IsDefined("FORM.myVariable")>
<!--- strip out url, client., cgi., session., caller. --->
<!--- This example only lets you set form variables --->
<cfset myName = ReplaceList(FORM.myVariable,
 "url,client,cgi,session,caller", "FORM,FORM,FORM,FORM,FORM")>

<cfset temp = SetVariable(myName, FORM.myValue)>
<cfset varName = myName>
<cfset varNameValue = Evaluate(myName)>
<cfoutput>
   <p>Your variable, #varName#
   <p>The value of #varName# is #varNameValue#
</cfoutput>
</cfif>
```

Sgn

Description Determines the sign of a number.

Return value
- 1, if *number* is positive.
- 0, if *number* is 0.
- -1, if *number* is negative.

Category Mathematical functions

Syntax `Sgn(number)`

See also `Abs`

Parameters

Parameter	Description
number	A number

Example

```
<h3>Sgn Example</h3>
<p>Sgn determines the sign of a number. Returns 1 if number is positive;
          0 if number is 0; -1 if number is negative.
<p>Sgn(14): <cfoutput>#Sgn(14)#</cfoutput>
<p>Sgn(21-21): <cfoutput>#Sgn(21-21)#</cfoutput>
<p>Sgn(-0.007): <cfoutput>#Sgn(-0.007)#</cfoutput>
```

Sin

Description Calculates the sine of an angle.

Return value A number; the sine of the angle *number*.

Category Mathematical functions

Syntax `Sin(number)`

See also ASin, Atn, Cos, Pi, Tan

Parameters

Parameter	Description
number	Angle, in radians. To convert an angle from degrees to radians, use the PI function to multiply the degrees by pi/180.

Example

```
<h3>Sin Example</h3>

<!--- output its Sin value --->
<cfif IsDefined("FORM.SinNum")>
   <cfif IsNumeric(FORM.SinNum)>
      Sin(<cfoutput>#FORM.SinNum#</cfoutput>) =
       <cfoutput>#Sin(FORM.sinNum)# Degrees =
        #Evaluate(Sin(FORM.sinNum) * PI()/180)# Radians
       </cfoutput>
   <cfelse>
<!--- if it is empty, output an error message --->
      <h4>Please enter an angle for which you want the Sine value</h4>
   </cfif>
</cfif>

<form action = "sin.cfm">
<p>Type in a number to get its sine in Radians and Degrees
<br><input type = "Text" name = "sinNum" size = "25">
<p><input type = "Submit" name = ""> <input type = "RESET">
</form>
```

SpanExcluding

Description: Gets characters from a string, from the beginning to a character that is in a specified set of characters. The search is case-sensitive.

Return value: A string; characters from *string*, from the beginning to a character that is in *set*.

Category: String functions

Syntax: `SpanExcluding(string, set)`

See also: `GetToken`, `SpanIncluding`

Parameters

Parameter	Description
string	A string or a variable that contains one
set	A string or a variable that contains one. Must contain one or more characters

Example

```
<h3>SpanExcluding Example</h3>

<cfif IsDefined("FORM.myString")>
<p>Your string was <cfoutput>#FORM.myString#</cfoutput>
<p>Your set of characters was <cfoutput>#FORM.mySet#</cfoutput>
<p>Your string up until one of the characters in the set is:
<cfoutput>#SpanExcluding(FORM.myString, FORM.mySet)#</cfoutput>
</cfif>

<p>Returns all characters from string from beginning to a character
         from the set of characters. The search is case-sensitive.

<form action = "spanexcluding.cfm">
<p>Enter a string:
<br><input type = "Text" name = "myString" value = "Hey, you!">
<p>And a set of characters:
<br><input type = "Text" name = "mySet" value = "Ey">
<br><input type = "Submit" name = "">
</form>
```

SpanIncluding

Description: Gets characters from a string, from the beginning to a character that is not in a specified set of characters. The search is case-sensitive.

Return value: A string; characters from *string*, from the beginning to a character that is not in *set*.

Category: String functions

Syntax: `SpanIncluding(string, set)`

See also: GetToken, SpanExcluding

Parameters

Parameter	Description
string	A string or a variable that contains the search string.
set	A string or a variable that contains a set of characters. Must contain one or more characters

Example

```
<h3>SpanIncluding Example</h3>
<cfif IsDefined("FORM.myString")>
<p>Your string was <cfoutput>#FORM.myString#</cfoutput>
<p>Your set of characters was <cfoutput>#FORM.mySet#</cfoutput>
<p>Your string, until the characters in the set have been found, is:
<cfoutput>#SpanIncluding(FORM.myString, FORM.mySet)#</cfoutput>
</cfif>

<p>Returns characters of a string, from beginning to a character
         that is not in set. The search is case-sensitive.

<form action = "spanincluding.cfm" method="post">
<p>Enter a string:
<br><input type = "Text" name = "myString" value = "Hey, you!">
<p>And a set of characters:
<br><input type = "Text" name = "mySet" value = "ey,H">
<br><input type = "Submit" name = "">
</form>
```

Sqr

Description: Calculates the square root of a number.

Return value: Number; square root of *number*.

Category: Mathematical functions

Syntax: `Sqr(number)`

See also: `Abs`

Parameters:

Parameter	Description
number	A positive integer or a variable that contains one. Number whose square root to get.

Usage: The value in `number` must be greater than or equal to 0.

Example:

```
<h3>Sqr Example</h3>

<p>Returns the square root of a positive number.

<p>Sqr(2): <cfoutput>#Sqr(2)#</cfoutput>
<p>Sqr(Abs(-144)): <cfoutput>#Sqr(Abs(-144))#</cfoutput>
<p>Sqr(25^2): <cfoutput>#Sqr(25^2)#</cfoutput>
```

StripCR

Description: Deletes return characters from a string.

Return value: A copy of *string*, after removing return characters.

Category: Display and formatting functions, Other functions, String functions

Syntax: `StripCR(string)`

See also: ParagraphFormat

Parameters:

Parameter	Description
string	A string or a variable that contains one

Usage: Useful for preformatted (between <pre> and </pre> tags) HTML display of data entered in textarea fields.

Example:

```
<h3>StripCR Example</h3>

<p>Function StripCR is useful for preformatted HTML display of data
         (PRE) entered in textarea fields.
<cfif isdefined("Form.myTextArea")>

<pre>
<cfoutput>#StripCR(Form.myTextArea)#</cfoutput>
</pre>
</cfif>
<!--- use #Chr(10)##Chr(13)# to simulate line feed/carriage return combination --->
<form action = "stripcr.cfm">
<textarea name = "MyTextArea" cols = "35" rows = 8>
This is sample text and you see how it scrolls
   <cfoutput>#Chr(10)##Chr(13)#</cfoutput>
From one line
   <cfoutput>#Chr(10)##Chr(13)##Chr(10)##Chr(13)#</cfoutput>
         to the next
</textarea>
<input type = "Submit" name = "Show me the HTML version">
</form>
```

StructAppend

Description: Appends one structure to another.

Return value: True, upon successful completion; False, otherwise.

Category: Structure functions

Syntax: `StructAppend(struct1, struct2, overwriteFlag)`

See also: Structure functions

History: New in ColdFusion MX: this function can be used on XML objects.

Parameters

Parameter	Description
struct1	Structure to append.
struct2	Structure that contains the data to append to `struct1`
overwriteFlag	• Yes: values in `struct2` overwrite corresponding values in `struct1`. Default. • No

Usage: This function appends the fields and values of `struct2` to `struct1`; `struct2` is not modified. If `struct1` already contains a field of `struct2`, `overwriteFlag` determines whether the value in `struct2` overwrites it.

A structure's keys are unordered.

Example:

```
<html>
<body>
<!---- Create a Name structure --->
<cfset nameCLK=StructNew()>
<cfset nameCLK.first="Chris">
<cfset nameCLK.middle="Lloyd">
<cfset nameCLK.last="Gilson">
<!--- Create an address struct --->
<cfset addrCLK=StructNew()>
<cfset addrCLK.street="17 Gigantic Rd">
<cfset addrCLK.city="Watertown">
<cfset addrCLK.state="MA">
<cfset addrCLK.zip="02472">
<!---- Create a Person structure --->
<cfset personCLK=StructNew()>
<cfset personCLK.name=#nameCLK#>
<cfset personCLK.addr=#addrCLK#>
<!--- Display the contents of the person struct before the Append --->
<p>
The person struct <b>before</b> the Append call:<br>
<cfloop collection=#personCLK# item="myItem">
<cfoutput>
#myItem#<br>
</cfoutput>
</cfloop>
```

```
<!--- Merge the Name struct into the top-level person struct --->
<cfset bSuccess = StructAppend( personCLK, addrCLK )>

<!--- Display the contents of the person struct, after the Append --->
<p>
The person struct <b>after</b> the Append call:<br>
<cfloop collection=#personCLK# item="myItem">
   <cfoutput>
      #myItem#<br>
   </cfoutput>
</cfloop>
```

StructClear

Description: Removes all data from a structure.

Return value: True, on successful execution; False, otherwise.

Category: Structure functions

Syntax: `StructClear(structure)`

See also: Structure functions

History: New in ColdFusion MX: this function can be used on XML objects.

Parameters

Parameter	Description
structure	Structure to clear

Usage: Do not call this function on a session variable. For more information, see TechNote 14143, "*ColdFusion 4.5 and the StructClear(Session) function*," at http://www.coldfusion.com/Support/KnowledgeBase/SearchForm.cfm. (The article applies to ColdFusion 4.5, 5.x, and ColdFusion MX.)

Example

```
<!--- Shows StructClear function. Calls cf_addemployee custom tag which
   uses the addemployee.cfm file. --->
<body>
<h1>Add New Employees</h1>
<!--- Establish parms for first time through --->
<cfparam name = "Form.firstname" default = "">
<cfparam name = "Form.lastname" default = "">
<cfparam name = "Form.email" default = "">
<cfparam name = "Form.phone" default = "">
<cfparam name = "Form.department" default = "">
<cfif form.firstname eq "">
 <p>Please fill out the form.
<cfelse>
 <cfoutput>
  <cfscript>
   employee = StructNew();
   StructInsert(employee, "firstname", Form.firstname);
   StructInsert(employee, "lastname", Form.lastname);
   StructInsert(employee, "email", Form.email);
   StructInsert(employee, "phone", Form.phone);
   StructInsert(employee, "department", Form.department);
  </cfscript>
 </cfoutput>
<!--- Call the custom tag that adds employees --->
 <cf_addemployee empinfo = "#employee#">
 <cfscript>StructClear(employee);</cfscript>
 </cfif>
```

StructCopy

Description: Copies a structure. Copies top-level keys, values, and arrays in the structure by value; copies nested structures by reference.

Return value: A copy of a structure, with the same keys and values; if *structure* does not exist, throws an exception.

Category: Structure functions

Syntax: `StructCopy(structure)`

See also: Structure functions

Parameters

Parameter	Description
structure	Structure to copy

Usage: The following code shows how this function copies a structure that contains a string field, a number field, and a two-dimensional array at the top level:

```
<cfoutput>
   <cfset assignedCopy = StructNew()>
  <cfset assignedCopy.string = #struct.string#>
   <cfset assignedCopy.number = #struct.number#>
   <cfset assignedCopy.array = ArrayNew(2)>
   <cfset assignedCopy.array[1][1] = #struct.array[1][1]#>
   <cfset assignedCopy.array[1][2] = #sruct.array[1][2]#>
</cfoutput>
```

The following code shows how *StructCopy* copies a nested structure:

```
<cfoutput>
  <cfset assignedCopy.nestedStruct = struct.nestedStruct>
</cfoutput>
```

To copy a structure entirely by value, use Duplicate on page 422.

The following table shows how variables are assigned:

Variable type	Assigned by
structure.any_simple_value Boolean Binary Base64	Value
structure.array	Value
structure.nested_structure	Reference
structure.object	Reference
structure.query	Reference

Example

```
<!--- This code shows assignment by-value and by-reference. --->
// This script creates a structure that StructCopy copies by value. <br>
<cfscript>
   // Create elements.
   s = StructNew();
   s.array = ArrayNew(2);

   // Assign simple values to original top-level structure fields.
   s.number = 99;
   s.string = "hello tommy";

   // Assign values to original top-level array.
   s.array[1][1] = "one one";
   s.array[1][2] = "one two";
</cfscript>

<!--- Output original structure --->
<hr>
<b>Original Values</b><br>
<cfoutput>
   // Simple values <br>
   s.number = #s.number#<br>
   s.string = #s.string#<br>
   // Array value <br>
   s.array[1][1] = #s.array[1][1]#<br>
   s.array[1][2] = #s.array[1][2]#<br>
</cfoutput>

// Copy this structure to a new structure. <br>
<cfset copied = StructCopy(s)>

<cfscript>
// Change the values of the original structure. <br>
   s.number = 100;
   s.string = "hello tommy (modified)";
   s.array[1][1] = "one one (modified)";
   s.array[1][2] = "one two (modified)";
</cfscript>
<hr>
<b>Modified Original Values</b><br>
<cfoutput>
   // Simple values <br>
   s.number = #s.number#<br>
   s.string = #s.string#<br>
   // Array value <br>
   s.array[1][1] = #s.array[1][1]#<br>
   s.array[1][2] = #s.array[1][2]#<br>
</cfoutput>
<hr>
<b>Copied structure values should be the same as the original.</b><br>
<cfoutput>
   // Simple values <br>
   copied.number = #copied.number#<br>
   copied.string = #copied.string#<br>
   // Array value <br>
```

```
   copied.array[1][1] = #copied.array[1][1]#<br>
   copied.array[1][2] = #copied.array[1][2]#<br>
</cfoutput>

// This script creates a structure that StructCopy copies by reference.
<cfscript>
   // Create elements.
   s = StructNew();
   s.nested = StructNew();
   s.nested.array = ArrayNew(2);
   // Assign simple values to nested structure fields.
   s.nested.number = 99;
   s.nested.string = "hello tommy";
   // Assign values to nested array.
   s.nested.array[1][1] = "one one";
   s.nested.array[1][2] = "one two";
</cfscript>

<!--- Output original structure --->
<hr>
<b>Original Values</b><br>
<cfoutput>
   // Simple values <br>
   s.nested.number = #s.nested.number#<br>
   s.nested.string = #s.nested.string#<br>

   // Array values <br>
   s.nested.array[1][1] = #s.nested.array[1][1]#<br>
   s.nested.array[1][2] = #s.nested.array[1][2]#<br>
</cfoutput>

// Use StructCopy to copy this structure to a new structure. <br>
<cfset copied = StructCopy(s)>
// Use Duplicate to clone this structure to a new structure. <br>
<cfset duplicated = Duplicate(s)>

<cfscript>
// Change the values of the original structure.
   s.nested.number = 100;
   s.nested.string = "hello tommy (modified)";
   s.nested.array[1][1] = "one one (modified)";
   s.nested.array[1][2] = "one two (modified)";
</cfscript>
<hr>
<b>Modified Original Values</b><br>
<cfoutput>
   // Simple values <br>
   s.nested.number = #s.nested.number#<br>
   s.nested.string = #s.nested.string#<br>

   // Array value <br>
   s.nested.array[1][1] = #s.nested.array[1][1]#<br>
   s.nested.array[1][2] = #s.nested.array[1][2]#<br>
</cfoutput>
```

```
<hr>
<b>Copied structure values should reflect changes to original.</b><br>
<cfoutput>
   // Simple values <br>
   copied.nested.number = #copied.nested.number#<br>
   copied.nested.string = #copied.nested.string#<br>
   // Array values <br>
   copied.nested.array[1][1] = #copied.nested.array[1][1]#<br>
   copied.nested.array[1][2] = #copied.nested.array[1][2]#<br>
</cfoutput>

