CFML Language Reference

ColdFusion 4.0 for Windows® NT, Windows 95/98, and Solaris

Allaire Corporation

Copyright Notice

Part number: AA-LNGRF-RK

Contents

CHAPTER 1

Introduction to CFML

ColdFusion is a rapid application development system for professional developers who want to create dynamic Web applications and interactive Web sites. It provides the fastest way to integrate browser, server, and database technologies into powerful Web applications. With ColdFusion, you can build everything from online stores to sophisticated business systems.

Developing applications with ColdFusion does not require coding in a traditional programming language; instead, you build applications by combining standard HTML with a straightforward server-side markup language, the ColdFusion Markup Language (CFML). This manual documents CFML.

Contents

Language Reference Features

The CFML Language Reference provides descriptions, syntax, usage, and code examples for:

- CFML Tags
- CFML Functions
- JavaScript objects used when implementing web distributed data exchange (WDDX)

Using Examples

Each tag and function features a runnable example, using code drawn from the *snippets* directory (found in *webroot*\cfdocs\snippets). The display and usage for these examples differs depending on the delivery mechanism:

- Printed and PDF documentation — Whenever possible, the printed documentation displays a complete example. In some cases, extraneous code has been removed to maintain clarity and to conserve space.
- HTML documentation — Each tag or function displays an examples button that you click to execute that element. The online example displays two frames: one that runs the example, and one that displays the associated code. In some cases, code examples are view-only due to logistic, administrative, or security reasons.

  The examples button uses a relative reference to access the page that runs the code snippet. Snippets will not run if files have been moved or if ColdFusion Server is not running.

  When running examples using ColdFusion Studio on a workstation that isn't running ColdFusion Server, you will need to specify the ColdFusion Server location using the Browse tab on the Settings dialog box.

Using Code Snippets

The snippets directory contains example code for every CFML tag and function. You can browse this directory to find examples of CFML tag and function usage in many different contexts.

Using End Tags

Except where noted in the syntax, CFML tags do not require an end tag. However, all CFML tags (except CFELSE and CFELSEIF) can accept an optional end tag, as follows:

- Shorthand — You can use shorthand notation to include the end tag with the base tag. For example:

```
<CFABORT/>
```

- Explicit specification — You can specify an end tag explicitly. However, there can be no white space between the start and end tags. For example:

```
<CFABORT></CFABORT>
<!--- The following will produce a validation error
<CFABORT>
</CFABORT> --->
```

You do not typically code optional end tags.

Developer Resources

Allaire Corporation is committed to setting the standard for customer support in developer education, technical support, and professional services. Our Web site is designed to give you quick access to the entire range of online resources.

Allaire Developer Services	
Resource	**Description**
Allaire Web site www.allaire.com	General information about Allaire products and services.
Technical Support www.allaire.com/support	Allaire offers a wide range of professional support programs. This page explains all of the available options.
Professional Education www.allaire.com/education	Information about classes, on-site training, and online courses offered by Allaire.
Developer Community www.allaire.com/developer	All of the resources you need to stay on the cutting edge of ColdFusion development, including online discussion groups, Knowledge Base, Component Exchange, Resource Library, technical papers and more.
Allaire Alliance www.allaire.com/partners	The growing network of solution providers, application developers, resellers, and hosting services creating solutions with ColdFusion.

About ColdFusion Documentation

The documentation set is designed to provide support for all components of the ColdFusion development system. Both the print and online versions are organized to allow you to quickly locate the information you need.

Documentation set

The documentation set contains:

Getting Started with ColdFusion — Covers system installation and basic configuration, describes the components of the ColdFusion development system, and introduces the ColdFusion Markup Language (CFML).

Administering ColdFusion Server — Describes configuration options for maximizing performance, managing data sources, setting security levels, and a range of development and site management tasks.

Developing Web Applications with ColdFusion — Presents the fundamentals of ColdFusion application development and deployment, including data sources, user interfaces, and Web technologies. The development tools in ColdFusion Studio are covered in detail.

Advanced ColdFusion Development — Gives an overview of CFML elements such as functions, expressions, arrays, scripting, and XML data exchange. Also discusses custom tags, CF API tags, integrating object technologies, and site management.

CFML Language Reference — Provides the complete syntax, with example code, of all CFML elements.

Quick Reference Card — An online (Acrobat) guide to CFML.

Documentation distribution

The ColdFusion CD-ROM contains the complete document set. The setup program installs the document set by default.

The print manuals are available in Adobe Acrobat (PDF) format from the dochome.htm page in the `/cfdocs` directory of your Web root. If the files are not available locally, you get them from our Web site at http://www.allaire.com/products/COLDFUSION/Documentation.cfm.

You can also access the documentation in HTML from both of these locations.

Reading online documentation

You can open the online documents in a number of ways:

- From your browser, click the ColdFusion Documentation link on the Welcome to ColdFusion page. Each page contains links to other documents and a search window.

- In ColdFusion Studio, click the Help tab in the Resources area to open the help tree. You can expand the list to select topics by title.

Documentation conventions

When reading, please be aware of these formatting cues:

- Code samples, filenames, and URLs are set in a distinct font
- Notes and tips are identified by bold type in the margin
- Bulleted lists present options and features
- Numbered steps indicate procedures
- Menu levels are separated by the greater than (>) sign
- Text for you to type in is set in *italics*
- Sample code that has been removed to save space is indicated by an ellipsis (...)

Contacting Allaire

Corporate headquarters

Allaire Corporation
One Alewife Center
Cambridge, MA 02140

Tel: 617.761.2000 voice
Fax: 617.761.2001 fax

Web site: http://allaire.com

Technical support

Telephone support is available Monday through Friday 8 A.M. to 8 P.M. Eastern time (except holidays)

Toll Free: 888.939.2545 (U.S. and Canada)
Tel: 617.761.2100 (outside U.S. and Canada)

Postings to the ColdFusion Support Forum can be made at any time.

Sales

Toll Free: 888.939.2545
Tel: 617.761.2100
Fax: 617.761.2101
Email: sales@allaire.com

Web: www.allaire.com/store

CHAPTER 2

ColdFusion Tags

This chapter describes each of the tags in the ColdFusion Markup Language (CFML). The introduction contains an alphabetical summary of ColdFusion tags, a list of new tags in ColdFusion 4.0, and a list of tags by category. The remainder of this chapter provides complete descriptions of each tag, listed alphabetically.

Contents

Alphabetical List of ColdFusion Tags

The ColdFusion Markup Language (CFML) consists of a set of tags you use in your ColdFusion pages to interact with data sources, manipulate data, and display output. Using CFML tags is very simple; tag syntax is much like HTML element syntax.

The following table provides brief descriptions of each CFML tag.

CFML Tag Summary	
CFML Tag	**Description**
CFABORT	Stops processing of a ColdFusion page at the tag location.
CFAPPLET	Embeds Java applets in a CFFORM.
CFAPPLICATION	Defines application name, activates client variables.
CFASSOCIATE	Enables sub-tag data to be saved with the base tag.
CFAUTHENTICATE	Authenticates a user and sets the security context for an application.
CFBREAK	Breaks out of a CFML looping construct.
CFCACHE	Caches ColdFusion pages.
CFCOL	Defines table column header, width, alignment, and text.
CFCOLLECTION	Creates and administers Verity collections.
CFCONTENT	Defines the content type and filename of a file to be uploaded to the browser.
CFCOOKIE	Defines and sets cookie variables.
CFDIRECTORY	Performs typical directory-handling tasks from within your ColdFusion application.
CFERROR	Displays customized HTML error pages when errors occur.
CFEXIT	Aborts processing of currently executing CFML custom tag.
CFFILE	Performs typical file-handling tasks from within your ColdFusion application.
CFFORM	Builds an input form and performs client-side input validation.

CFML Tag Summary (Continued)	
CFML Tag	**Description**
CFFTP	Permits FTP file operations.
CFGRID	Used in CFFORM to create a grid control for tabular data.
CFGRIDCOLUMN	Used in CFFORM to define the columns used in a CFGRID.
CFGRIDROW	Used with CFGRID to define a grid row.
CFGRIDUPDATE	Performs updates directly to ODBC data source from edited grid data.
CFHEADER	Generates HTTP headers.
CFHTMLHEAD	Writes text, including HTML, to the HEAD section of a specified page.
CFHTTP	Used to perform GET and POST to upload files or post a form, cookie, query, or CGI variable directly to a specified server.
CFHTTPPARAM	Used with CFHTTP to specify parameters necessary for a CFHTTP POST operation.
CFIF CFELSE CFELSEIF	Used to create IF-THEN-ELSE constructs.
CFINCLUDE	Embeds references to ColdFusion pages.
CFINDEX	Used to create Verity search indexes.
CFINPUT	Used in CFFORM to create input elements such as radio buttons, checkboxes, and text entry boxes.
CFINSERT	Inserts records in an ODBC data source.
CFLDAP	Provides access to LDAP directory servers.
CFLOCATION	Opens a ColdFusion page or HTML file.
CFLOCK	Synchronizes a section of CFML code.
CFLOOP	Repeats a set of instructions based on a set of conditions.
CFMAIL	Assembles and posts an email message.
CFMODULE	Used to invoke a custom tag.
CFOBJECT	Creates and uses COM or CORBA objects.

CFML Tag Summary (Continued)	
CFML Tag	**Description**
CFOUTPUT	Displays output of database query or other operation.
CFPARAM	Defines a parameter and its initial default value.
CFPOP	Retrieves messages from a POP mail server.
CFPROCPARAM	Specifies parameter information for a stored procedure.
CFPROCRESULT	Specifies a result set name that other ColdFusion tags use to access the result set from a stored procedure.
CFQUERY	Passes SQL to a database.
CFREGISTRY	Reads, writes, and deletes keys and values in the system registry.
CFREPORT	Embeds a Crystal Reports report.
CFSCHEDULE	Schedules page execution with option to produce static pages.
CFSCRIPT	Encloses a set of CFScript statements.
CFSEARCH	Executes searches against data indexed in Verity collections using CFINDEX.
CFSELECT	Used in CFFORM to create a drop-down list box form element.
CFSET	Defines a variable.
CFSETTING	Define and control a variety ColdFusion settings.
CFSLIDER	Used in CFFORM to create a slider control element.
CFSTOREDPROC	Specifies database connection information and identifies the stored procedure to be executed.
CFSWITCH/CFCASE/ CFDEFAULTCASE	Evaluates a passed expression and passes control to the CFCASE tag that matches the expression result.
CFTABLE	Builds a table.
CFTEXTINPUT	Places a single-line text entry box in a CFFORM.
CFTHROW	Raises a developer-specified exception.
CFTRANSACTION	Groups CFQUERYs into a single transaction; performs rollback processing.

CFML Tag Summary (Continued)	
CFML Tag	**Description**
CFTREE	Used in CFFORM to create a tree control element.
CFTREEITEM	Used with CFTREE to populate a tree control element in a CFFORM.
CFTRY/CFCATCH	Allows developers to catch and process exceptions in ColdFusion pages.
CFUPDATE	Updates rows in a database data source.
CFWDDX	Serializes and de-serializes CFML data structures to the XML-based WDDX format.

New Tags in ColdFusion 4.0

CFASSOCIATE
CFAUTHENTICATE
CFCACHE
CFCOLLECTION
CFLOCK
CFPROCPARAM
CFPROCRESULT
CFREGISTRY
CFSCRIPT
CFSTOREDPROC
CFSWITCH/CFCASE/CFDEFAULTCASE
CFTHROW
CFTRY/CFCATCH
CFWDDX

Database Manipulation Tags

CFINSERT
CFPROCPARAM
CFPROCRESULT
CFQUERY
CFSTOREDPROC
CFTRANSACTION
CFUPDATE

Data Output Tags

CFCOL
CFCONTENT
CFHEADER
CFOUTPUT
CFTABLE

Variable Manipulation Tags

CFCOOKIE
CFPARAM
CFSCRIPT
CFSET

Flow-Control Tags

CFABORT
CFBREAK
CFEXIT
CFIF/CFELSE/CFELSEIF
CFLOCATION
CFLOOP
CFSWITCH/CFCASE/CFDEFAULTCASE
CFTHROW
CFTRY/CFCATCH

Internet Protocol Tags

CFFTP
CFHTTP
CFHTTPPARAM
CFLDAP
CFMAIL
CFPOP

File Management Tags

CFDIRECTORY

CFFILE

Web Application Framework Tags

CFAPPLICATION

CFAUTHENTICATE

CFERROR

ColdFusion Forms Tags

CFAPPLET

CFFORM

CFGRID

CFGRIDCOLUMN

CFGRIDROW

CFGRIDUPDATE

CFINPUT

CFSELECT

CFSLIDER

CFTEXTINPUT

CFTREE

CFTREEITEM

External System Tags

CFCOLLECTION

CFINDEX

CFOBJECT

CFREPORT

CFSEARCH

Other Tags

CFASSOCIATE
CFCACHE
CFHTMLHEAD
CFINCLUDE
CFLOCK
CFMODULE
CFREGISTRY
CFSCHEDULE
CFSETTING
CFWDDX

CFABORT

The CFABORT tag stops processing of a page at the tag location. ColdFusion simply returns everything that was processed before the CFABORT tag. CFABORT is often used with conditional logic to stop processing a page because of a particular condition.

Syntax

```
<CFABORT SHOWERROR="text">
```

SHOWERROR
Optional. Specify the error you want to display when CFABORT executes. This error message appears in the standard ColdFusion error page.

Usage

When combining CFABORT and CFERROR, remember that CFERROR is meant to redirect output to a specified page. CFABORT is intended to halt processing immediately.

If the CFABORT tag does not contain a SHOWERROR attribute value, processing stops immediately and the page contents are shown all the way up to the line containing the CFABORT tag.

When using CFABORT with SHOWERROR by itself (that is without defining an error page using CFERROR) page processing stops once the CFABORT tag is reached and the message defined in SHOWERROR is displayed to the client.

If you have a page in which you've defined both an error page using CFERROR and a CFABORT tag using the SHOWERROR attribute, ColdFusion redirects output to the error page specified in the CFERROR tag.

Example

```
<!--- this example demonstrates the use of CFABORT
to stop the processing of a CFLOOP.  Note that in
the second example, where CFABORT is used,
the result never appears --->

<HTML>
<HEAD>
<TITLE>CFABORT Example</TITLE>
</HEAD>
<BODY bgcolor=FFFFFF>

<H1>CFABORT Example</H1>

<P>
<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>

<!--- reset the variable and show the use of CFABORT --->
<H3>Example B: Use CFABORT to halt the instruction</H3>

<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>
    <!--- the processing is stopped, and subsequent operations
    are not carried out by the CFAS --->
    <CFELSE>
    <CFSET myVariable = myVariable + 1>
    </CFIF>
</CFLOOP>

<CFOUTPUT>
<P> The value of myVariable after incrementing through the loop
    #counter# times is: #myVariable#
</CFOUTPUT>

</BODY>
</HTML>
```

CFAPPLET

Used in a CFFORM, CFAPPLET allows you to reference custom Java applets that have been previously registered using the ColdFusion Administrator.

To register a Java applet, open the ColdFusion Administrator and click the Applets button.

Syntax

```
<CFAPPLET APPLETSOURCE="applet_name"
    NAME="form_variable_name"
    HEIGHT="pixels"
    WIDTH="pixels"
    VSPACE="pixels"
    HSPACE="pixels"
    ALIGN="alignment"
    NOTSUPPORTED="text"
    param_1="value"
    param_2="value"
    param_n="value">
```

APPLETSOURCE
Required. The name of the registered applet.

NAME
Required. The form variable name for the applet.

HEIGHT
Optional. The height in pixels.

WIDTH
Optional. The width in pixels.

VSPACE
Optional. Space above and below applet in pixels.

HSPACE
Optional. Space on each side of the applet in pixels.

ALIGN
Optional. Alignment. Valid entries are:

- Left
- Right
- Bottom
- Top
- TextTop
- Middle

- AbsMiddle
- Baseline
- AbsBottom

NOTSUPPORTED
Optional. The text you want to display if the page containing 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>"
```

By default, if no message is specified, the following message appears:

```
<B>Browser must support Java to <BR>
view ColdFusion Java Applets!</B>
```

param*n*
Optional. The valid name of a registered parameter for the applet. Specify a parameter only if you want to override parameter values already defined for the applet using the ColdFusion Administrator.

Usage

Since Java applets must be pre-registered, the CFAPPLET tag can be very simple, taking the default parameter values as they were registered in the ColdFusion Administrator. You can also override parameters by invoking them directly in the CFAPPLET tag.

Example

```
<!--- This example shows the use of CFAPPLET --->
<HTML>
<HEAD>
<TITLE>CFAPPLET Example</TITLE>
</HEAD>

<BODY>
<H3>CFAPPLET Example</H3>

<P>Used in a CFFORM, CFAPPLET allows you to reference
custom Java applets that have been previously registered
using the ColdFusion Administrator.
<P>To register a Java applet, open the ColdFusion Administrator
and click the "Applets" link under the "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="copytext.cfm">
<CFAPPLET appletsource="copytext" name="copytext">
</CFFORM>

</BODY>
</HTML>
```

CFAPPLICATION

Defines scoping for a ColdFusion application, enables or disables storing client variables, and specifies a client variable storage mechanism. By default, client variables are disabled. Also used to enable session variables and to set timeouts for both session and application variables. Session and application variables are stored in memory.

Syntax

```
<CFAPPLICATION NAME="Name"
    CLIENTMANAGEMENT="Yes/No"
    CLIENTSTORAGE="Storage Type"
    SETCLIENTCOOKIES="Yes/No"
    SESSIONMANAGEMENT="Yes/No"
    SESSIONTIMEOUT=#CreateTimeSpan(days, hours,
      minutes, seconds)#
    APPLICATIONTIMEOUT=#CreateTimeSpan(days, hours,
      minutes, seconds)#>
```

NAME

The name you want to give your application. This name can be up to 64 characters long. Required for application variables to work. Optional for client and session variables.

CLIENTMANAGEMENT

Optional. Yes or No. Enables client variables. Default is No.

CLIENTSTORAGE

Optional. Specifies the mechanism for storing client variables:

- *datasourcename* — ColdFusion stores client variables in the specified ODBC or native data source. To use this option you must create a client variable storage repository using the Variables page of the ColdFusion Administrator.
- Registry — ColdFusion stores client variables in the system registry. This is the default.
- Cookie — ColdFusion stores client variables on the client machine in a cookie. Storing client data in a cookie is scalable to large numbers of clients, but this storage mechanism has some limitations. Chief among them is that if the client turns off cookies in the browser, client variables won't work.

SETCLIENTCOOKIES

Optional. Yes or No. Yes enables client cookies. Default is Yes.

If you set this attribute to "NO", ColdFusion does not automatically send the CFID and CFTOKEN cookies to the client browser; you must manually code CFID and CFTOKEN on the URL for every page that uses Session or Client variables.

SESSIONMANAGEMENT

Optional. Yes or No. Yes enables session variables. Default is No.

SESSIONTIMEOUT
Optional. Enter the CreateTimeSpan function and the values you want in days, hours, minutes, and seconds, separated by commas to specify the lifespan of any session variables that are set. The default value is specified in the Variables page of the ColdFusion Administrator.

APPLICATIONTIMEOUT
Optional. Enter the CreateTimeSpan function and the values you want in days, hours, minutes, and seconds, separated by commas to specify the lifespan of any application variables that are set. The default value is specified in the Variables page of the ColdFusion Administrator.

Usage

CFAPPLICATION is typically used in the `Application.cfm` file to set defaults for a specific ColdFusion application.

CFAPPLICATION enables application variables unless they have been disabled in the ColdFusion Administrator. Using the SESSIONMANAGEMENT attribute to enable session variables is also overridden by the Administrator. See *Administering ColdFusion Server* for information about the ColdFusion Administrator.

If you are running ColdFusion on a cluster, you must specify either Cookie or a data source name for CLIENTSTORAGE; you cannot specify Registry.

Example

```
<!--- This example illustrates CFAPPLICATION --->
<!-- Begin Application -->
<!---
    MODULE:   application.cfm
    PURPOSE:  application.cfm file for our sample
    CREATED:  today's date
    AUTHOR:   the author
    COPYRIGHT: (c) 1998 the author for a company
    CHANGE HISTORY:
--->
<!--- name application, set session variables to on --->
<CFAPPLICATION NAME='GetLeadApp' SESSIONMANAGEMENT='Yes'>
<!--- set datasource for this application --->
<CFSET dsn = 'my_dsn'>
<!--- set global error handling for this application --->
<CFERROR TYPE='REQUEST' TEMPLATE='request_err.cfm'
MAILTO='webmaster@mysite.com'>
<CFERROR TYPE='VALIDATION' TEMPLATE='val_err.cfm'
MAILTO='webmaster@mysite.com'>

<!--- set some global variables for this application to
be triggered on every template --->
<CFSET MainPage = 'default.cfm'>
<CFSET session.current_location = 'Davis, Porter, Alewife'>
<CFSET sm_location = 'dpa'>
<CFSET current_page = '#cgi.path_info#?#cgi.query_string#'>
<!-- End Application -->
```

CFASSOCIATE

The CFASSOCIATE tag allows sub-tag data to be saved with the base tag. This applies to custom tags only.

Syntax

```
<CFASSOCIATE BASETAG="tagname"
    DATACOLLECTION="collectionname">
```

BASETAG
Specifies the name of the base tag.

DATACOLLECTION
Optional. Specifies the name of the structure in which the base tag stores sub-tag data. The default is AssocAttribs.

Usage

Call this tag within a sub-tag to save sub-tag data in the base tag.

ColdFusion saves sub-tag attributes in a structure whose default name is AssocAttribs. Use the DataCollection attribute to specify a non-default structure name. Specify a non-default structure name when the base tag can have multiple sub tags and you want to segregate sub-tag attributes.

Example

```
<!--- Find the context --->
<cfif thisTag.executionMode is "start">
  <!--- Associate attributes
      This code occurs in a custom tag's
      sub tag. --->
  <CFASSOCIATE BASETAG="CF_TAGBASE">

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

CFAUTHENTICATE

The CFAUTHENTICATE tag authenticates a user, setting a security context for the application.

Syntax

```
<CFAUTHENTICATE SECURITYCONTEXT="context"
    USERNAME="user ID"
    PASSWORD="password"
    SETCOOKIE="yes/no"
    THROWONFAILURE="yes/no">
```

SECURITYCONTEXT
Required. Security context with which the specified user is authenticated. This context must have been previously defined in the security system.

USERNAME
Required. User to be authenticated.

PASSWORD
Required. Password for the user.

SETCOOKIE
Optional. Default is Yes. Indicates whether ColdFusion sets a cookie to contain authentication information. This cookie is encrypted and its contents include user name, security context, browser remote address, and the HTTP user agent.

THROWONFAILURE
Optional. Default is Yes. Indicates whether ColdFusion throws an exception (of type SECURITY) if authentication fails.

Usage

Code this tag in the Application.cfm file to set a security context for your application.

Call the IsAuthenticated function to determine if the user has been authenticated. If you specify No for SETCOOKIE, you must call CFAUTHENTICATE for every page in the application (perhaps in an Application.cfm file).

If you specify THROWONFAILURE=Yes, you can enclose CFAUTHENTICATE in a CFTRY/CFCATCH block to handle possible exceptions programmatically.

Example

```
<!--- This example shows the use of CFAUTHENTICATE
in an Application.cfm file --->
<CFIF NOT IsAuthenticated()>
  <CFTRY>
  <CFAUTHENTICATE SECURITYCONTEXT="Allaire" USERNAME=#user#
    PASSWORD=#pwd#>
  <CFCATCH TYPE="Security">
   <!--- the message to display --->
   <H3>Authentication error</H3>
    <CFOUTPUT>
    <!--- Display the message. Alternatively, you might place
```

```
      code here to define the user to the security domain. --->
    <P>#CFCATCH.message#
    </CFOUTPUT>
  </CFCATCH>
  </CFTRY>
</CFIF>
<CFAPPLICATION NAME="Personnel">
...
```

CFBREAK

Used to break out of a CFLOOP. See Breaking out of a loop, later in this chapter, for more information.