<hr>
<b>Duplicated structure values should remain unchanged.</b><br>
<cfoutput>
   // Simple values <br>
   duplicated.nested.number = #duplicated.nested.number#<br>
   duplicated.nested.string = #duplicated.nested.string#<br>
   // Array value <br>
   duplicated.nested.array[1][1] = #duplicated.nested.array[1][1]#<br>
   duplicated.nested.array[1][2] = #duplicated.nested.array[1][2]#<br>
</cfoutput>
```

StructCount

Description: Counts the keys in a structure.

Return value: A number; if *structure* does not exist, throws an exception.

Category: Structure functions

Syntax: `StructCount(structure)`

See also: Structure functions

History: New in ColdFusion MX: this function can be used on XML objects.

Parameters

Parameter	Description
structure	Structure to access

Example

```
<!--- This view-only example shows use of StructCount. --->
<p>This file is similar to addemployee.cfm, which is called by
          StructNew, StructClear, and StructDelete. To test, copy
          StructCount function to appropriate place in addemployee.cfm.
<!---
<cfswitch expression = "#ThisTag.ExecutionMode#">
  <cfcase value = "start">
   <cfif StructIsEmpty(attributes.EMPINFO)>
    <cfoutput>Error. No employee data was passed.</cfoutput>
     <cfexit method = "ExitTag">
   <cfelse>
    <cfquery name = "AddEmployee" datasource = "cfsnippets">
     INSERT INTO Employees
       (FirstName, LastName, Email, Phone, Department)
      VALUES
       <cfoutput>
        (
         '#StructFind(attributes.EMPINFO, "firstname")#' ,
         '#StructFind(attributes.EMPINFO, "lastname")#' ,
         '#StructFind(attributes.EMPINFO, "email")#' ,
         '#StructFind(attributes.EMPINFO, "phone")#' ,
         '#StructFind(attributes.EMPINFO, "department")#'
         )
       </cfoutput>
     </cfquery>
   </cfif>
   <cfoutput><hr>Employee Add Complete
     <p>#StructCount(attributes.EMPINFO)# columns added.</cfoutput>
  </cfcase>
</cfswitch> --->
```

StructDelete

Description Removes an element from a structure.

Return value A structure, after removing the element.

Category Structure functions

Syntax `StructDelete(`*structure*, *key* [, *indicatenotexisting*])

See also Structure functions

History New in ColdFusion MX: this function can be used on XML objects.

Parameters

Parameter	Description
structure	Structure or a variable that contains one. Contains element to remove
key	Element to remove
indicatenotexisting	• True: returns Yes if *key* exists; No if it does not. • False: returns Yes regardless of whether *key* exists. Default.

Example

```
<h3>StructDelete Function</h3>
<p>This example uses the StructInsert and StructDelete functions.
<!--- Establish parms for first time through --->
<cfparam name = "firstname" default = "Mary">
<cfparam name = "lastname" default = "Sante">
<cfparam name = "email" default = "msante@allaire.com">
<cfparam name = "phone" default = "777-777-7777">
<cfparam name = "department" default = "Documentation">

 <cfif IsDefined("FORM.Delete")>
 <cfoutput>
 Field to be deleted: #form.field#
 </cfoutput>
 <p>
  <CFScript>
   employee = StructNew();
   StructInsert(employee, "firstname", firstname);
   StructInsert(employee, "lastname", lastname);
   StructInsert(employee, "email", email);
   StructInsert(employee, "phone", phone);
   StructInsert(employee, "department", department);
  </CFScript>
   <cfoutput>
   employee is a structure: #IsStruct(employee)#
   </cfoutput>
   <cfset rc = StructDelete(employee, "#form.field#", "True")>
   <cfoutput>
   <p>Did I delete the field "#form.field#"? The code indicates: #rc#
   </p>
   </cfoutput>
</cfif>
```

```
<cfif NOT IsDefined("FORM.Delete")>
<form action = "structdelete.cfm">
     <p>Select the field to be deleted: 
     <select name = "field">
     <option value = "firstname">first name
     <option value = "lastname">last name
     <option value = "email">email
     <option value = "phone">phone
     <option value = "department">department
     </select>
     <input type = "submit" name = "Delete" value = "Delete">
  </form>
</cfif>
```

StructFind

Description: Determines the value associated with a key in a structure.

Return value: The value associated with a key in a structure; if *structure* does not exist, throws an exception.

Category: Structure functions

Syntax: `StructFind(structure, key)`

See also: Structure functions

Parameters

Parameter	Description
structure	Structure that contains the value to return
key	Key whose value to return

Usage: A structure's keys are unordered.

Example:

```
<!--- This view-only example shows the use of StructFind. --->
<p>This file is identical to addemployee.cfm, which is called by StructNew,
          StructClear, and StructDelete. It adds employees. Employee information
          is passed through the employee structure (EMPINFO attribute). In UNIX,
          you must also add the Emp_ID.
<!---
<cfswitch expression = "#ThisTag.ExecutionMode#">
  <cfcase value = "start">
   <cfif StructIsEmpty(attributes.EMPINFO)>
    <cfoutput>Error. No employee data was passed.</cfoutput>
     <cfexit method = "ExitTag">
   <cfelse>
     <cfquery name = "AddEmployee" datasource = "cfsnippets">
     INSERT INTO Employees (FirstName, LastName, Email, Phone, Department)
      VALUES
       <cfoutput>
        (
         '#StructFind(attributes.EMPINFO, "firstname")#' ,
         '#StructFind(attributes.EMPINFO, "lastname")#' ,
         '#StructFind(attributes.EMPINFO, "email")#' ,
         '#StructFind(attributes.EMPINFO, "phone")#' ,
         '#StructFind(attributes.EMPINFO, "department")#' )
       </cfoutput>
    </cfquery>
   </cfif>
   <cfoutput><hr>Employee Add Complete</cfoutput>
  </cfcase>
</cfswitch> --->
```

StructFindKey

Description: Searches recursively through a substructure of nested arrays, structures, and other elements, for structures whose values match the search key in the *value* parameter.

Return value: An array that contains structures with values that match *value*.

Category: Structure functions

Syntax: `StructFindKey(top, value, scope)`

See also: Structure functions

Parameters

Parameter	Description
top	ColdFusion object (structure or array) from which to start search. This attribute requires an object, not a name of an object.
value	String or a variable that contains one for which to search.
scope	• one: returns one matching key. Default. • all: returns all matching keys

Usage: Returns an array that includes one structure for each of the specified values it finds. The fields of each of these structures are:

- `Value`: value held in the found key
- `Path`: string that can be used to reach the found key
- `Owner`: parent object that contains the found key

A structure's keys are unordered.

Example:

```
<cfset aResults = StructFindKey( #request#, "bass" )>
```

StructFindValue

Description: Searches recursively through a substructure of nested arrays, structures, and other elements for structures with values that match the search key in the `value` parameter.

Return value: An array that contains structures with values that match the search key *value*. If none are found, returns an array of size 0.

Category: Structure functions

Syntax: `StructFindValue( top, value [, scope])`

See also: Structure functions

Parameters

Parameter	Description
top	ColdFusion object (a structure or an array) from which to start search. This attribute requires an object, not a name of an object.
value	String or a variable that contains one for which to search. The type must be a simple object. Arrays and structures are not supported.
scope	• one: function returns one matching key (default) • all: function returns all matching keys

Usage: The fields of each structure in the returned array are:

- `Key`: name of the key in which the value was found
- `Path`: string which could be used to reach the found key
- `Owner`: parent object that contains the found key

A structure's keys are unordered.

Example:

```
<cfset aResults = StructFindValue( #request#, "235" )>
```

StructGet

Description: Gets a structure(s) from a specified path.

Return value: An alias to the variable in the `PathDesired` parameter. If necessary, `StructGet` creates structures or arrays to make *PathDesired* a valid variable "path."

Category: Structure functions

Syntax: `StructGet(pathDesired)`

See also: Structure functions

History: New in ColdFusion MX: this function can be used on XML objects.

New in ColdFusion MX: if there is no structure or array present in `pathDesired`, this function creates structures or arrays to make *PathDesired* a valid variable "path."

Parameters

Parameter	Description
pathDesired	Pathname of variable that contains structure or array from which ColdFusion retrieves structure.

Usage: You can inadvertently create invalid structures using this function. For example, if array notation is used to expand an existing array, the specified new element is created, regardless of the type currently held in the array.

Example:

```
<!--- GetStruct() test --->
<cfset test = StructGet( "dog.myscope.test" )>
<cfset test.foo = 1>
<cfif NOT IsDefined("dog")>
   Dog is not defined<br>
</cfif>
<cfif NOT IsDefined("dog.myscope")>
   Dog.Myscope is not definded<br>
</cfif>
<cfif NOT Isdefined("dog.myscope.test")>
   Dog.Myscope.Test is not defined<br>
</cfif>
<cfif NOT Isdefined("dog.myscope.test.foo")>
   Dog.Myscope.Test.Foo is not defined<br>
</cfif>
<cfoutput>
   #dog.myscope.test.foo#<br>
</cfoutput>
<cfset test = StructGet( "request.myscope[1].test" )>
<cfset test.foo = 2>
<cfoutput>
   #request.myscope[1].test.foo#<br>
</cfoutput>
<cfset test = StructGet( "request.myscope[1].test[2]" )>
<cfset test.foo = 3>
```

```
<cfoutput>
   #request.myscope[1].test[2].foo#<br>
</cfoutput>
```

StructInsert

Description: Inserts a key-value pair into a structure.

Return value: True, upon successful completion. If `structure` does not exist, or if `key` exists and `allowoverwrite = "False"`, ColdFusion throws an exception.

Category: Structure functions

Syntax: `StructInsert(structure, key, value [, allowoverwrite ])`

See also: Structure functions

History: New in ColdFusion MX: this function can be used on XML objects.

Parameters:

Parameter	Description
structure	Structure to contain the new key-value pair.
key	Key that contains the inserted value.
value	Value to add.
allowoverwrite	Optional. Whether to allow overwriting a key. Default: False.

Usage: A structure's keys are unordered.

Example:

```
<h1>Add New Employees</h1>
<!--- Establish parms for first time through --->
<cfparam name = "FORM.firstname" default = "">
<cfparam name = "FORM.lastname" default = "">
<cfparam name = "FORM.email" default = "">
<cfparam name = "FORM.phone" default = "">
<cfparam name = "FORM.department" default = "">

<cfif FORM.firstname EQ "">
 <p>Please fill out the form.
<cfelse>
 <cfoutput>
  <CFScript>
   employee = StructNew();
   StructInsert(employee, "firstname", FORM.firstname);
   StructInsert(employee, "lastname", FORM.lastname);
   StructInsert(employee, "email", FORM.email);
   StructInsert(employee, "phone", FORM.phone);
   StructInsert(employee, "department", FORM.department);
 </CFScript>

 <p>First name is #StructFind(employee, "firstname")#</p>
 <p>Last name is #StructFind(employee, "lastname")#</p>
 <p>EMail is #StructFind(employee, "email")#</p>
 <p>Phone is #StructFind(employee, "phone")#</p>
 <p>Department is #StructFind(employee, "department")#</p>
 </cfoutput>
```

```
 <!--- Call the custom tag that adds employees --->
 <CF_ADDEMPLOYEE EMPINFO = "#employee#">
</cfif>

<Hr>
<form action = "structinsert.cfm">
   <p>First Name: 
   <input name = "firstname" type = "text" hspace = "30" maxlength = "30">
   <p>Last Name: 
   <input name = "lastname" type = "text" hspace = "30" maxlength = "30">
   <p>EMail: 
   <input name = "email" type = "text" hspace = "30" maxlength = "30">
   <p>Phone: 
   <input name = "phone" type = "text" hspace = "20" maxlength = "20">
   <p>Department: 
   <input name = "department" type = "text" hspace = "30" maxlength = "30">
   <p>
   <input type = "submit" value = "OK">
</form>
```

StructIsEmpty

Description: Determines whether a structure contains data.

Return value: True, if *structure* is empty; if *structure* does not exist, ColdFusion throws an exception.

Category: Decision functions, Structure functions

Syntax: `StructIsEmpty(structure)`

See also: Structure functions

History: New in ColdFusion MX: this function can be used on XML objects.

Parameters

Parameter	Description
structure	Structure to test

Example

```
<!--- This example illustrates use of StructIsEmpty. --->
<p>This file is identical to addemployee.cfm, which is called by StructNew,
         StructClear, and StructDelete. It adds employees. Employee information
         is passed through employee structure (EMPINFO attribute). In UNIX, you
         must also add the Emp_ID.
<cfswitch expression = "#ThisTag.ExecutionMode#">
  <cfcase value = "start">
   <cfif StructIsEmpty(attributes.EMPINFO)>
    <cfoutput>Error. No employee data was passed.</cfoutput>
     <cfexit method = "ExitTag">
   <cfelse>
    <!--- Add the employee; In UNIX, you must also add the Emp_ID --->
     <cfquery name = "AddEmployee" datasource = "cfsnippets">
     INSERT INTO Employees
       (FirstName, LastName, Email, Phone, Department)
      VALUES
       <cfoutput>
        (
          '#StructFind(attributes.EMPINFO, "firstname")#' ,
          '#StructFind(attributes.EMPINFO, "lastname")#' ,
          '#StructFind(attributes.EMPINFO, "email")#' ,
          '#StructFind(attributes.EMPINFO, "phone")#' ,
          '#StructFind(attributes.EMPINFO, "department")#'
          )
       </cfoutput>
    </cfquery>
   </cfif>
   <cfoutput><hr>Employee Add Complete</cfoutput>
  </cfcase>
</cfswitch>
```

StructKeyArray

Description Finds the keys in a ColdFusion structure.

Return value An array of keys; if *structure* does not exist, ColdFusion throws an exception.

Category Structure functions

Syntax `StructKeyArray(`*structure*`)`

See also Structure functions

Parameters

Parameter	Description
structure	Structure from which to extract a list of keys

Usage A structure's keys are unordered.

Example

```
<!--- Shows StructKeyArray function to copy keys from a structure to an array.
         Uses StructNew to create structure and fills its fields with the
         information the user enters in the form fields. --->
<h3>StructKeyArray Example</h3>
<h3>Extracting the Keys from the Employee Structure</h3>
<!-- Create structure. Check whether Submit was pressed. If so, define fields
         in employee structure with user entries on form. ----->
<cfset employee = StructNew()>
<cfif Isdefined("Form.Submit")>
   <cfif Form.Submit is "OK">
      <cfset employee.firstname = FORM.firstname>
      <cfset employee.lastname = FORM.lastname>
      <cfset employee.email = FORM.email>
      <cfset employee.phone = FORM.phone>
      <cfset employee.company = FORM.company>
   <cfelseIf Form.Submit is "Clear">
      <cfset rc = StructClear(employee)>
   </cfif>
</cfif>
<p> This example uses the StructNew function to create a structure called
         "employee" that supplies employee info. Its fields are filled by
         the form. After you enter employee information in structure, the
         example uses StructKeyArray function to copy all of the keys from
         the structure into an array. </p>
<hr size = "2" color = "#0000A0">
<form action = "structkeyarray.cfm">
<table cellspacing = "2" cellpadding = "2" border = "0">
   <tr>
   <td>First Name:</td>
   <td><input name = "firstname" type = "text"
      value = "" hspace = "30" maxlength = "30"></td>
   </tr>
   <tr>
   <td>Last Name:</td>
   <td><input name = "lastname" type = "text"
      value = "" hspace = "30" maxlength = "30"></td>
```

```
    </tr>
    <tr>
    <td>EMail</td>
    <td><input name = "email" type = "text"
      value = "" hspace = "30" maxlength = "30"></td>
    </tr>
    <tr>
    <td>Phone:</td>
    <td><input name = "phone" type = "text"
      value = "" hspace = "20" maxlength = "20"></td>
    </tr>
    <tr>
    <td>Company:</td>
    <td><input name = "company" type = "text"
      value = "" hspace = "30" maxlength = "30"></td>
    </tr>
    <tr>
    <td><input type = "submit" name = "submit"
      value = "OK"></td>
    <td><b>After you submit the FORM, scroll down to see the array.</b>
    </td>
    </tr>
</table>
</form>
<cfif NOT StructISEmpty(employee)>
    <hr size = "2" color = "#0000A0">
    <cfset keysToStruct = StructKeyArray(employee)>
    <cfloop index = "i" from = "1" to = "#ArrayLen(keysToStruct)#">
      <p><cfoutput>Key#i# is #keysToStruct[i]#</cfoutput></p>
      <p><cfoutput>Value#i# is #employee[keysToStruct[i]]#</cfoutput>
      </p>
    </cfloop>
</cfif>
```

StructKeyExists

Description: Determines whether a specific key is present in a structure.

Return value: True, if *key* is in *structure*; if *structure* does not exist, ColdFusion throws an exception.

Category: Decision functions, Structure functions

Syntax: `StructKeyExists(structure, "key")`

See also: Structure functions

Parameters:

Parameter	Description
structure	Name of structure to test
key	Key to test

Usage: This function can sometimes be used in place of the `IsDefined` function, when working with the URL and Form scopes, which are structures. The following pieces of code are equivalent:

```
cfif IsDefined("Form.JediMaster")>
<cfif StructKeyExists(Form,"JediMaster")>
```

A structure's keys are unordered.

Example:

```
<!--- This example shows the use of StructKeyExists. --->
<p>This file is similar to addemployee.cfm, which is called by StructNew,
          StructClear, and StructDelete. To test, copy the &LT;CFELSEif&GT;
          statement to the appropriate place in addemployee.cfm. It is a custom tag
          to add employees. Employee information is passed through the employee
          structure (the EMPINFO attribute). In UNIX, you must also add the Emp_ID.