Syntax

```
<CFBREAK>
```

Example

```
<!--- This example shows the use of CFBREAK to exit
a loop when a condition is met --->

<!--- select a list of courses and use CFLOOP to find a condition
and then break the loop --->
<CFQUERY NAME="GetCourses" DATASOURCE="cfsnippets">
SELECT *
FROM courses
ORDER by Number
</CFQUERY>
<HTML>
<HEAD>
<TITLE>
CFBREAK Example
</TITLE>
</HEAD>
<BODY bgcolor=silver>

<H1>CFBREAK Example</H1>
<P>This example uses CFLOOP to cycle through a query to find a desired
value. (In our example, a list of values corresponding to courses in the
cfsnippets datasource).
When the conditions of the query are met, CFBREAK stops the loop.
...
<!--- loop through the query until desired value is found,
   then use CFBREAK to exit the query --->
<CFLOOP QUERY="GetCourses">
    <CFIF GetCourses.Number is "#form.courseNum#">
    <CFOUTPUT>
    <H4>Your Desired Course was found:</H4>
    <PRE>#Number##Descript#</PRE></CFOUTPUT>
    <CFBREAK>
    <CFELSE>
        <BR>Searching...
    </CFIF>
</CFLOOP>
</CFIF>

</BODY>
</HTML>
```

CFCACHE

CFCACHE allows you to speed up pages considerably in cases where the dynamic content doesn't need to be retrieved each time a user accesses the page. To accomplish this, it creates temporary files that contain the static HTML returned from a particular run of the ColdFusion page.

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

Syntax

```
<CFCACHE
    ACTION="action"
    PROTOCOL="protocol name"
    TIMEOUT="timeout date-time"
    DIRECTORY="directory name"
    EXPIREURL="wildcarded URL reference">
```

ACTION

Optional. Specifies one of the following:

- CACHE — Cache the page. The default is CACHE.
- FLUSH — Refresh the cached page. If you specify FLUSH, you can also specify the DIRECTORY and EXPIREURL attributes.

PROTOCOL

Optional. Specifies the protocol used to create pages from cache. Specify either HTTP:// or HTTPS://. The default is HTTP://.

TIMEOUT

Optional. DateTime that specifies the oldest acceptable cached page. If the cached page is older than the specified datetime, ColdFusion refreshes the page. By default, ColdFusion uses all cached pages. For example, if you want a cached file to be no older than 4 hours, code the following:

```
<CFCACHE TIMEOUT="#DateAdd("h", "-4", Now() )#">
```

DIRECTORY

Optional. Used with ACTION=FLUSH. Specifies the fully qualified path of a directory containing the cfcache.map to be used when ACTION=FLUSH. The default is the directory of the current page.

EXPIREURL

Optional. Used with ACTION=FLUSH. EXPIREURL takes a wildcarded URL reference that ColdFusion matches against all mappings in the cfcache.map file. The default is to flush all mappings. For example, "foo.cfm" matches "foo.cfm"; "foo.cfm?*" matches "foo.cfm?x=5" and "foo.cfm?x=9".

Usage

In its simplest form, all you need to do is code <CFCACHE> at the top of a page to be cached:

In addition to the cached files themselves, CFCACHE uses a mapping file to control caching. This file is stored in the directory of the files being cached. It is named cfcache.map and uses a format similar to a Windows INI file. The mapping of a URL with parameters is stored as follows. Assume a directory "c:\InetPub\wwwroot\dir1" that has a CFM file called "foo.cfm", which can be invoked with or without URL parameters. The cfcache.map file entries for foo.cfm will look like this:

```
 [foo.cfm]
Mapping=C:\InetPub\wwwroot\dir1\CFCBD.tmp
SourceTimeStamp=08/31/1998 08:59:04 AM

[foo.cfm?x=5]
Mapping=C:\InetPub\wwwroot\dir1\CFCBE.tmp
SourceTimeStamp=08/31/1998 08:59:04 AM

[foo.cfm?x=9]
Mapping=C:\InetPub\wwwroot\dir1\CFCBF.tmp
SourceTimeStamp=08/31/1998 08:59:04 AM
```

The cfcache.map file in a given directory stores mappings for that directory only. Any time the timestamp of the underlying page changes, ColdFusion updates the cache file for that URL only. ColdFusion uses the SourceTimeStamp field to determine if the currently cached file is up to date or needs to be rebuilt.

You can refresh the cache in the following ways:

- TIMEOUT attribute — ColdFusion tests the timestamp of the cached file against the TIMEOUT attribute. If the cached file's timestamp is older than TIMEOUT, the old file is deleted and a new one created. You can use fixed dates if necessary, but it's preferable to use relative dates. This is the preferred technique and it works for seconds, hours, days, weeks, years, etc.
- ACTION=FLUSH — You use ACTION=FLUSH to force the clean up of cached files. It can take two attributes, DIRECTORY and EXPIREURL.
- Manually — Manually or programmatically (using CFFILE) delete the .tmp files. This is not recommended.

Note the following regarding CFCACHE:

- CFCACHE requires that ColdFusion Server "simultaneous requests" be greater than 1. When a cache file is generated, the requested page requires two connections to satisfy the request. When a cached file is found, only one request is required.
- Debug settings have no effect on CFCACHE unless the template explicitly turns it on. When generating a cached file, CFCACHE uses `<CFSETTING SHOWDEBUGOUTPUT="NO">`.
- ColdFusion does not cache pages that are dependent on anything other than URL parameters.
- To use CFCACHE with secure sockets, specify `PROTOCOL="https://"`.
- If a template returns an error for any reason, the error page gets cached.

Example

```
<!--- This example will produce as many cached files as there
      are possible URL parameter permutations. --->
<CFCACHE TIMEOUT="#DateAdd("h", "-4", Now() )#">
<HTML>
<HEAD>
<TITLE>CFCACHE Example</TITLE>
</HEAD>
<BODY>
<H1>CFCACHE Example</H1>

<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>
</BODY>
</HTML>
```

CFCOL

Defines table column header, width, alignment, and text. Always used inside a CFTABLE.

Syntax

```
<CFCOL HEADER="text"
    WIDTH="number"
    ALIGN="position"
    TEXT="text">
```

HEADER
The text to use for the column's header.

WIDTH
The width of the column in characters (the default is 20). If the length of the data displayed exceeds the width value, the data is truncated to fit.

ALIGN
Column alignment, Left, Right, or Center.

TEXT
Double-quote delimited text that determines what displays in the column. The rules for the text attribute are identical to the rules for CFOUTPUT sections, meaning that it can consist of a combination of literal text, HTML tags, and query record set field references. This means you can embed hyperlinks, image references, and even input controls within table columns.

Example

```
<!--- This example shows the use of CFCOL and CFTABLE
to align information returned from a query --->

<!--- this query selects employee information from the
cfsnippets datasource --->
<CFQUERY NAME="GetEmployees" DATASOURCE="cfsnippets">
SELECT Emp_ID, FirstName, LastName, EMail, Phone, Department
FROM   Employees
</CFQUERY>

<HTML>
<HEAD>
<TITLE>
CFCOL Example
</TITLE>
</HEAD>

<BODY>
<H3>CFCOL Example</H3>

<!--- Note the use of the HTMLTABLE attribute to display the
CFTABLE as an HTML table, rather simply as PRE formatted information --->
<CFTABLE QUERY="GetEmployees" STARTROW="1" COLSPACING="3" HTMLTABLE>
```

```
<!--- each CFCOL tag sets the width of a column in the table,
as well as specifying the header information and the text/CFML
with which to fill 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>

</BODY>
</HTML>
```

CFCOLLECTION

The CFCOLLECTION tag allows you to create and administer Verity collections.

Syntax

```
<CFCOLLECTION ACTION="action"
    COLLECTION="collection"
    PATH="implementation directory"
    LANGUAGE="language">
```

ACTION
Required. Specifies the action to perform:

- CREATE — Creates a new collection using the specified path and optionally specified language.
- REPAIR — Fixes data corruption in the collection.
- DELETE — Destroys the collection.
- OPTIMIZE — Purges and reorganizes data for efficiency.
- MAP — Assigns an alias to an existing Verity collection.

COLLECTION
Required. Specifies a collection name.

PATH
Required for CREATE. Specifies a path to the Verity collection.

LANGUAGE
Optional for CREATE. To use the LANGUAGE attribute you must have the ColdFusion International Search Pack installed. Valid entries are:

- English (default)
- German
- Finnish
- French
- Danish
- Dutch
- Italian
- Norwegian
- Portuguese
- Spanish
- Swedish

Usage CFCOLLECTION works at the collection level only. To add content to a collection, use CFINDEX.

Note the following regarding mapped collections:

- Mapping allows you to assign an alias to a Verity collection created by a tool other than ColdFusion.
- The ACTION, COLLECTION, and PATH attributes are required.
- The path must point to a valid Verity collection; mapping does not validate the path.
- Deleting a mapped collection unregisters the alias; the base collection is not deleted.

Example

```
<!--- This example shows the basic functionality
of the CFCOLLECTION tag (create, repair, optimize, delete) --->
<HTML>
<HEAD>
    <TITLE>CFCOLLECTION</TITLE>
</HEAD>
<BODY bgcolor=silver>
<H3>CFCOLLECTION</h3>

<!--- see if a collection name has been specificied ... --->
<CFIF IsDefined("form.CollectionName") AND
IsDefined("form.CollectionAction")>
    <CFIF form.CollectionName is not "">
        <CFOUTPUT>
        <CFSWITCH EXPRESSION=#FORM.CollectionAction#>
          <CFCASE VALUE="Create">
            <CFCOLLECTION ACTION="CREATE"
             COLLECTION="#FORM.CollectionName#"
             PATH="C:\CFUSION\Verity\Collections\">
            <H3>Collection created.</H3>
          </CFCASE>
          <CFCASE VALUE="Repair">
            <CFCOLLECTION ACTION="REPAIR"
             COLLECTION="#FORM.CollectionName#">
            <H3>Collection repaired.</H3>
          </CFCASE>
          <CFCASE VALUE="Optimize">
            <CFCOLLECTION ACTION="OPTIMIZE"
             COLLECTION="#FORM.CollectionName#">
            <H3>Collection optimized.</H3>
          </CFCASE>
          <CFCASE VALUE="Delete">
            <CFCOLLECTION ACTION="DELETE"
             COLLECTION="#FORM.CollectionName#">
           <H3>Collection deleted.</H3>
          </CFCASE>
        </CFSWITCH>
...
```

CFCONTENT

Defines the MIME type returned by the current page. Optionally allows you to specify the name of a file to be returned with the page.

ColdFusion administrators can disable the CFCONTENT tag in the ColdFusion Administrator Basic Security page.

Syntax

```
<CFCONTENT TYPE="file_type"
    DELETEFILE="Yes/No"
    FILE="filename">
```

TYPE
Required. Defines the File/ MIME content type returned by the current page.

DELETEFILE
Optional. Yes or No. Yes deletes the file after the upload operation. Defaults to No.

FILE
Optional. Denotes the name of the file being retrieved.

Example

```
<!---  This example shows the use of CFCONTENT to return the
contents of the CF Documentation page dynamically to the browser.
You may need to change the path and/or drive letter.
(graphics will not display) --->
<HTML>
<HEAD>
<TITLE>
CFCONTENT Example
</TITLE>
</HEAD>

<BODY>
<H3>CFCONTENT Example</H3>

<!--- Files may be set to delete after downloading,
allowing for the posting of changing content. --->
<CFCONTENT TYPE="text/html"
   FILE="c:\inetpub\wwwroot\cfdocs\main.htm" DELETEFILE="No">

</BODY>
</HTML>
```

CFCOOKIE

Defines cookie variables, including expiration and security options.

Syntax

```
<CFCOOKIE NAME="cookie_name"
    VALUE="text"
    EXPIRES="period"
    SECURE="Yes/No"
    PATH="urls"
    DOMAIN=".domain">
```

NAME
Required. The name of the cookie variable.

VALUE
Optional. The value assigned to the cookie variable.

EXPIRES
Optional. Schedules the expiration of a cookie variable. Can be specified as a date (as in, 10/09/97), number of days (as in, 10, 100), NOW, or NEVER. Using NOW effectively deletes the cookie from the client's browser.

SECURE
Optional. Indicates the variable has to transmit securely. If the browser does not support Secure Socket Layer (SSL) security, the cookie is not sent.

PATH
Optional. Specifies the subset of URLs within the specified domain to which this cookie applies:

`PATH="/services/login"`
Separate multiple entries with a semicolon (`;`).

DOMAIN
Specifies the domain for which the cookie is valid and to which the cookie content can be sent. An explicitly specified domain must always start with a dot. This can be a subdomain, in which case the valid domains will be any domain names ending in this string.

For domain names ending in country codes (such as `.jp`, `.us`), the subdomain specification must contain at least three periods, for example, `.mongo.stateu.us`. In the case of special top level domains, only two periods are needed, as in `.allaire.com`.

When specifying a PATH value, you must include a valid DOMAIN.

Separate multiple entries with a semicolon (`;`).

Usage Cookies written with CFCOOKIE do not get written to the `cookies.txt` file until the browser session ends. Until the browser is closed, the cookie resides in memory. If you

do not have an EXPIRES attribute in a CFCOOKIE, the cookie set exists only as long as the client browser is open. When the browser is closed, the cookie expires. It is never written to the `cookies.txt` file.

Example

```
<!--- This example shows how to set a CFCOOKIE variable,
and also how to delete that variable --->

<!--- First select a group of users who have entered
comments into the sample database --->
<CFQUERY NAME="GetAolUser" DATASOURCE="cfsnippets">
SELECT EMail, FromUser, Subject, Posted
FROM   Comments
</CFQUERY>

<HTML>
<HEAD>
<TITLE>
CFCOOKIE Example
</TITLE>
</HEAD>

<BODY bgcolor=silver>
<H3>CFCOOKIE Example</H3>

<!--- if the URL variable delcookie exists,
set the cookie's expiration date to NOW --->
<CFIF IsDefined("url.delcookie") is True>
    <CFCOOKIE NAME="TimeVisited"
    VALUE="#Now()#"
    EXPIRES="NOW">
<CFELSE>
<!--- Otherwise, loop through the list of visitors,
and stop when you match the string aol.com in the
visitor's email 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>

</BODY>
</HTML>
```

CFDIRECTORY

Use the CFDIRECTORY tag to handle all interactions with directories.

Note ColdFusion administrators can disable CFDIRECTORY processing in the ColdFusion Administrator Basic Security page.

Syntax

```
<CFDIRECTORY ACTION="directory action"
    DIRECTORY="directory name"
    NAME="query name"
    FILTER="list filter"
    MODE="permission"
    SORT="sort specification"
    NEWDIRECTORY="new directory name">
```

ACTION
Optional. Defines the action to be taken with directory(ies) specified in DIRECTORY. Valid entries are:

- List (default)
- Create
- Delete
- Rename.

DIRECTORY
Required for all ACTIONs. The name of the directory you want the action to be performed against.

NAME
Required for ACTION="List". Ignored for all other actions. Name of output query for directory listing.

FILTER
Optional for ACTION="List". Ignored for all other actions. File extension filter to be applied to returned names, for example: `*.cfm`. Only one mask filter can be applied at a time.

MODE
Optional. Used with ACTION="Create" to define the permissions for a directory in Solaris. Ignored in Windows. Valid entries correspond to the octal values (not symbolic) of the UNIX `chmod` command. Permissions are assigned for owner, group, and other, respectively. For example:

`MODE=644`
Assigns all, owner read/write permission, group and other read/write permissions.

`MODE=666`

Assigns read/write permissions for owner, group, and other.

`MODE=777`
Assigns read, write, and execute permissions for all.

SORT
Optional for ACTION="List". Ignored for all other actions. List of query columns to sort directory listing by. Any combination of columns from query output can be specified in comma separated list. ASC or DESC can be specified as qualifiers for column names. ASC is the default. For example:

```
SORT="dirname ASC, filename2 DESC, size, datelastmodified"
```

NEWDIRECTORY
Required for ACTION="Rename". Ignored for all other actions. The new name of the directory specified in the DIRECTORY attribute.

ACTION=LIST

When using the ACTION=LIST, CFDIRECTORY returns five result columns you can reference in your CFOUTPUT:

- Name – Directory entry name.
- Size – Size of directory entry.
- Type – File type: File or Dir for File or Directory.
- DateLastModified – Date an entry was last modified.
- Attributes – File attributes, if applicable.
- Mode – (Solaris only) The octal value representing the permissions setting for the specified directory. For information about octal values, refer to the UNIX man pages for the `chmod` shell command.

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

```
#mydirectory.Name#
#mydirectory.Size#
#mydirectory.Type#
#mydirectory.DateLastModified#
#mydirectory.Attributes#
#mydirectory.Mode#
```

Example

```
<!--- This example shows the use of CFDIRECTORY
to display the contents of the snippets directory in
CFDOCS --->
<HTML>
<HEAD>
<TITLE>
CFDIRECTORY Example
</TITLE>
</HEAD>
```

```
<BODY>
<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="c:\inetpub\wwwroot\cfdocs\snippets"
    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>

</BODY>
</HTML>
```

CFERROR

Provides the ability to display customized HTML pages when errors occur. This allows you to maintain a consistent look and feel within your application even when errors occur.

Syntax

```
<CFERROR TYPE="err_type"
    TEMPLATE="path"
    MAILTO="email_address">
```

TYPE
The type of error that this custom error page is designed to handle:

- Specify Request to handle errors that occur during the processing of a page. Request is the default.
- Specify Validation to handle data input validation errors that occur when submitting a form. A validation error handler is only useful if placed inside the `Application.cfm` file.

TEMPLATE
The relative path to the file containing the error page. Note that this page can only include the error variables described below and cannot include CFML tags.

MAILTO
The email address of the administrator who should be notified of the error. This value is available to your custom error page using the MailTo property of the error object, such as #Error.MailTo#.

Usage

The CFERROR tag is normally used to customize the error messages for all the pages in an application. As a result, you generally embed it in the `Application.cfm` file. For more information about the `Application.cfm` file, refer to *Developing Web Applications with ColdFusion*.

To help ensure that error pages display successfully, pages you specify with CFERROR should not be encrypted with the `cfcrypt` utility.

CFERROR Error Page Variables

The error page specified in the TEMPLATE attribute of the CFERROR tag may contain one or more error variables, which will be substituted by ColdFusion when an error is displayed. Beyond the specification of these variables, no other ColdFusion tags, such as CFIF, CFSET, and so on, may be used in the error page.

Error pages where TYPE="Request"

Error variables available when CFERROR uses TYPE="Request" are as follows:

Custom Error Pages where TYPE="Request"	
Error Variable	**Description**
Error.Diagnostics	Detailed error diagnostics from ColdFusion Server.
Error.MailTo	Email address of administrator who should be notified (corresponds to the value set in the MAILTO attribute of CFERROR).
Error.DateTime	Date and time when the error occurred.
Error.Browser	Browser that was running when the error occurred.
Error.RemoteAddress	IP address of the remote client.
Error.HTTPReferer	Page from which the client accessed the link to the page where the error occurred.
Error.Template	Page being executed when the error occurred.
Error.QueryString	URL query string of the client's request.

Error pages where TYPE="Validation"

Error variables available when CFERROR uses TYPE="Validation" are as follows:

Custom Error Pages where TYPE="Validation"	
Error Variable	**Description**
Error.ValidationHeader	Text for header of validation message.
Error.InvalidFields	Unordered list of validation errors that occurred.
Error.ValidationFooter	Text for footer of validation message.

Example

```
<!--- This example shows the use of CFERROR --->
<HTML>
<HEAD>
<TITLE>CFERROR Example</TITLE>
</HEAD>

<BODY>
<H3>CFERROR Example</H3>
```

```
<P>CFERROR provides the ability to display customized
HTML pages when errors occur. This allows you to
maintain a consistent look and feel within your
application even when errors occur. Note that no CFML
can be displayed in the resulting templates except
for the specialized error variables.
<P>CFTRY/CFCATCH provides a more interactive way to
handle your CF errors within a CF template than CFERROR,
but CFERROR is still a good safeguard against general
errors.
<P>You can also use CFERROR within the Application.cfm
to specify error handling responsibilities for an entire
application.

<!--- Example of CFERROR call within a template --->
<CFERROR TYPE="REQUEST"
     TEMPLATE="request_err.cfm"
     MAILTO="admin@mywebsite.com">

<!--- Example of the template 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>
</BODY>
</HTML>        --->

</BODY>
</HTML>
```

CFEXIT

CFEXIT can be used to:

- Abort the processing of the currently executing CFML custom tag.
- Exit the template within the currently executing CFML custom tag.
- Reexecute a section of code within the currently executing CFML custom tag.

Syntax

```
<CFEXIT METHOD="method">
```

METHOD
Optional. Specifies one of the following:

- ExitTag (default) — Aborts processing of the currently executing CFML custom tag.
- ExitTemplate — Exits the template of the currently executing CFML custom tag.
- Loop — Reexecutes the body of the currently executing CFML custom tag.

Usage

If a CFEXIT tag is encountered outside the context of a custom tag, for example in the base page or an included page, the tag acts exactly like CFABORT. CFEXIT can help simplify error checking and validation logic in custom tags.

CFEXIT behaves differently depending on location and execution mode:

METHOD attribute	Location of CFEXIT call	Behavior
ExitTag	Base template	Terminate processing
	Execution mode = Start	Continue after end tag
	Execution mode = End	Continue after end tag
ExitTemplate	Base template	Terminate processing
	Execution mode = Start	Continue from first child in body
	Execution mode = End	Continue after end tag
Loop	Base template	Error

METHOD attribute	Location of CFEXIT call	Behavior
Loop (contd.)	Execution mode = Start	Error
	Execution mode = End	Continue from first child in body

Example

```
<!--- This example shows the use of CFEXIT, and
is a read-only example --->
<HTML>
<HEAD>
<TITLE>CFEXIT Example</TITLE>
</HEAD>

<BODY>
<H3>CFEXIT Example</H3>

<P>CFEXIT can be used to abort the processing of the
currently executing CFML custom tag.  Execution will resume
immediately 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 of 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 inside a CFML custom tag, as
follows: --->
<!--- Place this code (uncomment the appropriate
sections) inside 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 --->

<!--- And place this code inside 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 of a custom tag, it functions
like a CFABORT.  For example, the text after this message
will not be processed:
<CFEXIT>
<P>This text will not be executed due to the existence of
the CFEXIT tag above it.

</BODY>
</HTML>
```

CFFILE

Use the CFILE tag to handle all interactions with files. The attributes you use with CFILE depend on the value of the ACTION attribute. For example, if the ACTION is "Write," ColdFusion expects the attributes associated with writing a text file. See the individual CFFILE topics below for details about which attributes apply to which ACTIONs.

Note ColdFusion administrators can disable CFFILE processing in the ColdFusion Administrator Basic Security page.

CFFILE topics

- CFFILE ACTION="Upload"
- CFFILE ACTION="Move"
- CFFILE ACTION="Rename"
- CFFILE ACTION="Copy"
- CFFILE ACTION="Delete"
- CFFILE ACTION="Read"
- CFFILE ACTION="Write"
- CFFILE ACTION="Append"

CFFILE ACTION attributes

Depending on the value you assign to the ACTION attribute of CFFILE, there are several additional attributes you can set. This table shows which attributes you can use with each CFFILE ACTION.

Attributes Used with CFFILE ACTIONs	
ACTION	**Attributes**
Upload	ACCEPT DESTINATION FILEFIELD NAMECONFLICT MODE ATTRIBUTES
Move	SOURCE DESTINATION ATTRIBUTES

Attributes Used with CFFILE ACTIONs (Continued)	
ACTION	**Attributes**
Rename	SOURCE DESTINATION ATTRIBUTES
Copy	SOURCE DESTINATION ATTRIBUTES
Delete	FILE
Read	FILE VARIABLE
Write	OUTPUT FILE MODE ATTRIBUTES
Append	OUTPUT FILE MODE ATTRIBUTES

Sections that follow describe these values and attributes in greater detail.