<cfswitch expression = "#ThisTag.ExecutionMode#">
  <cfcase value = "start">
   <cfif StructIsEmpty(attributes.EMPINFO)>
    <cfoutput>Error. No employee data was passed.</cfoutput>
     <cfexit method = "ExitTag">
   <cfelseIf NOT StructKeyExists(attributes.EMPINFO, "department")>
    <cfscript>StructUpdate(attributes.EMPINFO, "department",
      "Unassigned");
     </cfscript>
    <cfexit method = "ExitTag">
        <cfelse>
```

StructKeyList

Description Extracts keys from a ColdFusion structure.

Return value A list of keys; if *structure* does not exist, ColdFusion throws an exception.

Category Structure functions

Syntax `StructKeyList(`*structure* `[, `*delimiter*`])`

See also Structure functions

Parameters

Parameter	Description
structure	Structure from which to extract a list of keys.
delimiter	Optional. Character that separates keys in list. Default: comma.

Usage A structure's keys are unordered.

Example

```
<!--- This example shows how to use StructKeyList to list the keys
         in a structure. It uses StructNew function to create structure
         and fills it with information user enters in form fields. --->
<!--- This section creates structure and checks whether Submit has been pressed.
         If so, code defines fields in the employee structure with what the
         user entered in the form. --->
<cfset employee = StructNew()>
<cfif Isdefined("Form.Submit")>
   <cfif Form.Submit is "OK">
      <cfset employee.firstname = FORM.firstname>
      <cfset employee.lastname = FORM.lastname>
      <cfset employee.email = FORM.email>
      <cfset employee.phone = FORM.phone>
      <cfset employee.company = FORM.company>
   <cfelseIf Form.Submit is "Clear">
      <cfset rc = StructClear(employee)>
   </cfif>
</cfif>
<html>
<head>
   <title>StructKeyList Function</title>
</head>
<body>
<h3>StructKeyList Function</h3>
<h3>Listing the Keys in the Employees Structure</h3>
<p>This example uses StructNew function to create structure "employee" that
         supplies employee information. The fields are filled with the
         contents of the following form.</p>
<p>After you enter employee information into structure, example uses
         <b>StructKeyList</b> function to list keys in structure.</p>
<p>This code does not show how to insert information into a database.
         See cfquery for more information about database insertion.
<hr size = "2" color = "#0000A0">
<form action = "structkeylist.cfm">
```

```
<table cellspacing = "2" cellpadding = "2" border = "0">
   <tr>
   <td>First Name:</td>
   <td><input name = "firstname" type = "text"
      value = "" hspace = "30" maxlength = "30"></td>
   </tr>
   <tr>
   <td>Last Name:</td>
   <td><input name = "lastname" type = "text"
      value = "" hspace = "30" maxlength = "30"></td>
   </tr>
   <tr>
   <td>EMail</td>
   <td><input name = "email" type = "text"
      value = "" hspace = "30" maxlength = "30"></td>
   </tr>
   <tr>
   <td>Phone:</td>
   <td><input name = "phone" type = "text"
      value = "" hspace = "20" maxlength = "20"></td>
   </tr>
   <tr>
   <td>Company:</td>
   <td><input name = "company" type = "text"
      value = "" hspace = "30" maxlength = "30"></td>
   </tr>
   <tr>
   <td><input type = "submit" name = "submit" value = "OK"></td>
   <td><b>After you submit form, scroll down to see the list.</b></td>
   </tr>
</table>
</form>
<cfif NOT StructISEmpty(employee)>
   <hr size = "2" color = "#0000A0">
   <cfset keysToStruct = StructKeyList(employee,"<li>")>
   <p>Here are the keys to the structure:</p>
   <ul>
   <li><cfoutput>#keysToStruct#</cfoutput>
   </ul>
   <p>If fields are correct, we can process new employee information.
          If they are not correct, consider rewriting application.</p>
</cfif>
```

StructNew

Description: Creates a structure.

Return value: A structure.

Category: Structure functions

Syntax: `StructNew()`

See also: Structure functions

Parameters: None

Example:

```
<!--- Shows StructNew. Calls CF_ADDEMPLOYEE, which uses the |
         addemployee.cfm file to add employee record to database. --->
<h1>Add New Employees</h1>
<cfparam name = "FORM.firstname" default = "">
<cfparam name = "FORM.lastname" default = "">
<cfparam name = "FORM.email" default = "">
<cfparam name = "FORM.phone" default = "">
<cfparam name = "FORM.department" default = "">
<cfif FORM.firstname EQ "">
 <p>Please fill out the form.
<cfelse>
   <cfoutput>
  <cfscript>
   employee = StructNew();
   StructInsert(employee, "firstname", FORM.firstname);
   StructInsert(employee, "lastname", FORM.lastname);
   StructInsert(employee, "email", FORM.email);
   StructInsert(employee, "phone", FORM.phone);
   StructInsert(employee, "department", FORM.department);
 </cfscript>
   <p>First name is #StructFind(employee, "firstname")#
   <p>Last name is #StructFind(employee, "lastname")#
   <p>EMail is #StructFind(employee, "email")#
   <p>Phone is #StructFind(employee, "phone")#
   <p>Department is #StructFind(employee, "department")#
   </cfoutput>
<!--- Call the custom tag that adds employees --->
 <CF_ADDEMPLOYEE EMPINFO = "#employee#">
</cfif>
```

StructSort

Description: Finds and sorts structures that contain top-level key names (strings).

Return value: An array of structures of top-level key names (strings), sorted by the value of the specified subelement. The key values may be simple values or complex elements.

Category: Structure functions

Syntax: `StructSort(base, sortType, sortOrder, pathToSubElement)`

See also: Structure functions

Parameters

Parameter	Description
base	A ColdFusion struct with one field (an associative array).
sorttype	• numeric • text: case sensitive (all lower-case letters precede the first upper-case letter). Default. • textnocase
sortorder	• asc: ascending (a to z) sort order. Default. • desc: descending (z to a) sort order
pathToSubelement	String or a variable that contains one. Path to apply to each top-level key, to reach element value by which to sort. Default: nothing (top-level entries sorted by their own values).

Usage: The `pathToSubElement` string does not support array notation; only substructures of structures are supported.

Example:

```
<cfscript>
   salaries = StructNew() ;
   employees = StructNew() ;
   departments = StructNew() ;
   for ( i=1; i lt 6; i=i+1 )
   {
      salary = 120000 - i*10000 ;
      salaries["employee#i#"] = salary ;

      employee = StructNew() ;
      employee["salary"] = salary ;
      // employee.salary = salary ;
      employees["employee#i#"] = employee ;

      departments["department#i#"] = StructNew() ;
      departments["department#i#"].boss = employee ;
   }
</cfscript>
```

```
<cfoutput>
<p>list of employees based on the salary (text search): <br>
1) #ArrayToList( StructSort( salaries ) )#<br>
2) #ArrayToList( StructSort( salaries, "text", "ASC" ) )#<br>
3) #ArrayToList( StructSort( salaries, "textnocase", "ASC" ) )#<br>
4) #ArrayToList( StructSort( salaries, "text", "DESC" ) )#<br>
<p>list of employees based on the salary (numeric search): <br>
5) #ArrayToList( StructSort( salaries, "numeric", "ASC" ) )#<br>
6) #ArrayToList( StructSort( salaries, "numeric", "DESC" ) )#<br>
<p>list of employees based on the salary (subfield search): <br>
7) #ArrayToList( StructSort( employees, "numeric", "DESC", "salary" ) )#<br>
8) #ArrayToList( StructSort( employees, "text", "ASC",  "salary" ) )#<br>
<p>list of departments based on the salary (sub-sub-field search): <br>
9) #ArrayToList( StructSort( departments, "text", "ASC", "boss.salary" ) )#<br>
</cfoutput>

<!--- add an invalid element and test that it throws an error --->
<p><p>
<cfset employees[ "employee4" ] = StructNew()>
<cftry>
   <cfset temp = StructSort( employees, "text", "ASC", "salary" )>
   <cfoutput>We have a problem - this was supposed to throw an exception!<br>
          </cfoutput>
<cfcatch type="any">
   <cfoutput>
      ERROR: <b>This error was expected!</b><br>
      #cfcatch.message# - #cfcatch.detail#<br>
   </cfoutput>
</cfcatch>
</cftry>
```

StructUpdate

Description Updates a key with a value.

Return value True, on successful execution; if the structure does not exist, ColdFusion throws an error.

Category Structure functions

Syntax `StructUpdate(structure, key, value)`

See also Structure functions

History New in ColdFusion MX: this function can be used on XML objects.

Parameters

Parameter	Description
structure	Structure to update
key	Key, the value of which to update
value	New value

Example

```
<!--- This example shows the use of StructUpdate. --->
<p>This file is similar to addemployee.cfm, which is called by StructNew,
          StructClear, and StructDelete. To test this file, copy the
          &LT;CFELSEIF&GT; statement to the appropriate place in
          addemployee.cfm. It is an example of a custom tag used to add
          employees. Employee information is passed through the employee
          structure (the EMPINFO attribute). In UNIX, you must also add the Emp_ID.

<cfswitch expression = "#ThisTag.ExecutionMode#">
  <cfcase value = "start">
   <cfif StructIsEmpty(attributes.EMPINFO)>
    <cfoutput>Error. No employee data was passed.</cfoutput>
     <cfexit method = "ExitTag">
   <cfelseIf StructFind(attributes.EMPINFO, "department") EQ "">
    <cfscript>
     StructUpdate(attributes.EMPINFO, "department", "Unassigned");
    </cfscript>
    <cfexit method = "ExitTag">
        <cfelse>
```

Tan

Description Calculates the tangent of an angle.

Return value A number; the tangent of an angle.

Category Mathematical functions

Syntax `Tan(number)`

See also `Atn`, `ASin`, `Cos`, `Sin`, `Pi`

Parameters

Parameter	Description
number	Angle, in radians. To convert an angle from degrees to radians, use the `PI` function, and multiply the degrees by PI/180.

Example

```
<h3>Tan Example</h3>
<p>Returns the tangent of an angle.
<p>Tan(1): <cfoutput>#Tan(1)#</cfoutput>
<p>Tan(Pi()/4): <cfoutput>#Tan(Pi()/4)#</cfoutput>
```

TimeFormat

Description: Formats a time value.

Return value: A custom-formatted time value. If no mask is specified, returns a time value using the `hh:mm tt` format. For international time formatting, see `LSTimeFormat`.

Category: Date and time functions, Display and formatting functions

Syntax: `TimeFormat(time [, mask ])`

See also: `CreateTime`, `Now`, `ParseDateTime`

History: New in ColdFusion MX: This function processes extra characters within the `mask` value differently than in earlier releases, as follows:

- ColdFusion 5 and earlier: the function returns the time format and an apostrophe-delimited list of the extra characters. For example, `TimeFormat(now(), "hh:mm:ss dog")` returns `8:17:23 d'o'g`.
- ColdFusion MX: the function returns the time format and the extra characters. For example, for the call above, it returns `8:17:23 dog`.

If the extra characters are single-quoted (for example, `hh:mm:ss 'dog'`), ColdFusion 5 and ColdFusion MX return the time format and the extra characters: `8:17:23 dog`.

New in ColdFusion MX: This function supports the `short`, `medium`, `long`, and `full` `mask` attribute options.

Parameters

Parameter	Description
time	A date/time value or string to convert
mask	Masking characters that determine the format: • `h`: Hours; no leading zero for single-digit hours (12-hour clock) • `hh`: Hours; leading zero for single-digit hours (12-hour clock) • `H`: Hours; no leading zero for single-digit hours (24-hour clock) • `HH`: Hours; leading zero for single-digit hours (24-hour clock) • `m`: Minutes; no leading zero for single-digit minutes • `mm`: Minutes; a leading zero for single-digit minutes • `s`: Seconds; no leading zero for single-digit seconds • `ss`: Seconds; leading zero for single-digit seconds • `t`: One-character time marker string, such as A or P • `tt`: Multiple-character time marker string, such as AM or PM • short • medium • long • full

Usage

When passing a date/time value as a string, you must enclose it in quotation marks. Otherwise, it is interpreted as a number representation of a date/time object.

Database query results for date and time values can vary in sequence and formatting unless you use functions to format the results. To ensure that dates and times display with appropriate formatting, and that users of your ColdFusion application are not confused by dates and times displayed, Macromedia recommends that you use the `DateFormat` and `TimeFormat` functions to format date and time values from queries. For more information and examples, see TechNote 22183, "*ColdFusion Server (5 and 4.5.x) with Oracle: Formatting Date and Time Query Results*," at http://www.coldfusion.com/Support/KnowledgeBase/SearchForm.cfm.

Example

```
<cfset todayDate = #Now()#>
<body>
<h3>TimeFormat Example</h3>
<p>Today's date is <cfoutput>#todayDate#</cfoutput>.
<p>Using Timeformat, we can display the value in different ways:
<cfoutput>
<ul>
   <li>#TimeFormat(todayDate)#
   <li>#TimeFormat(todayDate, "hh:mm:ss")#
   <li>#TimeFormat(todayDate, "hh:mm:sst")#
   <li>#TimeFormat(todayDate, "hh:mm:sstt")#
   <li>#TimeFormat(todayDate, "HH:mm:ss")#
</ul>
</cfoutput>
</body>
```

ToBase64

Description: Calculates the Base64 representation of a string or binary object. The Base64 format uses printable characters, allowing binary data to be sent in forms and e-mail, and stored in a database or file.

Return value: The Base64 representation of a string or binary object.

Category: Conversion functions, Other functions, String functions

Syntax: `ToBase64(string or binary_object[, encoding])`

See also:
- `cffile` for information about loading and reading binary data
- `cfwddx` for information about serializing and deserializing binary data
- `IsBinary` and `ToBinary` for checking for binary data and converting a Base64 object to binary format

History: New in ColdFusion MX: the `encoding` attribute is new.

Parameters:

Parameter	Description
string or binary_object	A string, the name of a string, or a binary object.
encoding	For a string, defines how characters are represented in a byte array: • US-ASCII • ISO-8859-1 • UTF-8 • UTF-16 Default: the encoding option of the page on which the function is called. See cfcontent on page 66. The Java platform determines the available options. For a binary object, this parameter is ignored.

Usage: Base64 provides -bit encoding of 8-bit ASCII characters. High ASCII values and binary objects are not safe for transport over internet protocols such as HTTP and SMTP; using Base64 safely sends ASCII and binary data over these protocols.

Base64 lets you store binary objects in a database.

Note: To reverse Base64 encoding of a string, you can convert it to a binary object, then convert the binary object to a string, using the `toString` function.

Example:

```
<h3>ToBase64 Example</h3>
<!--- Initialize data. ---->
<cfset charData = "">
<!--- Create string of ASCII characters (32-255); concatenate them --->
<cfloop index = "data" from = "32" to = "255">
   <cfset ch = chr(data)>
   <cfset charData = charData & ch>
</cfloop>
```

```
<p>
The following string is the concatenation of all characters (32 to 255)
         from the ASCII table.<br>
<cfoutput>#charData#</cfoutput>
</p>
<!--- Create a Base64 representation of this string. ---->
<cfset data64 = toBase64(charData)>

<!----- Convert string to binary. ------->
<cfset binaryData = toBinary(data64)>
<!--- Convert binary back to Base64. ---->
<cfset another64 = toBase64(binaryData)>
<!---- Compare another64 with data64 to ensure that they are equal. ---->
<cfif another64 eq data64>
   <h3>Base64 representations are identical.</h3>
<cfelse>
   <h3>Conversion error.</h3>
</cfif>
```

ToBinary

Description: Calculates the binary representation of Base64-encoded data.

Return value: The binary representation of Base64-encoded data.

Category: Conversion functions, Other functions, String functions

Syntax: `ToBinary(string_in_Base64 or binary_value)`

See also:
- `cffile` for information about loading and reading binary data
- `cfwddx` for information about serializing and deserializing binary data
- `IsBinary` and `ToBase64` for checking format and converting to Base64
- `Len` for determining the length of a binary object

Parameters

Parameter	Description
string_in_Base64 or binary_value	A string or a variable that contains one: • In Base64 format to convert to binary • In binary format to test whether it is valid

Usage: Base64 provides 6-bit encoding of 8-bit ASCII characters. From Base64 data, you can recreate the binary object that it represents, such as a GIF, JPG, or executable file.

Example:

```
<h3>ToBinary Example</h3>
<!---- Initialize data. ---->
<cfset charData = "">
<!---- Create a string of ASCII characters (32-255); concatenate them. ---->
<cfloop index = "data" from = "32" to = "255">
   <cfset ch = chr(data)>
   <cfset charData = charData & ch>
</cfloop>
<p>The following string is the concatenation of all characters (32 to 255)
         from the ASCII table.<br>
<cfoutput>#charData#</cfoutput></p>
<!----- Create a Base64 representation of this string. ----->
<cfset data64 = toBase64(charData)>

<!--- Convert string to binary. ---->
<cfset binaryData = toBinary(data64)>
<!--- Convert binary back to Base64. --->
<cfset another64 = toBase64(binaryData)>
<!---- Compare another64 with data64 to ensure that they are equal. ---->
<cfif another64 eq data64>
   <h3>Base64 representation of binary data is identical to the Base64
         representation of string data.</h3>
<cfelse>
   <h3>Conversion error.</h3>
</cfif>
```

ToString

Description: Converts a value to a string.

Return value: A string.

Category: Conversion functions, Other functions, String functions

Syntax: `ToString(any_value[, encoding])`

See also: ToBase64, ToBinary

History: New in ColdFusion MX: ColdFusion supports the Java UCS-2 representation of Unicode character values 0–65535. (ColdFusion 5 and earlier releases supported ASCII values 1–255.)

New in ColdFusion MX: the `encoding` attribute is new.

Parameters

Parameter	Description
any_value	Value to convert to a string
encoding	Defines how characters are represented in a string: • US-ASCII • ISO-8859-1 • UTF-8 • UTF-16 Default: the encoding option of the page on which the function is called. See cfcontent on page 66. The Java platform determines the available options.

Usage: This function can convert simple values and binary values that do not contain Byte zero. If this function cannot convert a value, it throws an exception. This function can convert an XML document object to a string representation.

Note: You can use this function to reverse Base64 encoding of a string. Convert the Base64 encoded object to a binary object, then use this function to convert the binary object to a string.

Example:

```
<h3>ToString Example</h3>
<!---- Initialize data. ------>
<cfset charData = "">
<!----- Create string of ASCII characters (32-255) and concatenate them. ---->
<cfloop index = "data" from = "32" to = "255">
   <cfset ch = chr(data)>
   <cfset charData = charData & ch>
</cfloop>
<p>The following string is the concatenation of characters (32 to 255)
         from the ASCII table.<br>
<cfoutput>#charData#</cfoutput></p>

<!------ Create a Base64 representation of this string. ---->
<cfset data64 = toBase64(#charData#)>
```

```
<p>
The following string is the Base64 representation of the string.<br>
<cfoutput>#data64#</cfoutput></p>
<!---- Create a binary representation of Base64 data. --->
<cfset dataBinary = toBinary(data64)>

<!---- Create the string repesentation of the binary data. ----->
<cfset dataString = toString(dataBinary)>
<p>The following is the string representation of the binary data.<br>
<cfoutput>#dataString#</cfoutput></p>
```

Trim

Description Removes leading and trailing spaces from a string.

Return value A copy of *string*, after removing leading and trailing spaces.

Category String functions

Syntax `Trim(string)`

See also LTrim, RTrim

Parameters

Parameter	Description
string	A string or a variable that contains one

Example

```
<h3>Trim Example</h3>
<cfif IsDefined("FORM.myText")>
   <cfoutput>
      <pre>
         Your string:"#FORM.myText#"
         Your string:"#trim(FORM.myText)#"
         (trimmed on both sides)
      </pre>
   </cfoutput>
</cfif>
<form action = "trim.cfm">
<p>Type in some text, and it will be modified by trim to remove leading
          spaces from the left and right
<p><input type = "Text" name = "myText" value = "   TEST   ">
<p><input type = "Submit" name = "">
</form>
```

UCase

Description: Converts the alphabetic characters in a string to uppercase.

Return value: A copy of a string, converted to uppercase.

Category: String functions

Syntax: `UCase(string)`

See also: LCase

Parameters

Parameter	Description
string	A string or a variable that contains one

Example

```
<h3>UCase Example</h3>

<cfif IsDefined("FORM.sampleText")>
   <cfif FORM.sampleText is not "">
      <p>Your text, <cfoutput>#FORM.sampleText#</cfoutput>,
          returned in uppercase is <cfoutput>#UCase(FORM.sampleText)#</
          cfoutput>.
   <cfelse>
      <p>Please enter some text.
   </cfif>
</cfif>

<form action = "ucase.cfm">
<p>Enter your sample text, and press "submit" to see the text returned in
          uppercase:
<p><input type = "Text" name = "SampleText" value = "sample">

<input type = "Submit" name = "" value = "submit">
</form>
```

URLDecode

Description: Decodes a URL-encoded string.

Return value: A copy of a string, decoded.

Category: Conversion functions, Other functions, String functions

Syntax: `URLDecode(urlEncodedString[, charset])`

See also: `URLEncodedFormat`

History: New in ColdFusion MX: ColdFusion supports the Java UCS-2 representation of Unicode character values 0–65535. (Earlier releases supported ASCII values.)

New in ColdFusion MX: the `charset` parameter is new.

Parameters

Parameter	Description
urlEncodedString	URL-encoded string or a variable that contains one.
charset	A Java character set name. Optional. The following values are typically used: • UTF-8 • ISO-8859-1 • UTF-16 • US-ASCII • UTF-16BE • UTF-16LE For a list of character sets, see: http://www.w3.org/International/O-charset-lang.html

Usage: URL encoding formats some characters with a percent sign and the two-character hexadecimal representation of the character. For example, a character whose code is 129 is encoded as `%81`. A space is encoded with a plus sign.

Query strings in HTTP are always URL-encoded.

Example: This example creates, encodes, and decodes a string that contains ASCII character codes.

```
<cfscript>
   // Build string
   s = "";
   for (c = 1; c lte 256; c = c + 1)
   {
      s = s & chr(c);
   }
   // Encode string and display result
   enc = URLEncodedFormat(s);
   writeOutput("Encoded string is: '#enc#'.<br>");
   // Decode and compare result with original
   dec = URLDecode(enc);
```

```
    if (dec neq s)
    {
      writeOutput("Decoded is not the same as encoded.");
    }
    else
    {
      writeOutput("All's quiet on the Western front.");
    }
</cfscript>
```

URLEncodedFormat

Description: Generates a URL-encoded string. For example, it replaces spaces with `%20`, and non-alphanumeric characters with equivalent hexadecimal escape sequences. Passes arbitrary strings within a URL (ColdFusion automatically decodes URL parameters that are passed to a page).

Return value: A copy of a string, URL-encoded.

Category: Conversion functions, Other functions, String functions

Syntax: `URLEncodedFormat(string [, charset ])`

See also: `URLDecode`

History: New in ColdFusion MX: the `charset` parameter is new.

Parameters

Parameter	Description
string	A string or a variable that contains one
charset	A Java character set name. Optional. The following values are typically used: • UTF-8 • ISO-8859-1 • UTF-16 • US-ASCII • UTF-16BE • UTF-16LE For a list of character sets, see: http://www.w3.org/International/O-charset-lang.html

Usage: URL encoding formats some characters with a percent sign and the two-character hexadecimal representation of the character. For example, a character whose code is 129 is encoded as `%81`. A space is encoded with a plus sign.

Query strings in HTTP are always URL-encoded.

Example:

```
<h3>URLEncodedFormat Example</h3>
<cfif IsDefined("url.myExample")>
   <p>The url variable url.myExample was passed from the previous link ...
   its value is:
   <br><b>"<cfoutput>#url.myExample#</cfoutput>"</b>
</cfif>
<p>This function returns a URL encoded string.
<cfset s = "My url-encoded string has special characters & other stuff">
<p> <A HREF = "urlencodedformat.cfm?myExample=<cfoutput>#URLEncodedFormat(s)#
</cfoutput>">Click me</A>
```

URLSessionFormat

Description

Depending on whether a client computer accepts cookies, this function does the following:

- If the client does not accept cookies: automatically appends all required client identification information to a URL
- If the client accepts cookies: does not append information

This function automatically determines which identifiers are required, and sends only the required information. It provides a more secure and robust method for supporting client identification than manually encoding the information in each URL, because it sends only required information, when it is required, and it is easier to code.

Return value

A URL; if cookies are disabled for the browser, client and session data are appended.

Category

Other functions

Syntax

```
URLSessionFormat(request_URL)
```

Parameters

Parameter	Description
request_URL	URL of a ColdFusion page

Usage

In the following example, the `cfform` tag posts a request to another page and sends the client identification, if required. If cookie support is detected, the function returns the following:

```
myactionpage.cfm
```

If the detected cookie is not turned on, or cookie support cannot be reliably detected, the function return value is as follows:

```
myactionpage.cfm?jsessionid=xxxx;cfid=xxxx&cftoken=xxxxxxxx
```

Example

```
<cfform
        method="Post"
        action="#URLSessionFormat("MyActionPage.cfm")#>
```

Val

Description: Converts numeric characters that occur at the beginning of a string to a number.

Return value: A number. If conversion fails, returns zero.

Category: Conversion functions, String functions

Syntax: `Val(string)`

See also: `IsNumeric`

Parameters

Parameter	Description
string	A string or a variable that contains one

Usage: This function works as follows:

- If `TestValue = "234A56?7'"`, `Val(TestValue)` returns 234
- If `TestValue = "234'5678'9?'"`, `Val(TestValue)` returns 234
- If `TestValue = "BG234"`, `Val(TestValue)` returns the value 0, (not an error)
- If `TestValue = "0"`, `Val(TestValue)` returns the value 0, (not an error)

Example:

```
<h3>Val Example</h3>
<cfif IsDefined("FORM.theTestValue")>
   <cfif Val(FORM.theTestValue) is not 0>
      <h3>The string <cfoutput>#DE(FORM.theTestValue)#</cfoutput>
          can be converted to a number:
      <cfoutput>#Val(FORM.theTestValue)#</cfoutput></h3>
   <cfelse>
      <h3>The beginning of the string <cfoutput>#DE(FORM.theTestValue)#
       </cfoutput> cannot be converted to a number</h3>
   </cfif>
</cfif>
<form action = "val.cfm">
<p>Enter a string, and determine whether its beginning can be evaluated
         to a numeric value.
<p>
<input type = "Text"
   name = "TheTestValue"
   value = "123Boy">
<input type = "Submit"
   value = "Is the beginning numeric?"
   name = "">
</form>
```

ValueList

Description — Inserts a delimiter between each value in an executed query. ColdFusion does not evaluate the arguments.

Return value — A delimited list of the values of each record returned from an executed query.

Category — Other functions, Query functions

Syntax — `ValueList(query.column [, delimiter ])`

See also — `QuotedValueList`

Parameters

Parameter	Description
query.column	Name of an executed query and column. Separate query name and column name with a period.
delimiter	A delimiter character to separate column data items.

Example

```
<h3>ValueList Example</h3>

<!--- use the contents of a query to create another dynamically --->
<cfquery name = "GetDepartments" datasource = "cfsnippets">
   SELECT Dept_ID FROM Departments
   WHERE Dept_ID IN ('BIOL')
</cfquery>

<cfquery name = "GetCourseList" datasource = "cfsnippets">
SELECT *
   FROM CourseList
   WHERE Dept_ID IN ('#ValueList(GetDepartments.Dept_ID)#')
</cfquery>

<cfoutput QUERY = "GetCourseList" >
<pre>#Course_ID##Dept_ID##CorNumber##CorName#</pre>
</cfoutput>
```

Week

Description From a date/time object, determines the week number within the year.

Return value An integer in the range 1–53; the ordinal of the week, within the year.

Category Date and time functions

Syntax `Week(date)`

See also `DatePart`

Parameters

Parameter	Description
date	A date/time object in the range 100 AD–9999 AD. See "How ColdFusion processes two-digit year values" on page 377.

Usage When passing date as a string, enclose it in quotation marks. Otherwise, it is interpreted as a number representation of a date.

Example

```
<h3>Week Example</h3>
<cfif IsDefined("FORM.year")>
More information about your date:
<cfset yourDate = CreateDate(FORM.year, FORM.month, FORM.day)>
<cfoutput>
   <p>Your date, #DateFormat(yourDate)#.
   <br>It is #DayofWeekAsString(DayOfWeek(yourDate))#, day
      #DayOfWeek(yourDate)# in the week.
   <br>This is day #Day(YourDate)# in the month of
      #MonthAsString(Month(yourDate))#, which has #DaysInMonth(yourDate)# days.
   <br>We are in week #Week(yourDate)# of #Year(yourDate)# (day
      #DayofYear(yourDate)# of #DaysinYear(yourDate)#). <br>
   <cfif IsLeapYear(Year(yourDate))>This is a leap year
     <cfelse>This is not a leap year
   </cfif>
</cfoutput>
</cfif>
```

WriteOutput

Description

Appends text to the page-output stream.

This function writes to the page-output stream regardless of conditions established by the `cfsetting` tag.

Category

Other functions

Syntax

`WriteOutput(`*`string`*`)`

Parameters

Parameter	Description
string	A string or a variable that contains one

Usage

Within the `cfquery` and `cfmail` tags, this function does not output to the current page; it writes to the current SQL statement or mail text. Do not use `WriteOutput` within `cfquery` and `cfmail`.

Although you can call this function anywhere within a page, it is most useful inside a `cfscript` block.

Example

```
...
<cfscript>
  employee = StructNew();
  StructInsert(employee, "firstname", FORM.firstname);
  StructInsert(employee, "lastname", FORM.lastname);
  StructInsert(employee, "email", FORM.email);
  StructInsert(employee, "phone", FORM.phone);
  StructInsert(employee, "department", FORM.department);
  WriteOutput("About to add " & FORM.firstname & " " & FORM.lastname);
</cfscript>
```

XmlChildPos

Description: Gets the position of a child element within an XML document object.

Return value: The position, in an XmlChildren array, of the *N*th child that has the specified name.

Category: Extensibility functions, XML functions

Syntax: `XmlChildPos(elem, childName, N)`

See also: `cfxml`, `IsXmlDoc`, `XmlChildPos`, `XmlFormat`, `XmlNew`, `XmlParse`, `XmlSearch`, `XmlTransform`

History: New in ColdFusion MX: this function is new.

Parameters:

Parameter	Description
elem	XML element within which to search
childName	XML child element for which to search
N	Index of XMLchild element for which to search

Usage: The returned index can be used in the `ArrayInsertAt` and `ArrayDeleteAt` functions.

Example: This example searches the XML document object mydoc.employee.XmlChildren for the mydoc.employee.name[2] element:

```
XmlChildPos(mydoc.employee, "name", 2)
```

XmlElemNew

Description: Creates an XML document object element.

Return value: An XML document object element.

Category: Extensibility functions, XML functions

Syntax: `XmlElemNew(xmlObj, childName)`

See also: `cfxml`, `IsXmlDoc`, `XmlChildPos`, `XmlFormat`, `XmlNew`, `XmlParse`, `XmlSearch`, `XmlTransform`

History: New in ColdFusion MX: this function is new.

Parameters

Parameter	Description
xmlObj	The name of an XML object. An XML document or an element.
childName	The name of the element to create. This element becomes a child element of `xmlObj` in the tree.

Example: The following example creates and displays a ColdFusion document object. For more information on this example, see *Developing ColdFusion MX Applications with CFML*.