CFFILE ACTION="Upload"

Use CFFILE with the Upload action to upload a file specified in a form field to a directory on the Web server.

Note that the MODE attribute applies only to ColdFusion on Solaris.

Syntax

```
<CFFILE ACTION="Upload"
    FILEFIELD="formfield"
    DESTINATION="directory"
    NAMECONFLICT="behavior"
    ACCEPT="file_extension"
    MODE="permission"
    ATTRIBUTES="file_attributes">
```

FILEFIELD

Required. The name of the form field that was used to select the file.

Note: Do not use pound signs (#) to specify the field name.

DESTINATION
Required. The destination directory on the Web server where the file should be saved.

Note: The directory does not need to be beneath the root of the Web server document directory.

NAMECONFLICT
Optional. Default is error. Determines how the file should be handled if its name conflicts with the name of a file that already exists in the directory. Valid entries are:

- Error — Default. The file will not be saved, and ColdFusion will stop processing the page and return an error.
- Skip — Neither saves the file nor throws an error. This setting is intended to allow custom behavior based on inspection of FILE properties.
- Overwrite — Replaces an existing file if it shares the same name as the CFFILE destination.
- MakeUnique — Automatically generates a unique filename for the upload. This name will be stored in the FILE object variable "ServerFile." You can use this variable to record what name was used when the file was saved.

ACCEPT
Optional. Use to limit what types of files will be accepted. Enter one or more MIME types, each separated by comma, of the file types you want to accept. For example, to allow uploads of GIF and Microsoft Word files, enter:

```
ACCEPT="image/gif, application/msword"
```

Note: The browser uses the file extension to determine file type.

MODE
Optional. Defines permissions for an uploaded file in Solaris. Ignored in Windows. Valid entries correspond to the octal values (not symbolic) of the UNIX chmod command. Permissions are assigned for owner, group, and other, respectively. For example:

```
MODE=644
```

Assigns the owner read/write permissions and group/other read permission.

```
MODE=666
```

Assigns read/write permissions for owner, group, and other.

```
MODE=777
```

Assigns read, write, and execute permissions for all.

ATTRIBUTES
Optional. A comma-delimited list of file attributes to be set on the file being uploaded. The following file attributes are supported:

- ReadOnly
- Temporary

- Archive
- Hidden
- System
- Normal

If ATTRIBUTES is not used, the file's attributes are maintained. If Normal is specified as well as any other attributes, Normal is overridden by whatever other attribute is specified.

Individual attributes must be specified explicitly. For example, if you specify just the ReadOnly attribute, all other existing attributes are overwritten.

Examples The following example will create a unique filename if there is a name conflict when the file is uploaded:

```
<CFFILE ACTION="Upload"
    FILEFIELD="FileContents"
    DESTINATION="c:\web\uploads\"
    ACCEPT="text/html"
    NAMECONFLICT="MakeUnique">
```

UNIX Examples This following three examples show the use of the MODE attribute for UNIX. The first example creates the file `/tmp/foo` with permissions defined as `rw-r-r--` (owner=read/write, group/other=read).

```
<CFFILE ACTION="Write"
    FILE="/tmp/foo"
    MODE=644>
```

This example appends to the specified file and makes permissions read/write (rw) for all.

```
<CFFILE ACTION="Append"
    DESTINATION="/home/tomj/testing.txt"
    MODE=666
    OUTPUT="Is this a test?">
```

The next example uploads a file and gives it `rwx-rw-rw` permissions (owner/group/other=read/write).

```
CFFILE ACTION="Upload"
    FILEFIELD="fieldname"
    DESTINATION="/tmp/program.exe"
    MODE=755>
```

CFFILE ACTION="Move"

The CFFILE MOVE action can be used to move a file from one location on the server to another.

Syntax

```
<CFFILE ACTION="Move"
```

```
    SOURCE="file_name"
    DESTINATION="directory_name"
    ATTRIBUTES="file_attributes">
```

SOURCE
Required. The file to move (with directory location).

DESTINATION
Required. The directory to which the file will be moved. For UNIX, a trailing slash must be included in the target when moving a directory. Backward slashes (\), used in Windows are interpreted as forward slashes (/) in UNIX, and vice versa.

ATTRIBUTES
Optional. A comma-delimited list of file attributes to be set on the file being moved. The following file attributes are supported:

- ReadOnly
- Temporary
- Archive
- Hidden
- System
- Normal

If ATTRIBUTES is not used, the file's attributes are maintained. If Normal is specified as well as any other attributes, Normal is overridden by whatever other attribute is specified.

Individual attributes must be specified explicitly. For example, if you specify just the ReadOnly attribute, all other existing attributes are overwritten.

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

```
<CFFILE ACTION="Move"
    SOURCE="c:\files\upload\keymemo.doc"
    DESTINATION="c:\files\memo\">
```

CFFILE ACTION="Rename"

Use CFFILE with the Rename action to rename a file that already exists on the server.

Syntax

```
<CFFILE ACTION="Rename"
    SOURCE="file_name"
    DESTINATION="file_name"
    ATTRIBUTES="file_attributes">
```

SOURCE
Required. The file to rename (with directory location).

DESTINATION
Required. The new name for the file (with directory location).

ATTRIBUTES
Optional. A comma-delimited list of file attributes to be set on the file being renamed. The following file attributes are supported:

- ReadOnly
- Temporary
- Archive
- Hidden
- System
- Normal

If ATTRIBUTES is not used, the file's attributes are maintained. If Normal is specified as well as any other attributes, Normal is overridden by whatever other attribute is specified.

Individual attributes must be specified explicitly. For example, if you specify just the ReadOnly attribute, all other existing attributes are overwritten.

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"

The CFFILE tag can be used to copy a file from one directory to another on the server.

Syntax

```
<CFFILE ACTION="Copy"
    SOURCE="file_name"
    DESTINATION="directory_name"
    ATTRIBUTES="file_attributes">
```

SOURCE
Required. The file to copy (with directory location).

DESTINATION
Required. The directory where the copy of the file will be saved. This must include the trailing backslash (slash on UNIX).

ATTRIBUTES
Optional. A comma-delimited list of file attributes to be set on the file being copied. The following file attributes are supported:

- ReadOnly

- Temporary
- Archive
- Hidden
- System
- Normal

If ATTRIBUTES is not used, the file's attributes are maintained. If Normal is specified as well as any other attributes, Normal is overridden by whatever other attribute is specified.

Individual attributes must be specified explicitly. For example, if you specify just the ReadOnly attribute, all other existing attributes are overwritten.

Example The following example saves a copy of the `keymemo.doc` file in the `c:\files\backup\` directory:

```
<CFFILE ACTION="Copy"
    SOURCE="c:\files\upload\keymemo.doc"
    DESTINATION="c:\files\backup\">
```

CFFILE ACTION="Delete"

The CFFILE tag can be used to delete a file on the server.

Syntax

```
<CFFILE ACTION="Delete"
    FILE="file_name">
```

FILE
Required. The file to delete (with directory location).

Example The following example permanently deletes the specified file:

```
<CFFILE ACTION="Delete"
    FILE="c:\files\upload\#Variables.DeleteFileName#">
```

CFFILE ACTION="Read"

You can use the CFFILE tag to read an existing text file. The file is read into a dynamic parameter you can use anywhere in the page like any other dynamic parameter. For example, you could read a text file and then insert its contents into a database. Or you could read a text file and then use one of the find and replace functions to modify its contents.

Syntax

```
<CFFILE ACTION="Read"
    FILE="file_name"
    VARIABLE="var_name">
```

FILE
Required. The text file to be read (with directory location).

VARIABLE
Required. The name of the variable that will contain the contents of the text file after it has been read.

Example The following example creates a variable named “Message” that will contain the contents of the file message.txt.

```
<CFFILE ACTION="Read"
    FILE="c:\web\message.txt"
    VARIABLE="Message">
```

The variable "Message" could then be used in the page. For example, you could display the contents of the `message.txt` file in the final Web page:

```
<CFOUTPUT>#File.Message#</CFOUTPUT>
```

ColdFusion supports a number of powerful functions for manipulating the contents of text files. You can also use variable created by a CFFILE Read operation in ArrayToList and ListToArray functions.

See String Functions and Array Functions for more information about working with strings and arrays.

CFFILE ACTION="Write"

You can use the CFFILE tag to write a text file based on dynamic content. For example, you could create static HTML files from this content or log actions in a text file.

Syntax

```
<CFFILE ACTION="Write"
    FILE="file_name"
    OUTPUT="content"
    MODE="permission"
    ATTRIBUTES="file_attributes">
```

FILE
Required. The name of the file to be created (with directory location).

OUTPUT
Required. The content of the file to be created.

MODE
Optional. Defines permissions for a file in Solaris. Ignored in Windows. Valid entries correspond to the octal values (not symbolic) of the Unix chmod command. Permissions are assigned for owner, group, and other, respectively. For example:

`MODE=644`
Assigns the owner read/write permissions and group/other read permission.

`MODE=666`
Assigns read/write permissions for owner, group, and other.

`MODE=777`
Assigns read, write, and execute permissions for all.

ATTRIBUTES
Optional. A comma-delimited list of file attributes to be set on the file being written. The following file attributes are supported:

- ReadOnly
- Temporary
- Archive
- Hidden
- System
- Normal

If ATTRIBUTES is not used, the file's attributes are maintained. If Normal is specified as well as any other attributes, Normal is overridden by whatever other attribute is specified.

Individual attributes must be specified explicitly. For example, if you specify just the ReadOnly attribute, all other existing attributes are overwritten.

Example

The following example creates a file with the information a user entered into 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 where:

```
UpdateTitle="FieldWork"
FullName="World B. Frueh"
Date="10/30/98"
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 text:

```
Created By: World B. Frueh
Date: 10/30/98
We had a wonderful time in Cambridgeport.
```

This following examples show the use of the MODE attribute for UNIX. The first, creates the file `/tmp/foo` with permissions defined as `rw-r--r--` (owner=read/write, group/other=read).

```
<CFFILE ACTION="Write"
    FILE="/tmp/foo"
    MODE=644>
```

This example appends to the specified file and makes permissions read/write (rw) for all.

```
<CFFILE ACTION="Append"
    DESTINATION="/home/tomj/testing.txt"
    MODE=666
    OUTPUT="Is this a test?">
```

The next example uploads a file and gives it `rwx-rw-rw` permissions (owner/group/other=read/write).

```
CFFILE ACTION="Upload"
    FILEFIELD="fieldname"
    DESTINATION="/tmp/program.exe"
    MODE=755>
```

CFFILE ACTION="Append"

Use CFFILE with the Append action to append additional text to the end of an existing text file, for example, when creating log files.

Syntax

```
<CFFILE ACTION="Append"
    FILE="file_name"
    OUTPUT="string"
    ATTRIBUTES="file_attributes">
```

FILE
Required. The file to which the content of the OUTPUT attribute is appended (with directory location).

OUTPUT
Required. The string to be appended to the file designated in the DESTINATION attribute.

ATTRIBUTES
Optional. A comma-delimited list of file attributes to be set on the file being appended. The following file attributes are supported:

- ReadOnly
- Temporary
- Archive
- Hidden
- System
- Normal

If ATTRIBUTES is not used, the file's attributes are maintained. If Normal is specified as well as any other attributes, Normal is overridden by whatever other attribute is specified.

Individual attributes must be specified explicitly. For example, if you specify just the ReadOnly attribute, all other existing attributes are overwritten.

Example The following example appends the text string "But Davis Square is the place to be." to the file `fieldwork.txt` which was created in the previous example:

```
<CFFILE ACTION="Append"
    FILE="c:\files\updates\fieldwork.txt"
    OUTPUT="<B>But Davis Square is the place to be.</B>">
```

CFFORM

CFFORM allows you to build a form with CFML custom control tags that provide much greater functionality than standard HTML form input elements.

Syntax

```
<CFFORM NAME="name"
    ACTION="form_action"
    ENABLECAB="Yes/No"
    ONSUBMIT="javascript"
    TARGET="window_name">
    ENCTYPE="type"
...
</CFFORM>
```

NAME
Optional. A name for the form you are creating.

ACTION
Required. The name of the ColdFusion page that will be executed when the form is submitted for processing.

ENABLECAB
Optional. Yes or No. Allows users to download the Microsoft cabinet (*.cab) file(s) containing the Java classes used for Java applet-based CFFORM controls. If Yes, on opening the page, users are asked if they want to download the CAB file.

ONSUBMIT
Optional. JavaScript function to execute after other input validation returns. Use this attribute to execute JavaScript for preprocessing data before the form is submitted. See *Developing Web Applications with ColdFusion* for information on using JavaScript for form validation.

TARGET
Optional. The name of the window or window frame where the form output will be sent.

ENCTYPE
Optional. The MIME type used to encode data sent via the POST method. The default value is application/x-www-form-urlencoded. It is recommended that you accept the default value. This attribute is included for compatibility with the HTML FORM tag.

Usage

The following custom control tags are available:

- CFINPUT — Creates a form input element (radio button, text box, or checkbox) and can validate form input.
- CFSELECT — Creates a drop down listbox.
- CFSLIDER — Creates a slider control.

- CFTEXTINPUT — Creates a text input box.
- CFTREE — Creates a tree control.
- CFGRID — Creates a grid control for displaying tabular data in a ColdFusion form.
- CFAPPLET — Embeds a registered Java applet in a ColdFusion form. Applets are registered in the ColdFusion Administrator.

The ENABLECAB attribute is supported only for MS Internet Explorer clients that have Authenticode 2.0 installed. Authenticode 2.0 can be downloaded from `http://www.microsoft.com/ie/security/authent2.htm`.

Note These CAB files are digitally signed using VeriSign digital IDs to ensure file security.

Incorporating HTML form tags

CFFORM allows you to incorporate standard HTML in two ways:

- You can add standard FORM tag attributes and their values to the CFFORM tag. These attributes and values are passed directly through ColdFusion to the browser in creating a form. For example, you can use FORM tag attributes like TARGET to enhance your CFFORM features.
- HTML tags that can ordinarily be placed within an HTML FORM tag can also be placed between <CFFORM> and </CFFORM> tags.

For example, you use a standard HTML INPUT tag to create a submit button in a CFFORM:

```
<CFFORM
...
    <INPUT TYPE="Submit" VALUE=" Update... ">
...
</CFFORM>
```

Example

```
<!--- This example shows the use of CFINPUT controls in
a CFFORM --->
<HTML>
<HEAD>
<TITLE>
CFFORM Example
</TITLE>
</HEAD>

<BODY>
<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" METHOD="POST" ENABLECAB="Yes">

<TABLE>
<TR>
    <TD>
    <H4>This example displays the radio button input type
    for CFINPUT.</H4>
    Yes <CFINPUT TYPE="Radio" NAME="TestVal1" VALUE="Yes" CHECKED="yes">
    No <CFINPUT TYPE="Radio" NAME="TestVal1" VALUE="No">
    </TD>
</TR>

<TR>
    <TD>
    <H4>This example displays the checkbox input type for CFINPUT.</H4>
    <CFINPUT TYPE="Checkbox" NAME="ChkTest2" VALUE="Yes">
    </TD>
</TR>

<TR>
    <TD>
    <H4>This example shows a 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 the
      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

CFFTP allows users to implement File Transfer Protocol operations. By default, CFFTP caches an open connection to an FTP server.

Note The CFFTP tag is for moving files between a ColdFusion server and an FTP server. CFFTP cannot move files between a ColdFusion server and a browser (client). Use CFFILE ACTION="UPLOAD" to transfer files from the client to a ColdFusion server; use CFCONTENT to transfer files from a ColdFusion server to the browser.

CFFTP topics:

Establishing a Connection with CFFTP

Use this form of the CFFTP tag to establish a connection with an FTP server.

If you use connection caching to an already active FTP connection, you don't need to respecify the connection attributes:

- USERNAME
- PASSWORD
- SERVER

Note Changes to a cached connection, such as changing RETRYCOUNT or TIMEOUT values, may require reestablishing the connection.

Syntax

```
<CFFTP ACTION="action"
    USERNAME="name"
    PASSWORD="password"
    SERVER="server"
    TIMEOUT="timeout in seconds"
    PORT="port"
    CONNECTION="name"
    AGENTNAME="name"
    PROXYSERVER="proxyserver"
    PROXYBYPASS="proxybypass"
    RETRYCOUNT="number"
    STOPONERROR="Yes/No">
```

ACTION
Required. Determines the FTP operation to perform. To create an FTP connection, use Open. To terminate an FTP connection, use Close (with no other attributes except an optional connection name).

USERNAME
Required for Open. User name to pass in the FTP operation.

PASSWORD
Required for Open. Password to log the user.

SERVER
Required for Open. The FTP server to connect to, as in `ftp.myserver.com`

TIMEOUT
Optional. Value in seconds for the timeout of all operations, including individual data request operations. Defaults to 30 seconds.

PORT
Optional. The remote port to connect to. Defaults to 21 for FTP.

CONNECTION
Optional. The name of the FTP connection. Used to cache the current FTP connection or to reuse connection information from a previous connection of the same name. All calls to CFFTP with the same connection name will reuse the same FTP connection information.

AGENTNAME
Optional. Application or entity conducting transfer.

PROXYSERVER
Optional. A string that contains the name of the proxy server (or servers) to use if proxy access was specified. If this parameter is NULL, the tag reads proxy information from the registry.

PROXYBYPASS
Optional. An optional list of host names or IP addresses, or both, that are known locally. Requests to these names are not routed through the proxy. The list can contain wildcards, such as "157.55.* *int*", meaning any IP address starting with 157.55, or any name containing the substring "int", will bypass the proxy. If this parameter specifies the "<local>" macro as the only entry, the tag bypasses any host name that does not contain a period. For example, "www.microsoft.com" is routed to the proxy, whereas "internet" is not. If this parameter is NULL, the tag reads the bypass list from the registry.

RETRYCOUNT
Optional. Number of retries until failure is reported. Default is one (1).

STOPONERROR
Optional. Yes or No. When Yes, halts all processing and displays an appropriate error. Default is Yes.

When No, three variables are populated:

- CFFTP.Succeeded – Yes or No.

- CFFTP.ErrorCode – Error number
- CFFTP.ErrorText – Message text explaining error type

Example

```
<!--- This view-only example shows the use of CFFTP --->
<HTML>
<HEAD>
<TITLE>CFFTP Example</TITLE>
</HEAD>
<BODY>

<H3>CFFTP Example</H3>
<P>CFFTP allows users to 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 view-only 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.net"
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="lib"
       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>
--->

</BODY>
</HTML>
```

File and Directory Operations with CFFTP

Use this form of the CFFTP tag to perform file and directory operations with CFFTP.

If you use connection caching to an already active FTP connection, you don't need to respecify the connection attributes:

- USERNAME
- PASSWORD
- SERVER

Syntax

```
<CFFTP
    ACTION="action"
    USERNAME="name"
    PASSWORD="password"
    NAME="query_name"
    SERVER="server"
    ASCIIEXTENSIONLIST="extensions"
    TRANSFERMODE="mode"
    FAILIFEXISTS="Yes/No"
    DIRECTORY="directory name"
    LOCALFILE="filename"
    REMOTEFILE="filename"
    ATTRIBUTES="file attributes"
    ITEM="directory or file"
    EXISTING="file or directory name"
    NEW="file or directory name"
    PROXYSERVER="proxyserver"
    PROXYBYPASS="proxybypass">
```

ACTION

Required if connection is not already cached. If connection caching is used, the ACTION attribute is not required. Determines the FTP operation to perform. Can be one of the following:

- ChangeDir
- CreateDir
- ListDir
- GetFile
- PutFile
- Rename
- Remove
- GetCurrentDir
- GetCurrentURL
- ExistsDir
- ExistsFile

- Exists

Note: Names of objects (files and directories) are case-sensitive. Thus a ListDir on `test.log` will not find `test.LOG`.

USERNAME
Required if the FTP connection is not already cached. If connection caching is used, the USERNAME attribute is not required. User name to pass in the FTP operation.

PASSWORD
Required if the FTP connection is not already cached. If connection caching is used, the PASSWORD attribute is not required. Password to log the user.

NAME
Required for ACTION="ListDir". Specifies the query name to hold the directory listing.

SERVER
Required if the FTP connection is not already cached. If connection caching is used, the SERVER attribute is not required. The FTP server to connect to.

TIMEOUT
Optional. Value in seconds for the timeout of all operations, including individual data request operations. Defaults to 30 seconds.

PORT
Optional. The remote port to connect to. Defaults to 21 for FTP.

CONNECTION
Optional. The name of the FTP connection. Used to cache the current FTP connection or to reuse connection information from a previous connection of the same name. All calls to CFFTP with the same connection name will reuse the same FTP connection information.

ASCIIEXTENSIONLIST
Optional. A semicolon delimited list of file extensions that force ASCII transfer mode when TRANSFERMODE="Autodetect". Default extension list is:

```
txt;htm;html;cfm;cfml;shtm;shtml;css;asp;asa
```

TRANSFERMODE
Optional. The FTP transfer mode you want to use. Valid entries are ASCII, Binary, or Autodetect. Defaults to Autodetect.

AGENTNAME
Optional. Application or entity conducting transfer.

FAILIFEXISTS

Optional. Yes or No. Defaults to Yes. Specifies whether a GetFile operation will fail if a local file of the same name already exists.

DIRECTORY
Required for ACTION=ChangeDir, CreateDir, ListDir, and ExistsDir. Specifies the directory on which to perform an operation.

LOCALFILE
Required for ACTION=GetFile, and PutFile. Specifies the name of the file on the local file system.

REMOTEFILE
Required for ACTION=GetFile, PutFile, and ExistsFile. Specifies the name of the file on the FTP server's file system.

ATTRIBUTES
Optional. Defaults to "Normal." A comma delimited list of file attributes. Specifies the file attributes for the local file in a GetFile. Can be any combination of the following:

- ReadOnly
- Hidden
- System
- Archive
- Directory
- Compressed
- Temporary
- Normal

File attributes differ according to environment.

ITEM
Required for ACTION=Exists, and Remove. Specifies the object, file or directory, of these actions.

EXISTING
Required for ACTION=Rename. Specifies the current name of the file or directory on the remote server.

NEW
Required for ACTION=Rename. Specifies the new name of the file or directory on the remote server.

RETRYCOUNT
Optional. Number of retries until failure is reported. Default is one (1).

STOPONERROR

Optional. Yes or No. When Yes, halts all processing and displays an appropriate error. Default is No.

When No, three variables are populated:

- CFFTP.Succeeded – Yes or No.
- CFFTP.ErrorCode – Error number (See STOPONERROR variables, below)
- CFFTP.ErrorText – Message text explaining error condition

PROXYSERVER
Optional. A string that contains the name of the proxy server (or servers) to use if proxy access was specified. If this parameter is NULL, the tag reads proxy information from the registry.

PROXYBYPASS
Optional. An optional list of host names or IP addresses, or both, that are known locally. Requests to these names are not routed through the proxy. The list can contain wildcards, such as "157.55.* *int*", meaning any IP address starting with 157.55, or any name containing the substring "int", will bypass the proxy. If this parameter specifies the "<local>" macro as the only entry, the tag bypasses any host name that does not contain a period. For example, "www.microsoft.com" is routed to the proxy, whereas "internet" is not. If this parameter is NULL, the tag reads the bypass list from the registry.

Connection caching

Once you've established a connection with CFFTP, you can reuse the connection 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 additional FTP operations that use the same CONNECTION name automatically make use of the information stored in the connection object. This facility helps save connection time and drastically improves file transfer operation performance.

Note Changes to a cached connection, such as changing RETRYCOUNT or TIMEOUT values, may require reestablishing the connection.