```
<cfset testVar = True>
<cfscript>
   MyDoc = XmlNew();
   MyDoc.xmlRoot = XmlElemNew(MyDoc,"MyRoot");
   if (testVar IS TRUE)
      MyDoc.MyRoot.XmlText = "The value of testVar is True.";
   else
      MyDoc.MyRoot.XmlText = "The value of testVar is False.";
   for (i = 1; i LTE 4; i = i + 1)
      {
      MyDoc.MyRoot.XmlChildren[i] = XmlElemNew(MyDoc,"childNode");
      MyDoc.MyRoot.XmlChildren[i].XmlText = "This is Child node " & i &".";
      }
</cfscript>
<cfdump var=#MyDoc#>
```

XmlFormat

Description: Escapes special XML characters in a string, so that the string is safe to use with XML.

Return value: A copy of *string* that is safe to use with XML.

Category: Extensibility functions, String functions, XML functions

Syntax: `XmlFormat(string)`

See also: `cfxml`, `IsXmlDoc`, `XmlChildPos`, `XmlChildPos`, `XmlNew`, `XmlParse`, `XmlSearch`, `XmlTransform`

History: New in ColdFusion MX: this function is new.

Parameters

Parameter	Description
string	A string or a variable that contains one

Usage: The characters that this function escapes include the following:

- Greater than symbol (>)
- Less than symbol (<)
- Single quotation mark (')
- Double quotation mark (")
- Ampersand symbol (&)

Example:

```
<h3>XMLFormat</h3>
<p>This example shows how XMLFormat is used to escape special
        XML characters and make the use of XML with ColdFusion easy.</p>
<XMP>
<?xml version = "1.0"?>
<cfoutput>
<someXML>
   <someElement someAttribute = "#XMLFormat("'a quoted value'")#>
     #XMLFormat("Body of element to be passed here.")#
   </someElement>
</someXML>
</cfoutput>
</XMP>
```

XmlNew

Description: Creates an XML document object.

Return value: An empty XML document object.

Category: Extensibility functions, XML functions

Syntax: `XmlNew([caseSensitive])`

See also: `cfxml`, `IsXmlDoc`, `XmlChildPos`, `XmlChildPos`, `XmlFormat`, `XmlParse`, `XmlSearch`, `XmlTransform`

History: New in ColdFusion MX: this function is new.

Parameters

Parameter	Description
caseSensitive	Determines how ColdFusion processes the case of XML document object component identifiers • yes: maintains case • no. ColdFusion ignores case. Default.

Usage

An XML document object is represented in ColdFusion as a structure.

The `caseSensitive` attribute value determines whether identifiers whose characters are of varying case, but are otherwise the same, refer to different components. For example:

- If `caseSensitive = "no"`, the names mydoc.employee.name[1] and mydoc.employee.NAME[1] refer to the same element
- If `caseSensitive = "yes"`, these names refer to two distinct elements

The following example creates and displays a ColdFusion document object. For more information on this example, see *Developing ColdFusion MX Applications with CFML*.

Example

```
<cfset testVar = True>
<cfscript>
  MyDoc = XmlNew();
  MyDoc.xmlRoot = XmlElemNew(MyDoc,"MyRoot");
  if (testVar IS TRUE)
    MyDoc.MyRoot.XmlText = "The value of testVar is True.";
  else
    MyDoc.MyRoot.XmlText = "The value of testVar is False.";
  for (i = 1; i LTE 4; i = i + 1)
    {
    MyDoc.MyRoot.XmlChildren[i] = XmlElemNew(MyDoc,"childNode");
    MyDoc.MyRoot.XmlChildren[i].XmlText = "This is Child node " & i &".";
    }
</cfscript>
<cfdump var=#MyDoc#>
```

XmlParse

Description: Converts an XML document that is represented as a string variable into an XML document object.

Return value: An XML document object.

Category: Conversion functions, Extensibility functions, XML functions

Syntax: `XmlParse(xmlString [, caseSensitive ] )`

See also: `cfxml`, `IsXmlDoc`, `XmlChildPos`, `XmlChildPos`, `XmlFormat`, `XmlNew`, `XmlSearch`, `XmlTransform`

History: New in ColdFusion MX: this function is new.

Parameters:

Parameter	Description
xmlString	An XML document object string
caseSensitive	• yes: maintains the case of document elements and attributes • no. Default.

Usage: The `caseSensitive` attribute value determines whether identifiers whose characters are of varying case, but are otherwise the same, refer to different components. For example:

- If `caseSensitive = "no"`, the names mydoc.employee.name[1] and mydoc.employee.NAME[1] refer to the same element
- If `caseSensitive = "yes"`, these names refer to two distinct elements

If the XML document is represented by a string variable, use the `XmlParse` tag directly on the variable. For example, if your application uses `cfhttp action="get"` to get the XML document, use the following code to create the XML document object:

```
<cfset myXMLDocument = XmlParse(cfhttp.fileContent)>
```

If the XML document is in a file, use the `cffile` tag to convert the file to a CFML variable, then use the `XmlParse` tag on the resulting variable. For example, if the XML document is in the file C:\temp\myxmldoc.xml, use the following code to convert the file to an XML document object:

```
<cffile action="read" file="C:\temp\myxmldoc.xml" variable="XMLFileText">
<cfset myXMLDocument=XmlParse(XMLFileText)>
```

Note: If the file is not encoded with the ASCII or Latin-1 character set, use the `cffile` tag `charset` attribute to specify the file's character set. For example, if the file is encoded in UTF, specify charset="UTF-8".

XmlSearch

Description: Uses an XPath language expression to search an XML document that is represented as a string variable.

Return value: An array of XML object nodes that match the search criteria.

Category: Extensibility functions, XML functions

Syntax: `XmlSearch(xmlDoc, xPathString)`

See also: `cfxml`, `IsXmlDoc`, `XmlChildPos`, `XmlChildPos`, `XmlFormat`, `XmlNew`, `XmlParse`, `XmlTransform`

History: New in ColdFusion MX: this function is new.

Parameters:

Parameter	Description
xmlDoc	XML document object
xPathString	XPath expression

Usage: XPath is specified by the World-Wide Web Consortium. For detailed information on XPath, see the W3C website at http://www.w3.org/TR/xpath.

Example: The following example extracts the elements named last, which contain employee last names, from the employeesimple.xml file, and displays the names.

```
<cffile action="read"
   file="C:\inetpub\wwwroot\examples\employeesimple.xml"
   variable="myxml">
<cfscript>
   myxmldoc = XmlParse(myxml);
   selectedElements = XmlSearch(myxmldoc, "/employee/name/last");
   for (i = 1; i LTE ArrayLen(selectedElements); i = i + 1)
      writeoutput(selectedElements[i].XmlText & "<br>");
</cfscript>
```

XmlTransform

Description: Applies an Extensible Stylesheet Language Transformation (XSLT)) to an XML document object that is represented as a string variable. An XSLT converts an XML document to another format or representation by applying an Extensible Stylesheet Language (XSL) stylesheet to it. For more information, see *Developing ColdFusion MX Applications with CFML*.

Return value: A string; an XML document after the XSLT is applied

Category: Conversion functions, Extensibility functions, XML functions

Syntax: `XmlTransform(xmlString | xmlObj, xslString)`

See also: `cfxml`, `IsXmlDoc`, `XmlChildPos`, `XmlChildPos`, `XmlFormat`, `XmlNew`, `XmlParse`, `XmlSearch`

History: New in ColdFusion MX: this function is new.

Parameters

Parameter	Description
xmlString | xmlObj	A string that represents the XML document, or a parsed object representation of it.
xslString	XSLT transformation to apply.

Year

Description From a date/time object, gets the year value.

Return value The year value of *date*.

Category Date and time functions

Syntax `Year(`*date*`)`

See also `DatePart`, `IsLeapYear`

Parameters

Parameter	Description
date	A date/time object in the range 100 AD–9999 AD. See "How ColdFusion processes two-digit year values" on page 377.

Usage When passing a date as a string, enclose it in quotation marks. Otherwise, it is interpreted as a number representation of a date.

Example

```
<h3>Year Example</h3>
<cfif IsDefined("FORM.year")>
   More information about your date:
   <cfset yourDate = CreateDate(FORM.year,FORM.month,FORM.day)>
   <cfoutput>
   <p>Your date, #DateFormat(yourDate)#.
   <br>It is #DayofWeekAsString(DayOfWeek(yourDate))#,
          day #DayOfWeek(yourDate)# in the week.
   <br>This is day #Day(yourDate)#
          in the month of #MonthAsString(Month(yourDate))#,
          which has #DaysInMonth(yourDate)# days.
   <br>We are in week #Week(yourDate)# of #Year(YourDate)#
          (day #DayofYear(yourDate)# of #DaysinYear(yourDate)#). <br>
   <cfif IsLeapYear(Year(yourDate))>
      This is a leap year
      <cfelse>This is not a leap year
   </cfif>
   </cfoutput>
</cfif>
```

YesNoFormat

Description Evaluates a number or Boolean value.

Return value Yes, for a non-zero value; No, otherwise.

Category Decision functions, Display and formatting functions

Syntax `YesNoFormat(value)`

See also `IsBinary`, `IsNumeric`

Parameters

Parameter	Description
value	A number or Boolean value

Example

```
<h3>YesNoFormat Example</h3>
<p>The YesNoFormat function returns non-zero values as "Yes"; zero values as "No".

<cfoutput>
<ul>
   <li>YesNoFormat(1):#YesNoFormat(1)#
   <li>YesNoFormat(0):#YesNoFormat(0)#
   <li>YesNoFormat("1123"):#YesNoFormat("1123")#
   <li>YesNoFormat("No"):#YesNoFormat("No")#
   <li>YesNoFormat(True):#YesNoFormat(True)#
</ul>
</cfoutput>
```

CHAPTER 6
ColdFusion C++ CFX Reference

This chapter describes the CFXAPI classes and members.

Contents

C++ class overview

A list of CFXAPI classes and members follows.

Class	Member
CCFXException class	CCFXException::GetError CCFXException::GetDiagnostics
CCFXQuery class	CCFXQuery::AddRow CCFXQuery::GetColumns CCFXQuery::GetData CCFXQuery::GetName CCFXQuery::GetRowCount CCFXQuery::SetData
CCFXRequest class	CCFXRequest::AddQuery CCFXRequest::AttributeExists CCFXRequest::CreateStringSet CCFXRequest::Debug CCFXRequest::GetAttribute CCFXRequest::GetAttributeList CCFXRequest::GetCustomData CCFXRequest::GetQuery CCFXRequest::ReThrowException CCFXRequest::SetCustomData CCFXRequest::SetVariable CCFXRequest::ThrowException CCFXRequest::Write CCFXRequest::WriteDebug
CCFXStringSet class	CCFXStringSet::AddString CCFXStringSet::GetCount CCFXStringSet::GetIndexForString CCFXStringSet::GetString

Deprecated class members

The following CFXAPI classes and members are deprecated. Do not use them in new applications. They do not work, and might cause an error, in later releases.

Class	Deprecated member	Deprecated as of this ColdFusion Server release
CCFXQuery Class	CCFXQuery::SetQueryString	ColdFusion MX
	CCFXQuery::SetTotalTime	ColdFusion MX
CCFXRequest Class	CCFXRequest::GetSetting	ColdFusion MX

CCFXException class

An abstract class that represents an exception thrown during processing of a ColdFusion Extension (CFX) procedure.

Exceptions of this type can be thrown by `CCFXRequest class`, `CCFXQuery class`, and `CCFXStringSet class`. Your ColdFusion Extension code must be written to handle exceptions of this type. For more information, see `CCFXRequest::ThrowException` and `CCFXRequest::ReThrowException`.

Class members

`virtual LPCSTR GetError()`	The CCFXException::GetError function returns a general error message.
`virtual LPCSTR GetDiagnostic()`	The CCFXException::GetDiagnostics function returns detailed error information.

CCFXException::GetError

Description Provides basic user output for exceptions that occur during processing.

CCFXException::GetDiagnostics

Description Provides detailed user output for exception that occur during processing.

Example This code block shows how GetError and GetDiagnostics work with ThrowException and ReThrowException.

```
// Write output back to the user here...
         pRequest->Write( "Hello from CFX_FOO2!" ) ;
         pRequest->ThrowException("User Error", "You goof'd...");

         // Output optional debug info
         if ( pRequest->Debug() )
         {
            pRequest->WriteDebug( "Debug info..." ) ;
         }

         // Catch Cold Fusion exceptions & re-raise them
         catch( CCFXException* e )
         {
            // This is how you would pull the error information
            LPCTSTR strError = e->GetError();
            LPCTSTR strDiagnostic = e->GetDiagnostics();

            pRequest->ReThrowException( e ) ;
         }

         // Catch ALL other exceptions and throw them as
         // Cold Fusion exceptions (DO NOT REMOVE! --
         // this prevents the server from crashing in
         // case of an unexpected exception)
```

```
catch( ... )
{
    pRequest->ThrowException(
    "Error occurred in tag CFX_FOO2",
    "Unexpected error occurred while processing tag." ) ;
}
```

CCFXQuery class

An abstract class that represents a query used or created by a ColdFusion Extension (CFX). Queries contain one or more columns of data that extend over a varying number of rows.

Class members

`virtual int AddRow()`	CCFXQuery::AddRow adds a row to a query.
`virtual int AddRow() virtual CCFXStringSet* GetColumns`	CCFXQuery::GetColumns retrieves a list of a query's column names.
`virtual LPCSTR GetData( int iRow, int iColumn )`	CCFXQuery::GetData retrieves a data element from a row and column of a query.
`virtual LPCSTR GetName()`	CCFXQuery::GetName retrieves the name of a query.
`virtual int GetRowCount()`	CCFXQuery::GetRowCount retrieves the number of rows in a query.
`virtual void SetData( int iRow, int iColumn, LPCSTR lpszData )`	CCFXQuery::SetData sets a data element within a row and column of a query.
`virtual void SetQueryString( LPCSTR lpszQuery )`	This function is deprecated. Do not use it in new applications. It might not work, and might cause an error, in later releases.
`virtual void SetTotalTime( DWORD dwMilliseconds )`	This function is deprecated. Do not use it in new applications. It might not work, and might cause an error, in later releases.

CCFXQuery::AddRow

Syntax `int CCFXQuery::AddRow(void)`

Description Add a row to the query. Call this function to append a row to a query.

Returns Returns the index of the row that was appended to a query.

Example The following example shows the addition of two rows to a three-column ('City', 'State', and 'Zip') query:

```
// First row
         int iRow ;
         iRow = pQuery->AddRow() ;
         pQuery->SetData( iRow, iCity, "Minneapolis" ) ;
         pQuery->SetData( iRow, iState, "MN" ) ;
         pQuery->SetData( iRow, iZip, "55345" ) ;
```

```
// Second row
iRow = pQuery->AddRow() ;
pQuery->SetData( iRow, iCity, "St. Paul" ) ;
pQuery->SetData( iRow, iState, "MN" ) ;
pQuery->SetData( iRow, iZip, "55105" ) ;
```

CCFXQuery::GetColumns

Syntax `CCFXStringSet* CCFXQuery::GetColumns(void)`

Description Retrieves a list of the column names contained in a query.

Returns Returns an object of CCFXStringSet class that contains a list of the columns in the query. ColdFusion automatically frees the memory that is allocated for the returned string set, after the request is completed.

Example The following example gets the list of columns, then iterates over the list, writing each column name back to the user:

```
// Get the list of columns from the query
        CCFXStringSet* pColumns = pQuery->GetColumns() ;
        int nNumColumns = pColumns->GetCount() ;

        // Print the list of columns to the user
        pRequest->Write( "Columns in query: " ) ;
        for( int i=1; i<=nNumColumns; i++ )
        {
        pRequest->Write( pColumns->GetString( i ) ) ;
        pRequest->Write( " " ) ;
        }
```

CCFXQuery::GetData

Syntax `LPCSTR CCFXQuery::GetData(int iRow, int iColumn)`

Description Gets a data element from a row and column of a query. Row and column indexes begin with 1. You can determine the number of rows in a query by calling CCFXQuery::GetRowCount. You can determine the number of columns in a query by retrieving the list of columns using CCFXQuery::GetColumns, and then calling CCFXStringSet::GetCount on the returned string set.

Returns Returns the value of the requested data element.

Parameters

Parameter	Description
iRow	Row to retrieve data from (1-based)
iColumn	Column to retrieve data from (1-based)

Example The following example iterates over the elements of a query and writes the data in the query back to the user in a simple, space-delimited format:

```
int iRow, iCol ;
          int nNumCols = pQuery->GetColumns()->GetCount() ;
          int nNumRows = pQuery->GetRowCount() ;
          for ( iRow=1; iRow<=nNumRows; iRow++ )
          {
          for ( iCol=1; iCol<=nNumCols; iCol++ )
          {
          pRequest->Write( pQuery->GetData( iRow, iCol ) ) ;
          pRequest->Write( " " ) ;
          }
          pRequest->Write( "<BR>" ) ;
          }
```

CCFXQuery::GetName

Syntax `LPCSTR CCFXQuery::GetName(void)`

Description Returns the name of a query.

Example The following example retrieves the name of a query and writes it back to the user:

```
CCFXQuery* pQuery = pRequest->GetQuery() ;
          pRequest->Write( "The query name is: " ) ;
          pRequest->Write( pQuery->GetName() ) ;
```

CCFXQuery::GetRowCount

Syntax `LPCSTR CCFXQuery::GetRowCount(void)`

Description Returns the number of rows contained in a query.

Example The following example retrieves the number of rows in a query and writes it back to the user:

```
CCFXQuery* pQuery = pRequest->GetQuery() ;
          char buffOutput[256] ;
          wsprintf( buffOutput,
          "The number of rows in the query is %ld.",
          pQuery->GetRowCount() ) ;
          pRequest->Write( buffOutput ) ;
```

CCFXQuery::SetData

Syntax `void CCFXQuery::SetData(int iRow, int iColumn, LPCSTR lpszData)`

Description Sets a data element within a row and column of a query. Row and column indexes begin with 1. Before calling `SetData` for a given row, call CCFXQuery::AddRow and use the return value as the row index for your call to `SetData`.

Parameters

Parameter	Description
iRow	Row of data element to set (1-based)
iColumn	Column of data element to set (1-based)
lpszData	New value for data element

Example The following example shows the addition of two rows to a three-column ('City', 'State', and 'Zip') query:

```
// First row
          int iRow ;
          iRow = pQuery->AddRow() ;
          pQuery->SetData( iCity, iRow, "Minneapolis" ) ;
          pQuery->SetData( iState, iRow, "MN" ) ;
          pQuery->SetData( iZip, iRow, "55345" ) ;

          // Second row
          iRow = pQuery->AddRow() ;
          pQuery->SetData( iCity, iRow, "St. Paul" ) ;
          pQuery->SetData( iState, iRow, "MN" ) ;
          pQuery->SetData( iZip, iRow, "55105" ) ;
```

CCFXRequest class

Abstract class that represents a request made to a ColdFusion Extension (CFX). An instance of this class is passed to the main function of your extension DLL. The class provides interfaces that can be used by the custom extension for the following actions:

- Reading and writing variables
- Returning output
- Creating and using queries
- Throwing exceptions

Class members

`virtual BOOL AttributeExists( LPCSTR lpszName )`	CCFXRequest::AttributeExists checks whether the attribute was passed to the tag.
`virtual LPCSTR GetAttribute( LPCSTR lpszName )`	CCFXRequest::GetAttribute gets the value of the passed attribute.
`virtual CCFXStringSet* GetAttributeList()`	CCFXRequest::GetAttributeList gets an array of attribute names passed to the tag.
`virtual CCFXQuery* GetQuery()`	CCFXRequest::GetQuery gets the query that was passed to the tag.
`virtual LPCSTR GetSetting( LPCSTR lpszSettingName )`	`CCFXRequest::GetSetting` This member is deprecated. Do not use it in new applications. It might not work, and might cause an error, in later releases.
`virtual void Write( LPCSTR lpszOutput )`	CCFXRequest::Write writes text output back to the user.
`virtual void SetVariable( LPCSTR lpszName, LPCSTR lpszValue )`	CCFXRequest::SetVariable sets a variable in the template that contains this tag.
`virtual CCFXQuery* AddQuery( LPCSTR lpszName, CCFXStringSet* pColumns )`	CCFXRequest::AddQuery adds a query to the template that contains this tag.
`virtual BOOL Debug()`	CCFXRequest::Debug checks whether the tag contains the `DEBUG` attribute.
`virtual void WriteDebug( LPCSTR lpszOutput )`	CCFXRequest::WriteDebug writes text output into the debug stream.
`virtual CCFXStringSet* CreateStringSet()`	CCFXRequest::CreateStringSet allocates and returns a `CCFXStringSet` instance.

`virtual void ThrowException( LPCSTR lpszError, LPCSTR lpszDiagnostics )`	CCFXRequest::ThrowException throws an exception and ends processing of this request.
`virtual void ReThrowException( CCFXException* e )`	CCFXRequest::ReThrowException re-throws an exception that has been caught.
`virtual void SetCustomData( LPVOID lpvData )`	CCFXRequest::SetCustomData sets custom (tag specific) data to carry with a request.
`virtual LPVOID GetCustomData()`	CCFXRequest::GetCustomData gets custom (tag specific) data for a request.

CCFXRequest::AddQuery

Syntax

```
CCFXQuery* CCFXRequest::AddQuery(LPCSTR lpszName,
        CCFXStringSet* pColumns)
```

Description

Adds a query to the calling template. The query can be accessed by CFML tags (for example, `CFOUTPUT` or `CFTABLE`) within the template. After calling `AddQuery`, the query is empty (it has 0 rows). To populate the query with data, call the CCFXQuery::AddRow and CCFXQuery::SetData functions.

Returns

Returns a pointer to the query that was added to the template (an object of class `CCFXQuery`). The memory allocated for the returned query is freed automatically by ColdFusion after the request is completed.

Parameters

Parameter	Description
lpszName	Name of query to add to the template (must be unique)
pColumns	List of column names to be used in the query

Example

The following example adds a query named 'People' to the calling template. The query has two columns ('FirstName' and 'LastName') and two rows:

```
// Create a string set and add the column names to it
        CCFXStringSet* pColumns = pRequest->CreateStringSet() ;
        int iFirstName = pColumns->AddString( "FirstName" ) ;
        int iLastName = pColumns->AddString( "LastName" ) ;

        // Create a query that contains these columns
        CCFXQuery* pQuery = pRequest->AddQuery( "People", pColumns ) ;

        // Add data to the query
        int iRow ;
        iRow = pQuery->AddRow() ;
```

```
pQuery->SetData( iRow, iFirstName, "John" ) ;
pQuery->SetData( iRow, iLastName, "Smith" ) ;
iRow = pQuery->AddRow() ;
pQuery->SetData( iRow, iFirstName, "Jane" ) ;
pQuery->SetData( iRow, iLastName, "Doe" ) ;
```

CCFXRequest::AttributeExists

Syntax `BOOL CCFXRequest::AttributeExists(LPCSTR lpszName)`

Description Checks whether the attribute was passed to the tag. Returns True if the attribute is available; False, otherwise.

Parameters

Parameter	Description
lpszName	Name of the attribute to check (case insensitive)

Example The following example checks whether the user passed an attribute named DESTINATION to the tag, and throws an exception if the attribute was not passed:

```
if ( pRequest->AttributeExists("DESTINATION")==FALSE )
          {
          pRequest->ThrowException(
          "Missing DESTINATION parameter",
          "You must pass a DESTINATION parameter in "
          "order for this tag to work correctly." ) ;
          }
```

CCFXRequest::CreateStringSet

Syntax `CCFXStringSet* CCFXRequest::CreateStringSet(void)`

Description Allocates and returns an instance. Always use this function to create string sets, as opposed to directly using the 'new' operator.

Returns Returns an object of CCFXStringSet class. The memory allocated for the returned string set is freed automatically by ColdFusion after the request is completed

Example The following example creates a string set and adds three strings to it:

```
CCFXStringSet* pColors = pRequest->CreateStringSet() ;
          pColors->AddString( "Red" ) ;
          pColors->AddString( "Green" ) ;
          pColors->AddString( "Blue" ) ;
```

CCFXRequest::Debug

Syntax `BOOL CCFXRequest::Debug(void)`

Description Checks whether the tag contains the `DEBUG` attribute. Use this function to determine whether to write debug information for a request. For more information, see CCFXRequest::WriteDebug.

Returns Returns True if the tag contains the DEBUG attribute; False, otherwise.

Example The following example checks whether the DEBUG attribute is present, and if it is, it writes a brief debug message:

```
if ( pRequest->Debug() )
          {
             pRequest->WriteDebug( "Top secret debug info" ) ;
          }
```

CCFXRequest::GetAttribute

Syntax `LPCSTR CCFXRequest::GetAttribute(LPCSTR lpszName)`

Description Retrieves the value of the passed attribute. Returns an empty string if the attribute does not exist. (To test whether an attribute was passed to the tag, use CCFXRequest::AttributeExists.)

Returns Returns the value of the attribute passed to the tag. If no attribute of that name was passed to the tag, an empty string is returned.

Parameters

Parameter	Description
lpszName	Name of the attribute to retrieve (case insensitive)

Example The following example retrieves an attribute named DESTINATION and writes its value back to the user:

```
LPCSTR lpszDestination = pRequest->GetAttribute("DESTINATION") ;
          pRequest->Write( "The destination is: " ) ;
          pRequest->Write( lpszDestination ) ;
```

CCFXRequest::GetAttributeList

Syntax `CCFXStringSet* CCFXRequest::GetAttributeList(void)`

Description Gets an array of attribute names passed to the tag. To get the value of one attribute, use CCFXRequest::GetAttribute.

Returns Returns an object of class CCFXStringSet class that contains a list of attributes passed to the tag. The memory allocated for the returned string set is freed automatically by ColdFusion after the request is completed.

Example The following example gets the list of attributes and iterates over the list, writing each attribute and its value back to the user.

```
LPCSTR lpszName, lpszValue ;
          CCFXStringSet* pAttribs = pRequest->GetAttributeList() ;
          int nNumAttribs = pAttribs->GetCount() ;

          for( int i=1; i<=nNumAttribs; i++ )
          {
             lpszName = pAttribs->GetString( i ) ;
             lpszValue = pRequest->GetAttribute( lpszName ) ;
```

```
        pRequest->Write( lpszName ) ;
        pRequest->Write( " = " ) ;
        pRequest->Write( lpszValue ) ;
        pRequest->Write( "<BR>" ) ;
    }
```

CCFXRequest::GetCustomData

Syntax `LPVOID CCFXRequest::GetCustomData(void)`

Description Gets the custom (tag specific) data for the request. This member is typically used from within subroutines of a tag implementation to extract tag data from a request.

Returns Returns a pointer to the custom data, or `NULL` if no custom data has been set during this request using CCFXRequest::SetCustomData.

Example The following example retrieves a pointer to a request specific data structure of hypothetical type MYTAGDATA:

```
void DoSomeGruntWork( CCFXRequest* pRequest )
        {
        MYTAGDATA* pTagData =
        (MYTAGDATA*)pRequest->GetCustomData() ;

        ... remainder of procedure ...
        }
```

CCFXRequest::GetQuery

Syntax `CCFXQuery* CCFXRequest::GetQuery(void)`

Description Retrieves a query that was passed to a tag. To pass a query to a custom tag, you use the `QUERY` attribute. This attribute should be set to the name of a query (created using the CFQUERY tag or another custom tag). The `QUERY` attribute is optional and should be used only by tags that process an existing data set.

Returns Returns an object of the CCFXQuery class that represents the query passed to the tag. If no query was passed to the tag, `NULL` is returned. The memory allocated for the returned query is freed automatically by ColdFusion after the request is completed.

Example The following example retrieves the query that was passed to the tag. If no query was passed, an exception is thrown:

```
CCFXQuery* pQuery = pRequest->GetQuery() ;
        if ( pQuery == NULL )
        {
            pRequest->ThrowException(
               "Missing QUERY parameter",
               "You must pass a QUERY parameter in "
               "order for this tag to work correctly." ) ;
        }
```

CCFXRequest::ReThrowException

Syntax `void CCFXRequest::ReThrowException(CCFXException* e)`

Description Re-throws an exception that has been caught within an extension procedure. This function is used to avoid having C++ exceptions that are thrown by DLL extension code propagate back into ColdFusion. Catch ALL C++ exceptions that occur in extension code, and either re-throw them (if they are of the CCFXException class) or create and throw a new exception pointer using CCFXRequest::ThrowException.

Parameters

Parameter	Description
e	A CCFXException that has been caught

Example The following code demonstrates how to handle exceptions in ColdFusion Extension DLL procedures:

```
try
        {
            ...Code that could throw an exception...
        }
        catch( CCFXException* e )
        {
            ...Do appropriate resource cleanup here...
            // Re-throw the exception
            pRequest->ReThrowException( e ) ;
        }
        catch( ... )
        {
            // Something nasty happened

            pRequest->ThrowException(
            "Unexpected error occurred in CFX tag", "" ) ;
        }
```

CCFXRequest::SetCustomData

Syntax `void CCFXRequest::SetCustomData(LPVOID lpvData)`

Description Sets custom (tag specific) data to carry with the request. Use this function to store request specific data to pass to procedures within your custom tag implementation.

Parameters

Parameter	Description
lpvData	Pointer to custom data

Example The following example creates a request-specific data structure of hypothetical type MYTAGDATA and stores a pointer to the structure in the request for future use:

```
void ProcessTagRequest( CCFXRequest* pRequest )
        try
        {
            MYTAGDATA tagData ;
            pRequest->SetCustomData( (LPVOID)&tagData ) ;

            ... remainder of procedure ...
        }
```

CCFXRequest::SetVariable

Syntax `void CCFXRequest::SetVariable(LPCSTR lpszName, LPCSTR lpszValue)`

Description Sets a variable in the calling template. If the variable name already exists in the template, its value is replaced. If it does not exist, a variable is created. The values of variables created using `SetVariable` can be accessed in the same manner as other template variables (for example, #MessageSent#).

Parameters

Parameter	Description
lpszName	Name of variable
lpszValue	Value of variable

Example The following example sets the value of a variable named `'MessageSent'` based on the success of an operation performed by the custom tag:

```
BOOL bMessageSent;
        ...attempt to send the message...
        if ( bMessageSent == TRUE )
        {
            pRequest->SetVariable( "MessageSent", "Yes" ) ;
        }
        else
        {
            pRequest->SetVariable( "MessageSent", "No" ) ;
        }
```

CCFXRequest::ThrowException

Syntax `void CCFXRequest::ThrowException(LPCSTR lpszError, LPCSTR lpszDiagnostics)`

Description Throws an exception and ends processing of a request. Call this function when you encounter an error that does not allow you to continue processing the request. This function is almost always combined with the CCFXRequest::ReThrowException to protect against resource leaks in extension code.

Parameters

Parameter	Description
lpszError	Short identifier for error
lpszDiagnostics	Error diagnostic information

Example The following example throws an exception indicating that an unexpected error occurred while processing a request:

```
char buffError[512] ;
          wsprintf( buffError,
          "Unexpected Windows NT error number %ld "
          "occurred while processing request.", GetLastError() ) ;

          pRequest->ThrowException( "Error occurred", buffError ) ;
```

CCFXRequest::Write

Syntax `void CCFXRequest::Write(LPCSTR lpszOutput)`

Description Writes text output back to the user.

Parameters

Parameter	Description
lpszOutput	Text to output

Example The following example creates a buffer to hold an output string, fills the buffer with data, and writes the output back to the user:

```
CHAR buffOutput[1024] ;
          wsprintf( buffOutput, "The destination is: %s",
          pRequest->GetAttribute("DESTINATION") ) ;
          pRequest->Write( buffOutput ) ;
```

CCFXRequest::WriteDebug

Syntax `void CCFXRequest::WriteDebug(LPCSTR lpszOutput)`

Description Writes text output into the debug stream. The text is only displayed to the end-user if the tag contains the `DEBUG` attribute. (For more information, see CCFXRequest::Debug.)

Parameters

Parameter	Description
lpszOutput	Text to output

Example The following example checks whether the DEBUG attribute is present; if so, it writes a brief debug message:

```
if ( pRequest->Debug() )
          {
             pRequest->WriteDebug( "Top secret debug info" ) ;
          }
```

CCFXStringSet class

Abstract class that represents a set of ordered strings. Strings can be added to a set and can be retrieved by a numeric index (index values for strings are 1-based). To create a string set, use CCFXRequest::CreateStringSet.

Class members

`virtual int AddString( LPCSTR lpszString )`	CCFXStringSet::AddString adds a string to the end of a list.
`virtual int GetCount()`	CCFXStringSet::GetCount gets the number of strings contained in a list.
`virtual LPCSTR GetString( int iIndex )`	CCFXStringSet::GetString gets the string located at the passed index.
`virtual int GetIndexForString( LPCSTR lpszString )`	CCFXStringSet::GetIndexForString gets the index for the passed string.

CCFXStringSet::AddString

Syntax `int CCFXStringSet::AddString(LPCSTR lpszString)`

Description Adds a string to the end of the list.

Returns The index of the string that was added.

Parameters

Parameter	Description
lpszString	String to add to the list

Example The following example demonstrates adding three strings to a string set and saving the indexes of the items that are added:

```
CCFXStringSet* pSet = pRequest->CreateStringSet() ;
          int iRed = pSet->AddString( "Red" ) ;
          int iGreen = pSet->AddString( "Green" ) ;
          int iBlue = pSet->AddString( "Blue" ) ;
```

CCFXStringSet::GetCount

Syntax `int CCFXStringSet::GetCount(void)`

Description Gets the number of strings in a string set. The value can be used with CCFXStringSet::GetString to iterate over the strings in the set (recall that the index values for strings in the list begin at 1).

Returns Returns the number of strings contained in the string set.

Example The following example demonstrates using `GetCount` with CCFXStringSet::GetString to iterate over a string set and write the contents of the list back to the user:

```
int nNumItems = pStringSet->GetCount() ;
          for ( int i=1; i<=nNumItems; i++ )
          {
             pRequest->Write( pStringSet->GetString( i ) ) ;
             pRequest->Write( "<BR>" ) ;
          }
```

CCFXStringSet::GetIndexForString

Syntax `int CCFXStringSet::GetIndexForString(LPCSTR` *`lpszString`*`)`

Description Searches for a passed string. The search is case-insensitive.

Returns If the string is found, its index within the string set is returned. If it is not found, the constant `CFX_STRING_NOT_FOUND` is returned.

Parameters

Parameter	Description
lpszString	String to search for

Example The following example demonstrates a search for a string and throwing an exception if it is not found:

```
CCFXStringSet* pAttribs = pRequest->GetAttributeList() ;

          int iDestination =
          pAttribs->GetIndexForString("DESTINATION") ;
          if ( iDestination == CFX_STRING_NOT_FOUND )
          {
             pRequest->ThrowException(
             "DESTINATION attribute not found."
             "The DESTINATION attribute is required "
             "by this tag." ) ;
          }
```

CCFXStringSet::GetString

Syntax `LPCSTR CCFXStringSet::GetString(int` *`iIndex`*`)`

Description Retrieves the string located at the passed index (index values are 1-based).

Returns Returns the string located at the passed index.

Parameters

Parameter	Description
iIndex	Index of string to retrieve

Example The following example demonstrates `GetString` with CCFXStringSet::GetCount to iterate over a string set and write the contents of a list back to the user:

```
int nNumItems = pStringSet->GetCount() ;
          for ( int i=1; i<=nNumItems; i++ )
          {
              pRequest->Write( pStringSet->GetString( i ) ) ;
              pRequest->Write( "<BR>" ) ;
          }
```

CHAPTER 7
ColdFusion Java CFX Reference

This chapter describes the Java interfaces available for building ColdFusion custom CFXs in Java.

Contents

Overview class libraries

The following Java interfaces are available for building ColdFusion custom CFXs in Java.

Interface	Methods
CustomTag interface	processRequest
Query interface	addRow getColumnIndex getColumns getData getName getRowCount setData
Request interface	attributeExists debug getAttribute getAttributeList getIntAttribute getQuery getSetting
Response interface	addQuery setVariable write writeDebug

CustomTag interface

```
public abstract interface CustomTag
```

Interface for implementing custom tags.

Classes that implement this interface can be specified in the CLASS attribute of the Java CFX tag. For example, in a class `MyCustomTag`, which implements this interface, the following CFML code calls the `MyCustomTag.processRequest` method:

```
<CFX_MyCustomTag>
```

Other attributes can be passed to the Java CFX tag. Their values are available using the Request object passed to the `processRequest` method.

Methods

Returns	Syntax	Description
`void`	`processRequest(Request request, Response response)`	Processes a request originating from the `CFX_mycustomtag` tag

processRequest

Description: Processes a request originating from the Java `CFX` tag.

Category: CustomTag interface

Syntax: `public void processRequest(Request request, Response response)`

Throws: `Exception` If an unexpected error occurs while processing the request.

Parameters

Parameter	Description
request	Parameters (attributes, query, and so on.) for this request
response	Interface for generating response to request (output, variables, queries, and so on.)

Query interface

```
public abstract interface Query
```

Interface to a query used or created by a CustomTag. A query contains tabular data organized by named columns and rows.

Methods

Returns	Method	Description
int	addRow()	Adds a row to the query
int	getColumnIndex(String name)	Gets the index of a column given its name
String[]	getColumns()	Gets a list of the column names in a query
String	getData(int iRow, int iCol)	Gets a data element from a row and column of a query.
String	getName()	Gets the name of a query
int	getRowCount()	Gets the number of rows in a query
void	setData(int iRow, int iCol, String data)	Sets a data element in a row and column of a query.

addRow

Description: Adds a row to a query. Call this method to append a row to a query.

Returns the index of the row that was appended to the query.

Category: Query interface

Syntax:

```
public int addRow()
```

See also: setData, getData

Example: The following example demonstrates the addition of two rows to a query that has three columns, 'City', 'State', and 'Zip':

```
// Define column indexes
int iCity = 1, iState = 2, iZip = 3 ;

// First row
int iRow = query.addRow() ;
query.setData( iRow, iCity, "Minneapolis" ) ;
query.setData( iRow, iState, "MN" ) ;
query.setData( iRow, iZip, "55345" ) ;
// Second row
iRow = query.addRow() ;
query.setData( iRow, iCity, "St. Paul" ) ;
query.setData( iRow, iState, "MN" ) ;
query.setData( iRow, iZip, "55105" ) ;
```

getColumnIndex

Description Returns the index of the column, or -1 if no such column exists.

Category Query interface

Syntax `public int getColumnIndex(String name)`

See also `getColumns`, `getData`

Parameters

Parameter	Description
name	Name of column to get index of (lookup is case insensitive)

Example The following example retrieves the index of the EMAIL column and uses it to output a list of the addresses contained in the column:

```
// Get the index of the EMAIL column
int iEMail = query.getColumnIndex( "EMAIL" ) ;

// Iterate over the query and output list of addresses
int nRows = query.getRowCount() ;
for( int iRow = 1; iRow <= nRows; iRow++ )
{
   response.write( query.getData( iRow, iEMail ) + "<BR>" ) ;
}
```

getColumns

Description Returns an array of strings containing the names of the columns in the query.

Category Query interface

Syntax `public String[] getColumns()`

Example The following example retrieves the array of columns, then iterates over the list, writing each column name back to the user:

```
// Get the list of columns from the query
String[] columns = query.getColumns() ;
int nNumColumns = columns.length ;

// Print the list of columns to the user
response.write( "Columns in query: " ) ;
for( int i=0; i<nNumColumns; i++ )
{
   response.write( columns[i] + " " ) ;
}
```

getData

Description Retrieves a data element from a row and column of a query. Row and column indexes begin with 1. You can find the number of rows in a query by calling `getRowCount`. You can find the number of columns in a query by calling `getColumns`.

Returns the value of the requested data element.

Category Query interface

Syntax `public String getData(int iRow, int iCol)`

Throws `IndexOutOfBoundsException` If an invalid index is passed to the method

See also `setData`, `addRow`

Parameters

Parameter	Description
iRow	Row to retrieve data from (1-based)
iCol	Column to retrieve data from (1-based)

Example The following example iterates over the rows of a query and writes the data back to the user in a simple, space-delimited format:

```
int iRow, iCol ;
int nNumCols = query.getColumns().length ;
int nNumRows = query.getRowCount() ;
for ( iRow = 1; iRow <= nNumRows; iRow++ )
{
   for ( iCol = 1; iCol <= nNumCols; iCol++ )
   {
      response.write( query.getData( iRow, iCol ) + " " ) ;
   }
   response.write( "<BR>" ) ;
}
```

getName

Description Returns the name of a query.

Category Query interface

Syntax `public String getName()`

Example The following example retrieves the name of a query and writes it back to the user:

```
Query query = request.getQuery() ;
   response.write( "The query name is: " + query.getName() ) ;
```

getRowCount

Description Retrieves the number of rows in a query.

Returns the number of rows contained in a query.

Category Query interface

Syntax `public int getRowCount()`

Example The following example retrieves the number of rows in a query and writes it back to the user:

```
Query query = request.getQuery() ;
   int rows = query.getRowCount() ;
   response.write( "The number of rows in the query is "
   + Integer.toString(rows) ) ;
```

setData

Description Sets a data element in a row and column of a query. Row and column indexes begin with 1. Before calling `setData` for a given row, call `addRow` and use the return value as the row index for your call to `setData`.

Category Query interface

Syntax `public void setData(int iRow, int iCol, String data)`

Throws `IndexOutOfBoundsException` If an invalid index is passed to the method

See also `getData`, `addRow`

Parameters

Parameter	Description
`iRow`	Row of data element to set (1-based)
`iCol`	Column of data element to set (1-based)
`data`	New value for data element

Example The following example demonstrates the addition of two rows to a query that has three columns, 'City', 'State', and 'Zip':

```
// Define column indexes
int iCity = 1, iState = 2, iZip = 3 ;

// First row
int iRow = query.addRow() ;
query.setData( iRow, iCity, "Minneapolis" ) ;
query.setData( iRow, iState, "MN" ) ;
query.setData( iRow, iZip, "55345" ) ;


// Second row
iRow = query.addRow() ;
```

```
query.setData( iRow, iCity, "St. Paul" ) ;
query.setData( iRow, iState, "MN" ) ;
query.setData( iRow, iZip, "55105" ) ;
```

Request interface

```
public abstract interface Request
```

Interface to a request made to a CustomTag. The interface includes methods for retrieving attributes passed to the tag (including queries) and reading global tag settings.

Methods

Returns	Syntax	Description
boolean	attributeExists(String name)	Checks whether the attribute was passed to this tag.
boolean	debug()	Checks whether the tag contains the DEBUG attribute.
String	getAttribute(String name)	Retrieves the value of the passed attribute.
String	getAttributeList()	Retrieves a list of attributes passed to the tag.
int	getIntAttribute(String name)	Retrieves the value of the passed attribute as an integer.
int	getIntAttribute(String name, int def)	Retrieves the value of the passed attribute as an integer (returns default if the attribute does not exist or is not a valid number).
Query	getQuery()	Retrieves the query that was passed to this tag.

attributeExists

Description: Checks whether the attribute was passed to this tag.

Returns `true` if the attribute is available, otherwise returns `false`.

Category: Request interface

Syntax:

```
public boolean attributeExists(String name)
```

See also: `getAttribute`, `getAttributeList`

Parameters

Parameter	Description
name	Name of the attribute to check (case insensitive)

Example: The following example checks whether the user passed an attribute named DESTINATION to the tag; if not, it throws an exception:

```
if ( ! request.attributeExists("DESTINATION") )
{
   throw new Exception(
   "Missing DESTINATION parameter",
```

```
   "You must pass a DESTINATION parameter in "
   "order for this tag to work correctly." ) ;
} ;
```

debug

Description: Checks whether the tag contains the DEBUG attribute. Use this method to determine whether to write debug information for this request. For more information, see `writeDebug`.

Returns true if the tag contains the DEBUG attribute otherwise returns false.

Category: Request interface

Syntax:
```
public boolean debug()
```

See also: `writeDebug`

Example: The following example checks whether the DEBUG attribute is present, and if so, it writes a brief debug message:

```
if ( request.debug() )
{
   response.writeDebug( "debug info" ) ;
}
```

getAttribute

Description: Retrieves the value of a passed attribute. Returns an empty string if the attribute does not exist (use `attributeExists` to test whether an attribute was passed to the tag). Use `getAttribute(String,String)` to return a default value rather than an empty string.

Returns the value of the attribute passed to the tag. If no attribute of that name was passed to the tag, an empty string is returned.

Category: Request interface

Syntax:
```
public String getAttribute(String name)
```

See also: `attributeExists`, `getAttributeList`, `getIntAttribute`, `getAttribute`

Parameters:

Parameter	Description
name	The attribute to retrieve (case insensitive)

Example: The following example retrieves an attribute named DESTINATION and writes its value back to the user:

```
String strDestination = request.getAttribute("DESTINATION") ;
response.write( "The destination is: " + strDestination ) ;
```

getAttributeList

Description Retrieves a list of attributes passed to the tag. To retrieve the value of one attribute, use the `getAttribute` member function.

Returns an array of strings containing the names of the attributes passed to the tag.

Category Request interface

Syntax `public String[] getAttributeList()`

See also `attributeExists`, `getAttributeList`

Example The following example retrieves the list of attributes, then iterates over the list, writing each attribute and its value back to the user:

```
String[] attribs = request.getAttributeList() ;
int nNumAttribs = attribs.length ;

for( int i = 0; i < nNumAttribs; i++ )
{
   String strName = attribs[i] ;
   String strValue = request.getAttribute( strName ) ;
   response.write( strName + "=" + strValue + "<BR>" ) ;
}
```

getIntAttribute

Description Retrieves the value of the passed attribute as an integer. Returns -1 if the attribute does not exist. Use `attributeExists` to test whether an attribute was passed to the tag. Use `getIntAttribute(String,int)` to return a default value rather than throwing an exception or returning -1.

Returns the value of the attribute passed to the tag. If no attribute of that name was passed to the tag, -1 is returned.

Category Request interface

Syntax `public int getIntAttribute(String name)`

Throws `NumberFormatException` If the attribute is not a valid number.

See also `attributeExists`, `getAttributeList`, `getIntAttribute`

Parameters

Parameter	Description
name	The attribute to retrieve (case insensitive)

Example The following example retrieves an attribute named PORT and writes its value back to the user:

```
int nPort = request.getIntAttribute("PORT") ;
if ( nPort != -1 )
   response.write( "The port is: " + String.valueOf(nPort) ) ;
```

getQuery

Description: Retrieves the query that was passed to this tag.

To pass a query to a custom tag, you use the QUERY attribute. It should be set to the name of a query (created using the `cfquery` tag). The QUERY attribute is optional and should be used only by tags that process an existing dataset.

Returns the Query that was passed to the tag. If no query was passed, returns null.

Category: Request interface

Syntax:

```
public Query getQuery()
```

Example: The following example retrieves a query that was passed to a tag. If no query was passed, an exception is thrown:

```
Query query = request.getQuery() ;
if ( query == null )
{
   throw new Exception(
   "Missing QUERY parameter. " +
   "You must pass a QUERY parameter in "
   "order for this tag to work correctly." ) ;
}
```

getSetting

Description: Retrieves the value of a global custom tag setting. Custom tag settings are stored in the CustomTags section of the ColdFusion Registry key.

Returns the value of the custom tag setting. If no setting of that name exists, an empty string is returned.

Category: Request interface

Syntax:

```
public String getSetting(String name)
```

Parameters:

Parameter	Description
name	The name of the setting to retrieve (case insensitive)

Usage: All custom tags implemented in Java share a registry key for storing settings. To avoid name conflicts, preface the names of settings with the name of your CustomTag class. For example, the code below retrieves the value of a setting named 'VerifyAddress' for a CustomTag class named MyCustomTag:

```
String strVerify = request.getSetting("MyCustomTag.VerifyAddress") ;
if ( Boolean.valueOf(strVerify) )
{
   // Do address verification...
}
```

Response interface

```
public abstract interface Response
```

Interface to response generated from a CustomTag. This interface includes methods for writing output, generating queries, and setting variables in the calling page.

Methods

Returns	Syntax	Description
Query	addQuery(String name, String[] columns)	Adds a query to the calling template.
void	setVariable(String name, String value)	Sets a variable in the calling template.
void	write(String output)	Outputs text back to the user.
void	writeDebug(String output)	Writes text output into the debug stream.

addQuery

Description: Adds a query to the calling template. The query can be accessed by CFML tags in the template. After calling `addQuery`, the query is empty (it has 0 rows). To populate the query with data, call the Query member functions `addRow` and `setData`.

Returns the Query that was added to the template.

Category: Response interface

Syntax:

```
public Query addQuery(String name, String[] columns)
```

Throws: IllegalArgumentException - if the name parameter is not a valid CFML variable name

See also: `addRow`, `setData`

Parameters

Parameter	Description
name	The name of the query to add to the template
columns	The column names to use in the query

Example: The following example adds a Query named 'People' to the calling template. The query has two columns ('FirstName' and 'LastName') and two rows:

```
// Create string array with column names (also track columns indexes)
String[] columns = { "FirstName", "LastName" } ;
int iFirstName = 1, iLastName = 2 ;

// Create a query which contains these columns
Query query = response.addQuery( "People", columns ) ;
```

```
// Add data to the query
int iRow = query.addRow() ;
query.setData( iRow, iFirstName, "John" ) ;
query.setData( iRow, iLastName, "Smith" ) ;
iRow = query.addRow() ;
query.setData( iRow, iFirstName, "Jane" ) ;
query.setData( iRow, iLastName, "Doe" ) ;
```

setVariable

Description: Sets a variable in the calling template. If the variable name specified exists in the template, its value is replaced. If it does not exist, a new variable is created.

Category: Response interface

Syntax: `public void setVariable(String name, String value)`

Throws: `IllegalArgumentException` If the `name` parameter is not a valid CFML variable name

Parameters

Parameter	Description
`name`	The name of the variable to set
value	The value to set the variable to

Example: For example, this code sets the value of a variable named 'MessageSent' based on the success of an operation performed by the custom tag:

```
boolean bMessageSent ;

  ...attempt to send the message...

  if ( bMessageSent == true )
  {
    response.setVariable( "MessageSent", "Yes" ) ;
  }
  else
  {
    response.setVariable( "MessageSent", "No" ) ;
  }
```

write

Description: Outputs text back to the user.

Category: Response interface

Syntax: `public void write(String output)`

Parameters

Parameter	Description
`output`	Text to output

Example The following example outputs the value of the DESTINATION attribute:

```
response.write( "DESTINATION = " +
   request.getAttribute("DESTINATION") ) ;
```

writeDebug

Description Writes text output into the debug stream. This text is displayed to the end-user only if the tag contains the DEBUG attribute (check for this attribute using the `Request.debug` member function).

Category Response interface

Syntax `public void writeDebug(String output)`

See also `debug`

Parameters

Parameter	Description
`output`	The text to output

Example The following example checks whether the DEBUG attribute is present; if so, it writes a brief debug message:

```
if ( request.debug() )
{
   response.writeDebug( "debug info" ) ;
}
```

Debugging classes reference

The constructors and methods supported by the `DebugRequest`, `DebugResponse`, and `DebugQuery` classes are as follows. These classes also support the other methods of the `Request`, `Response`, and `Query` interfaces, respectively.

DebugRequest

```
// initialize a debug request with attributes
public DebugRequest( Hashtable attributes ) ;

// initialize a debug request with attributes and a query
public DebugRequest( Hashtable attributes, Query query ) ;

// initialize a debug request with attributes, a query, and settings
public DebugRequest( Hashtable attributes, Query query,
                     Hashtable settings ) ;
```

DebugResponse

```
// initialize a debug response
public DebugResponse() ;

// print the results of processing
public void printResults() ;
```

DebugQuery

```
// initialize a query with name and columns
public DebugQuery( String name, String[] columns )
   throws IllegalArgumentException ;

// initialize a query with name, columns, and data
public DebugQuery( String name, String[] columns, String[][] data )
   throws IllegalArgumentException ;
```

CHAPTER 8
WDDX JavaScript Objects

This chapter provides information about JavaScript objects and functions used to WDDX in a ColdFusion application.

Contents

JavaScript object overview

These are the JavaScript objects and functions.

Class	Members
WddxSerializer object	serialize serializeVariable serializeValue write
WddxRecordset object	addColumn addRows getField getRowCount setField wddxSerialize

WDDX JavaScript objects are defined in the wddx.js file; this file is installed in the webroot/cfide/scripts directory.

To use these objects, you must put a `JavaScript` tag before the code that refers to the objects; for example:

```
<script type="text/javascript" src="/CFIDE/scripts/wddx.js"></script>
```

WddxSerializer object

The WddxSerializer object includes functions that serialize any JavaScript data structure.

Functions

The only function that developers typically call is `serialize`.

Function syntax	Description
object.**serialize**(rootobj)	Creates a WDDX packet for a passed WddxRecordset instance.
object.**serializeVariable**(name, obj)	Serializes a property of a structure. If an object is not a string, number, array, Boolean, or a date, WddxSerializer treats it as a structure.
object.**serializeValue**(obj)	Recursively serializes eligible data in a passed instance.
object.**write**(str)	Appends data to the serialized data stream.

serialize

Description: Creates a WDDX packet for a passed WddxRecordset instance.

Syntax: *object*.**serialize**(*rootobj*)

Parameters

Parameter	Description
object	Instance name of the WddxSerializer object
rootobj	JavaScript data structure to serialize

Return value: Returns a serialized WDDX packet as a String if the function succeeds, or a null value if an error occurs.

Usage: Call this function to serialize the data in a WddxRecordset instance.

Example: This example shows a JavaScript function that you can call to serialize a WddxRecordset instance. It copies serialized data to a form field for display:

```
function serializeData(data, formField)
{
   wddxSerializer = new WddxSerializer();
   wddxPacket = wddxSerializer.serialize(data);
   if (wddxPacket != null)
   {
      formField.value = wddxPacket;
   }
   else
   {
      alert("Couldn't serialize data");
   }
}
```

serializeVariable

Description Serializes a property of a structure. If an object is not a string, number, array, Boolean, or date, WddxSerializer treats it as a structure.

Syntax `object.serializeVariable( name, obj )`

Parameters

Parameter	Description
object	Instance name of a WddxSerializer object
name	Property to serialize
obj	Instance name of the value to serialize

Return value Returns a Boolean True if serialization was successful, or False if an error occurs.

Usage This is an internal function; you do not typically call it.

Example This example is from the WddxSerializer serializeValue function:

```
...
// Some generic object; treat it as a structure
   this.write("<struct>");
   for (prop in obj)
   {
      bSuccess = this.serializeVariable(prop, obj[prop]);
      if (! bSuccess)
      {
         break;
      }
   }
   this.write("</struct>");
...
```

serializeValue

Description Recursively serializes eligible data in a passed instance. Eligible data includes:

- String
- Number
- Boolean
- Date
- Array
- Recordset
- Any JavaScript object

This function serializes null values as empty strings.

Syntax `object.serializeValue( obj )`

Parameters

Parameter	Description
object	Instance name of the WddxSerializer object
obj	Instance name of the WddxRecordset object to serialize

Return value Returns a Boolean True if *obj* was serialized successfully; or False if an error occurs.

Usage This is an internal function; you do not typically call it.

Example This example is from the WddxSerializer serialize function:

```
...
this.wddxPacket = "";
this.write("<wddxPacket version='1.0'><header/><data>");
bSuccess = this.serializeValue(rootObj);
this.write("</data></wddxPacket>");
if (bSuccess)
{
   return this.wddxPacket;
}
else
{
   return null;
}
...
```

write

Description Appends data to a serialized data stream.

Syntax *object*.**write**(*str*)

Parameters

Parameter	Description
object	Instance name of the WddxSerializer object
str	String to be copied to the serialized data stream

Return value Returns an updated serialized data stream as a String.

Usage This is an internal function; you do not typically call it.

Example This example is from the WddxSerializer serializeValue function:

```
...
else if (typeof(obj) == "number")
{
   // Number value
   this.write("<number>" + obj + "</number>");
}
else if (typeof(obj) == "boolean")
{
   // Boolean value
```

```
    this.write("<boolean value='" + obj + "'/>");
}
...
```

WddxRecordset object

Includes functions that you call as needed when constructing a WDDX record set.

Functions

Function syntax	Description
object.addColumn(name)	Adds a column to all rows in a WddxRecordset instance.
object.addRows(n)	Adds rows to all columns in a WddxRecordset instance.
object.dump(escapeStrings)	Displays WddxRecordset object data.
object.getField(row, col)	Returns the element in a row/column position.
object.getRowCount()	Indicates the number of rows in a WddxRecordset instance.
object.setField(row, col, value)	Sets the element in a row/column position.
object.wddxSerialize(serializer)	Serializes a record set.

Return value HTML table of the WddxRecordset object data.

Usage Convenient for debugging and testing record sets. The boolean parameter `escapeStrings` determines whether `<>&` characters in string values are escaped as `<>&` in HTML.

Example

```
<!--- Create a simple query --->
<cfquery name = "q" datasource ="cfsnippets">
SELECT Message_Id, Thread_id, Username, Posted
FROM messages
</cfquery>
<!--- Load the wddx.js file, which includes the dump function --->
<script type="text/javascript" src="/CFIDE/scripts/wddx.js"></script>
<script>
// Use WDDX to move from CFML data to JS
<cfwddx action="cfml2js" input="#q#" topLevelVariable="qj">
// Dump the record set
document.write(qj.dump(true));
</script>
```

addColumn

Description Adds a column to all rows in a WddxRecordset instance.

Syntax *object*.addColumn(*name*)

Parameters

Parameter	Description
object	Instance name of the WddxRecordset object
name	Name of the column to add

Return value None.

Usage Adds a column to every row of the WDDX record set. Initially the new column's values are set to NULL.

Example This example calls the addColumn function:

```
// create a new record set
rs = new WddxRecordset();

// add a new column
rs.addColumn("NewColumn");

// extend the record set by 3 rows
rs.addRows(3);

// set an element in the first row
// newValue is a previously defined variable
rs.setField(0, "NewColumn", newValue);
```

addRows

Description Adds rows to all columns in a WddxRecordset instance.

Syntax *object*.addRows(*n*)

Parameters

Parameter	Description
object	Instance name of the WddxRecordset object
n	Integer; number of rows to add

Return value None.

Usage This function adds the specified number of rows to every column of a WDDX record set. Initially, the row/column values are set to NULL.

Example This example calls the addRows function:

```
// create a new record set
rs = new WddxRecordset();

// add a new column
rs.addColumn("NewColumn");

// extend the record set by 3 rows
rs.addRows(3);

// set an element in the first row
// newValue is a previously defined variable
rs.setField(0, "NewColumn", newValue);
```

getField

Description Returns the element in the specified row/column position.

Syntax `object.getField( row, col )`

Parameters

Parameter	Description
object	Instance name of the WddxRecordset object
row	Integer; zero-based row number of the value to return
col	Integer or string; column of the value to be returned.

Return value Returns the value in the specified row/column position.

Usage Call this function to access a value in a WDDX record set.

Example This example calls the getField function (the variable *r* is a reference to a WddxRecordset instance):

```
for (row = 0; row < nRows; ++row)
{
   o += "<tr>";
   for (i = 0; i < colNames.length; ++i)
   {
      o += "<td>" + r.getField(row, colNames[i]) + "</td>";
   }
   o += "</tr>";
}
```

getRowCount

Description Indicates the number of rows in a WddxRecordset instance.

Syntax `object.getRowCount( )`

Parameters

Parameter	Description
object	Instance name of a WddxRecordset object

Return value Integer. Returns the number of rows in the WddxRecordset instance.

Usage Call this function before a looping construct to determine the number of rows in a record set.

Example This example calls the `getRowCount` function:

```
function dumpWddxRecordset(r)
{
// Get row count
   nRows = r.getRowCount();
...
```

```
  for (row = 0; row < nRows; ++row)
...
```

setField

Description Sets the element in the specified row/column position.

Syntax *object*.setField(*row*, *col*, *value*)

Parameters

Parameter	Description
object	Instance name of a WddxRecordset object
row	Integer; row that contains the element to set
col	Integer or string; the column containing the element to set
value	Value to set

Return value None.

Usage Call this function to set a value in a WddxRecordset instance.

Example This example calls the setField function:

```
// create a new recordset
rs = new WddxRecordset();

// add a new column
rs.addColumn("NewColumn");

// extend the record set by 3 rows
rs.addRows(3);

// set an element in the first row
// newValue is a previously defined variable
rs.setField(0, "NewColumn", newValue);
```

wddxSerialize

Description Serializes a record set.

Syntax *object*.wddxSerialize(*serializer*)

Parameters

Parameter	Description
object	Instance name of the WddxRecordset object
serializer	WddxSerializer instance

Return value Returns a Boolean True if serialization was successful; or False if an error occurs.

Usage This is an internal function; you do not typically call it.

Example This example is from the WddxSerializer `serializeValue` function:

```
...
else if (typeof(obj) == "object")
{
if (obj == null)
{
// Null values become empty strings
this.write("<string></string>");
}
else if (typeof(obj.wddxSerialize) == "function")
{
// Object knows how to serialize itself
bSuccess = obj.wddxSerialize(this);
}
...
```