Example The following example opens an FTP connection, retrieves a file listing, showing file or directory name, path, URL, length, and modification date. Connection caching is used to maintain the link to the server, and automatic error checking is enabled.

```
<CFFTP CONNECTION=FTP
    USERNAME="betauser"
    PASSWORD="monroe"
    SERVER="beta.company.com"
    ACTION="Open"
    STOPONERROR="Yes">
```

```
<CFFTP CONNECTION=FTP
    ACTION="GetCurrentDir"
    STOPONERROR="Yes">

<CFOUTPUT>
    FTP directory listing of #cfftp.returnvalue#.<p>
</CFOUTPUT>

    <CFOUTPUT>Return is #cfftp.returnvalue#</CFOUTPUT><BR>

<CFFTP CONNECTION="FTP"
    ACTION="listdir"
    DIRECTORY="/*."
    NAME="q"
    STOPONERROR="Yes">
<HR>FTP Directory Listing:<P>
<CFTABLE QUERY="q" HTMLTABLE>
    <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>
```

CFGRID

Used inside CFFORM, CFGRID allows you to place a grid control in a ColdFusion form. A grid control is a table of data divided into rows and columns. CFGRID column data is specified with individual CFGRIDCOLUMN tags.

See also CFGRIDROW and CFGRIDUPDATE tags.

Syntax

```
<CFGRID NAME="name"
    HEIGHT="integer"
    WIDTH="integer"
    VSPACE="integer"
    HSPACE="integer"
    ALIGN="value"
    QUERY="query_name"
    INSERT="Yes/No"
    DELETE="Yes/No"
    SORT="Yes/No"
    FONT="column_font"
    FONTSIZE="size"
    ITALIC="Yes/No"
    BOLD="Yes/No"
    HREF="URL"
    HREFKEY="column_name"
    TARGET="URL_target"
    APPENDKEY="Yes/No"
    HIGHLIGHTHREF="Yes/No"
    ONVALIDATE="javascript_function"
    ONERROR="text"
    GRIDDATAALIGN="position"
    GRIDLINES="Yes/No"
    ROWHEIGHT="pixels"
    ROWHEADERS="Yes/No"
    ROWHEADERALIGN="position"
    ROWHEADERFONT="font_name"
    ROWHEADERFONTSIZE="size"
    ROWHEADERITALIC="Yes/No"
    ROWHEADERBOLD="Yes/No"
    ROWHEADERWIDTH="col_width"
    COLHEADERS="Yes/No"
    COLHEADERALIGN="position"
    COLHEADERFONT="font_name"
    COLHEADERFONTSIZE="size"
    COLHEADERITALIC="Yes/No"
    COLHEADERBOLD="Yes/No"
    BGCOLOR="color"
    SELECTCOLOR="color"
    SELECTMODE="mode"
    MAXROWS="number"
    NOTSUPPORTED="text"
    PICTUREBAR="Yes/No"
    INSERTBUTTON="text"
```

```
    DELETEBUTTON="text"
    SORTASCENDINGBUTTON="text"
    SORTDESCENDINGBUTTON="text">

</CFGRID>
```

NAME
Required. A name for the grid element.

HEIGHT
Optional. Height value of the grid control in pixels.

WIDTH
Optional. Width value of the grid control in pixels.

VSPACE
Optional. Vertical margin spacing above and below the grid control in pixels.

HSPACE
Optional. Horizontal margin spacing to the left and right of the grid control in pixels.

ALIGN
Optional. Alignment value. Valid entries are: Top, Left, Bottom, Baseline, Texttop, Absbottom, Middle, Absmiddle, Right.

QUERY
Optional. The name of the query associated with the grid control.

INSERT
Optional. Yes or No. Yes allows end users to insert new row data into the grid. Default is No.

DELETE
Optional. Yes or No. Yes allows end users to delete row data in the grid. Default is No.

SORT
Optional. Yes or No. When Yes sort buttons are added to the grid control. When clicked the sort buttons perform a simple text sort on the selected column. Default is No.

FONT
Optional. Font name to use for all column data in the grid control.

FONTSIZE
Optional. Font size for text in the grid control, measured in points.

ITALIC

Optional. Yes or No. Yes presents all grid control text in italic. Default is No.

BOLD
Optional. Yes or No. Yes presents all grid control text in boldface. Default is No.

HREF
Optional. URL to associate with the grid item or a query column for a grid that is populated from a query. If HREF is a query column, then the HREF value that is displayed is populated by the query. If HREF is not recognized as a query column, it is assumed that the HREF text is an actual HTML HREF.

HREFKEY
Optional. The name of a valid query column when the grid uses a query. The column specified becomes the Key no matter what the select mode is for the grid.

TARGET
Optional. Target attribute for HREF URL.

APPENDKEY
Optional. Yes or No. When used with HREF, Yes passes the CFGRIDKEY variable along with the value of the selected tree item in the URL to the application page specified in the CFFORM ACTION attribute. Default is Yes.

HIGHLIGHTHREF
Optional. Yes highlights links associated with a CFGRID with an HREF attribute value. No disables highlight. Default is Yes.

ONVALIDATE
Optional. The name of a valid JavaScript function used 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 and False otherwise.

ONERROR
Optional. The name of a valid JavaScript function you want to execute in the event of a failed validation.

GRIDDATAALIGN
Optional. Enter Left, Right, or Center to position data in the grid within a column. Default is Left.

GRIDLINES
Optional. Yes or No. Yes enables rules (lines) in the grid control, No suppresses row and column rules. Default is Yes.

ROWHEIGHT
Optional. Enter a numeric value for the number of pixels to determine the minimum row height for the grid control. Used with CFGRIDCOLUMN TYPE="Image", you can use ROWHEIGHT to define enough room for graphics you want to display in the row.

ROWHEADER
Optional. Yes or No. Yes displays row labels in the grid control. Defaults to Yes.

ROWHEADERALIGN
Optional. Enter Left, Right, or Center to position data within a row header. Default is Left.

ROWHEADERFONT
Optional. Font to use for the row label.

ROWHEADERFONTSIZE
Optional. Size font for row label text in the grid control, measured in points.

ROWHEADERITALIC
Optional. Yes or No. Yes presents row label text in italic. Default is No.

ROWHEADERBOLD
Optional. Yes or No. Yes presents row label text in boldface. Default is No.

ROWHEADERWIDTH
Optional. The width, in pixels, of the row header column.

COLHEADERS
Optional. Yes or No. Yes displays column headers in the grid control. Defaults to Yes.

COLHEADERALIGN
Optional. Enter Left, Right, or Center to position data within a column header. Default is Left.

COLHEADERFONT
Optional. Font to use for the column header in the grid control.

COLHEADERFONTSIZE
Optional. Size font for column header text in the grid control, measured in points.

COLHEADERITALIC
Optional. Yes or No. Yes presents column header text in italic. Default is No.

COLHEADERBOLD
Optional. Yes or No. Yes presents column header text in boldface. Default is No.

BGCOLOR
Optional. Background color value for the grid control. Valid entries are: black, magenta, cyan, orange, darkgray, pink, gray, white, lightgray, yellow.

A hex value can be entered in the form:

BGCOLOR="##*xxxxxx*"
where *x* is 0-9 or A-F. Use either two pound signs or no pound signs.

SELECTCOLOR
Optional. Background color for a selected item. See BGCOLOR for color options.

SELECTMODE
Optional. Selection mode for items in the grid control. Valid entries are:

- Edit — Users can edit grid data.
- Single — User selections are confined to the selected cell.
- Row — User selections automatically extend to the row containing selected cell.
- Column — User selections automatically extend to column containing selected cell.
- Browse — User can only browse grid data.

Default is Browse.

MAXROWS
Optional. Specifies the maximum number of rows you want to display in the grid.

NOTSUPPORTED
Optional. The text you want to display if the page containing 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>"
```

By default, if no message is specified, the following message appears:

```
<B>Browser must support Java to <BR>
view ColdFusion Java Applets!</B>
```

PICTUREBAR
Optional. Yes or No. When Yes, image buttons are used for the Insert, Delete, and Sort actions rather than text buttons. Default is No.

INSERTBUTTON
Optional. Text to use for the Insert action button. The default is Insert.

DELETEBUTTON
Optional. Text to use for the Delete action button. The default is Delete.

SORTASCENDINGBUTTON
Optional. The text to use for the Sort button. The default is "A -> Z".

SORTDESCENDINGBUTTON
Optional. The text to use for the Sort button. The default is "Z <- A".

Usage

You can populate a CFGRID with data from a CFQUERY. If you do not specify any CFGRIDCOLUMN entries, a default set of columns is generated. Each column in the query is included in the default column list. In addition, a default header for each

column is created by replacing any hyphen (-) or underscore (_) characters in the table column name with spaces. The first character and any character after a space is changed to uppercase; all other characters are lowercase.

Select mode and form variables

Grid data is submitted in a CFFORM as form variables, depending on the value of the SELECTMODE attribute as follows:

- When SELECTMODE="Single", grid data is returned as *grid_name.selectedname* and the value of the selected cell.
- When SELECTMODE="Column", grid data is returned as a comma-separated list of all the values for the selected column.
- When SELECTMODE="Row", grid data is returned as *grid_name.colum1_name* and *grid_name.column2_name* and their respective values for the selected row.
- When SELECTMODE="Browse", no selection data is returned.

Using SELECTMODE="Edit"

When SELECTMODE="Edit", one-dimensional arrays are used to store data about changes to the grid cells. For example, a one-dimensional array is used to store the type of edits made to grid cells:

```
gridname.RowStatus.Action [ action ]
```

Where *gridname* is the name of the CFGRID and *action* is U, I, or D for Update, Insert, and Delete, respectively.

ColdFusion also maintains both the value of the edited cell and the original value in one-dimensional arrays. You can reference this data in ColdFusion expressions as follows:

```
gridname.colname[ value ]
gridname.original.colname[ value ]
```

Where *gridname* is the name of the CFGRID, *colname* is the name of the column, and value is the index position containing the grid data.

Using the HREF attribute

When specifying a URL with grid items using the HREF attribute, the value of the SELECTMODE attribute determines whether the appended key value is limited to a single grid item or whether it extends to a grid column or row. When a user clicks on a linked grid item, a CFGRIDKEY variable is appended to the URL in the following form:

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

If the APPENDKEY attribute is set to No, then no grid values are appended to the URL.

The value of *selection* is determined by the value of the SELECTMODE attribute:

- When SELECTMODE="Single", *selection* is the value of the column clicked.
- When SELECTMODE="Row", *selection* is a comma-separated list of column values in the clicked row, beginning with the value of the first cell in the selected row.

When SELECTMODE="Column", selection is a comma-separated list of row values in the clicked column, beginning with the value of the first cell in the selected column.

Note CFGRID incorporates a Java applet, so browsers must be Java-enabled for CFGRID to work properly.

Example

```
<!--- This example shows the CFGRID, CFGRIDCOLUMN, CFGRIDROW,
and CFGRIDUPDATE tags in action --->

<!--- use a query to show the useful qualities of CFGRID --->

<!--- If the gridEntered form field has been tripped,
perform the gridupdate on the table specified in the database.
Using the default value keyonly=yes allows us to 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>
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" METHOD="POST" ENABLECAB="Yes">

<!--- We include Course_ID in the CFGRID, but do not allow
for its 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>

...
```

CFGRIDCOLUMN

Used with CFGRID in a CFFORM, you use CFGRIDCOLUMN to specify individual column data in a CFGRID control. Font and alignment attributes used in CFGRIDCOLUMN override any global font or alignment settings defined in CFGRID.

Syntax

```
<CFGRIDCOLUMN NAME="column_name"
    HEADER="header"
    WIDTH="column_width"
    FONT="column_font"
    FONTSIZE="size"
    ITALIC="Yes/No"
    BOLD="Yes/No"
    HREF="URL"
    HREFKEY="column_name"
    TARGET="URL_target"
    SELECT="Yes/No"
    DISPLAY="Yes/No"
    TYPE="type"
    HEADERFONT"font_name"
    HEADERFONTSIZE="size"
    HEADERITALIC="Yes/No"
    HEADERBOLD="Yes/No"
    DATAALIGN="position"
    HEADERALIGN="position"
    NUMBERFORMAT="format">
```

NAME
Required. A name for the grid column element. If the grid uses a query, the column name must specify the name of a query column.

HEADER
Optional. Text for the column header. The value of HEADER is used only when the CFGRID COLHEADERS attribute is Yes (or omitted, since it defaults to Yes).

WIDTH
Optional. The width of the column in pixels. By default the column is sized based on the longest column value.

FONT
Optional. Font name to use for data in the column. Defaults to browser-specified font.

FONTSIZE
Optional. Font size for text in the column. Defaults to browser-specified font size.

ITALIC
Optional. Yes or No. Yes presents text in the column in italic. Default is No.

BOLD
Optional. Yes or No. Yes presents text in the column in boldface. Default is No.

HREF
Optional. URL to associate with the grid item. You can specify a URL that is relative to the current page:

```
../mypage.cfm
```

Or an absolute URL:

```
http://myserver.com/mydir/mypage.cfm.
```

HREFKEY
Optional. The name of a valid query column when the grid uses a query. The column specified becomes the Key no matter what the select mode is for the grid.

TARGET
Optional. The name of the frame in which to open the link specified in HREF.

SELECT
Optional. Yes or No. Yes allows end users to select a column in a grid control. When No, the column cannot be edited, even if the CFGRID INSERT or DELETE attributes are enabled. The value of the SELECT attribute is ignored if the CFGRID SELECTMODE attribute is set to Row or Browse.

DISPLAY
Optional. Yes or No. Use to hide columns. Default is Yes to display the column.

TYPE
Optional. Enter Image or Numeric. When TYPE="Image", the grid attempts to display an image corresponding to the value in the column, which can be a built in ColdFusion image name, or an image of your choice in the `cfide\classes` directory or a subdirectory, referenced with a relative URL. Built-in image names are as follows:

- cd
- computer
- document
- element
- folder
- floppy
- fixed
- remote

If an image is larger than the column cell where it is being placed, the images is clipped to fit the cell.

When TYPE="Numeric", data in the grid can be sorted by the end user as numeric data rather than as simple character text.

HEADERFONT
Optional. Font to use for the column header. Defaults to browser-specified font.

HEADERFONTSIZE
Optional. Font size to use for the column header in pixels. Defaults to browser-specified font size.

HEADERITALIC
Optional. Yes or No. Yes presents column header text in italic. Default is No.

HEADERBOLD
Optional. Yes or No. Yes presents header text in boldface. Default is No.

DATAALIGN
Optional. Alignment for column data. Valid entries are: Left, Center, or Right. Default is Left.

HEADERALIGN
Optional. Alignment for the column header text. Valid entries are: Left, Center, or Right. Default is Left.

NUMBERFORMAT
Optional. The format for displaying numeric data in the grid.

NUMBERFORMAT mask characters

Mask characters you can use in the NUMBERFORMAT attribute correspond with those used in the NumberFormat CFML function. For more information about the NumberFormat function, see Chapter 3, "ColdFusion Functions," on page 201.

NumberFormat Mask Characters	
Character	**Meaning**
_ (underscore)	Optional digit placeholder.
9	Optional digit placeholder. Same as _, but shows decimal places more clearly.
.	Specifies the location of a mandatory decimal point.
0	Located to the left or right of a mandatory decimal point, to force padding with zeros.
( )	Places parentheses around the mask if the number is less than 0.

NumberFormat Mask Characters (Continued)	
Character	**Meaning**
+	Places + in front of positive numbers, - (minus sign) in front of negative numbers.
-	Place " " (space) in front of positive, - (minus sign) in front of negative numbers.
,	Separates thousands with commas.
L,C	Specifies left-justify or center-justify a number within the width of the mask column. L or C must appear as the first character of the mask. By default, numbers are right-justified.
$	Places a dollar sign in front of the formatted number. $ must appear as the first character of the mask.
^	Separates left from right formatting.

Example

```
<!--- This example shows the CFGRIDCOLUMN tag in action --->
...
<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">
...
```

CFGRIDROW

CFGRIDROW allows you to 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.

Syntax

```
<CFGRIDROW DATA="col1, col2, ...">
```

DATA
Required. A comma-separated list of column values. If a column value contains a comma character, it must be escaped with a second comma character.

Example

```
...

<!--- use a CFLOOP to loop through the query and define CFGRIDROW
data each time through the loop --->
    <CFLOOP query="GetCourses">
        <CFGRIDROW
          DATA="#Course_ID#,#Dept_ID#,#CorNumber#,#CorName#,
           #CorLevel#,#CorDesc#">
    </CFLOOP>
</CFGRID>
</CFFORM>

</BODY>
</HTML>
```

CFGRIDUPDATE

Used in a CFGRID, CFGRIDUPDATE allows you to perform updates to data sources directly from edited grid data. CFGRIDUPDATE provides a direct interface with your data source.

CFGRIDUPDATE first applies DELETE row actions followed by INSERT row actions and finally UPDATE row actions. Row processing stops if any errors are encountered.

Syntax

```
<CFGRIDUPDATE GRID="gridname"
    DATASOURCE="data source name"
    DBTYPE="type"
    DBSERVER="dbms"
    DBNAME="database name"
    TABLENAME="table name"
    USERNAME="data source username"
    PASSWORD="data source password"
    TABLEOWNER="table owner"
    TABLEQUALIFIER="qualifier"
    PROVIDER="COMProvider"
    PROVIDERDSN="datasource"
    KEYONLY="Yes/No">
```

GRID
Required. The name of the CFGRID form element that is the source for the update action.

DATASOURCE
Required. The name of the data source for the update action.

DBTYPE
Optional. The database driver type:

- ODBC (default) — ODBC driver.
- Oracle73 — Oracle 7.3 native database driver. Using this option, the ColdFusion Server computer must have Oracle 7.3.3 (or greater) client software installed.
- Oracle80 —Oracle 8.0 native database driver. Using this option, the ColdFusion Server computer must have Oracle 8.0 (or greater) client software installed.
- Sybase11 —Sybase System 11 native database driver. Using this option, the ColdFusion Server computer must have Sybase 11.1.1 (or greater) client software installed. Sybase patch ebf 7729 is recommended.

DBSERVER
Optional. For native database drivers, specifies the name of the database server machine. If specified, DBSERVER overrides the server specified in the data source.

DBNAME

Optional. The database name (Sybase System 11 driver only). If specified, DBNAME overrides the default database specified in the data source.

TABLENAME
Required. The name of the table you want to update. Note the following:

- ORACLE drivers — This specification must be in uppercase.
- Sybase driver — This specification is case-sensitive and must be in the same case as that used when the table was created

USERNAME
Optional. If specified, USERNAME overrides the username value specified in the ODBC setup.

PASSWORD
Optional. If specified, PASSWORD overrides the password value specified in the ODBC setup.

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 the qualifier for the table. The purpose of table qualifiers varies across drivers. For SQL Server and Oracle, the qualifier refers to the name of the database that contains the table. For the Intersolv dBase driver, the qualifier refers to the directory where the DBF files are located.

PROVIDER
Optional. COM provider (OLE-DB only).

PROVIDERDSN
Optional. Data source name for the COM provider (OLE-DB only).

KEYONLY
Optional. Yes or No. Yes specifies that in the update action, the WHERE criteria is confined to just the key values. No specifies that in addition to the key values, the original values of any changed fields are included in the WHERE criteria. Default is Yes.

Example

```
<!--- This example shows the CFGRID, CFGRIDCOLUMN, CFGRIDROW,
and CFGRIDUPDATE tags in action --->
...
<!--- If the gridEntered form field has been tripped,
perform the gridupdate on the table specified in the database.
Using the default value keyonly=yes allows us to 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

CFHEADER generates custom HTTP response headers to return to the client.

Syntax

```
<CFHEADER NAME="header_name"
    VALUE="header_value">
```

NAME
Required. A name for the header.

VALUE
Required. A value for the HTTP header.

Example

```
<!--- This example shows the use of CFHEADER --->
<HTML>
<HEAD>
<TITLE>CFHEADER Example</TITLE>
</HEAD>

<BODY>
<H3>CFHEADER Example</H3>

<P>CFHEADER generates custom HTTP response headers
to return to the client.
<P>The following example forces the browser client
to purge its cache of a requested file.
<CFHEADER Name="Expires" Value="#Now()#">

</BODY>
</HTML>
```

CFHTMLHEAD

CFHTMLHEAD writes the text specified in the TEXT attribute to the <HEAD> 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 page header.

Syntax

```
<CFHTMLHEAD TEXT="text">
```

TEXT
The text you want to add to the <HEAD> area of an HTML page. Everything inside the quotation marks is placed in the <HEAD> section.

Example

```
<!---  This example shows the use of CFHTMLHEAD --->
<CFHTMLHEAD TEXT="<TITLE>This is an example of a generated header</TITLE>
<BASE HREF='http://www.allaire.com/'>
">

<HTML>
<HEAD>
</HEAD>

<BODY>
<H3>CFHTMLHEAD Example</H3>

<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>
</HTML>
```

CFHTTP

The CFHTTP tag allows you to execute POST and GET operations on files. Using CFHTTP, you can execute standard GET operations as well as create a query object from a text file. POST operations allow you to upload MIME file types to a server, or post cookie, formfield, URL, file, or CGI variables directly to a specified server.

Syntax

```
<CFHTTP URL="hostname"
    PORT="port_number"
    METHOD="get_or_post"
    USERNAME="username"
    PASSWORD="password"
    NAME="queryname"
    COLUMNS="query_columns"
    PATH="path"
    FILE="filename"
    DELIMITER="character"
    TEXTQUALIFIER="character"
    RESOLVEURL="Yes/No"
    PROXYSERVER="hostname">
</CFHTTP>
```

Note Terminate CFHTTP POST operations with </CFHTTP>. Termination is not required with CFHTTP GET operations.

URL
Required. Full URL of the host name or IP address of the server on which the file resides.

PORT
Optional. The port number on the server from which the object is being requested. Default is 80. When used with RESOLVEURL, the URLs of retrieved documents that specify a port number are automatically resolved to preserve links in the retrieved document.

METHOD
Required. GET or POST. Use GET to download a text or binary file, or to create a query from the contents of a text file. Use POST to send information to a server page or a CGI program for processing. POST requires the use of a CFHTTPPARAM tag.

USERNAME
Optional. When required by a server, a valid username.

PASSWORD
Optional. When required by a server, a valid password.

NAME

Optional. The name to assign to a query when a query is to be constructed from a file.

COLUMNS
Optional. The column names for a query. If no column names are specified, defaults to the columns listed in the first row of a text file.

PATH
Optional. The path to the directory in which a file is to be stored. If a path is not specified in a POST or GET operation, a variable is created (CFHTTP.FileContent) that you can use to present the results of the POST operation in a CFOUTPUT.

FILE
Required in a POST operation if PATH is specified. The filename to be used for the file that is accessed. For GET operations, defaults to the name specified in URL. Enter path information in the PATH attribute.

DELIMITER
Required for creating a query. Valid characters are a tab or comma. Default is a comma (,).

TEXTQUALIFIER
Required for creating a query. Indicates the start and finish of a column. Should be appropriately escaped when embedded in a column. For example, if the qualifier is a quotation mark, it should be escaped as """". If there is no text qualifier in the file, specify a blank space as " ". Default is the quote mark (").

RESOLVEURL
Optional. Yes or No. Default is No. For GET and POST operations, when Yes, any page reference returned into the FileContent internal variable will have its internal URLs fully resolved, including port number, so that links remain intact. The following HTML tags, which can contain links, will be resolved:

- 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.

Usage

Note the following:

- **HTTP GET** — A user can specify a URL that points to a text or binary file. The file will be downloaded and its contents stored in a CF variable or in a file so that the user can manipulate the data. The internal variable FileContent is available for text and MIME file types. The MimeType variable is available for all file manipulations. These variables can be accessed in the following manner:

  ```
  #CFHTTP.FileContent#
  ```

  ```
  #CFHTTP.MimeType#
  ```

- **GET file into a query** — To download a file in a ColdFusion page so that a query can be built using the file, the file must be either comma-separated or tab-delimited. Although risky, text qualification may be omitted. The file will be parsed and an appropriate query built from it. Columns may be specified in the attribute list so that the client can override the columns specified in the file. There is error checking within the tag that prevents a user from either entering an invalid column name or using an invalid column name that was specified in the original file. If such an illegal filename is encountered, the illegal characters are stripped. Such 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.
- **HTTP POST** — CFHTTPPARAM tags can be nested inside a CFHTTP tag in a POST operation. The browser can be pointed to a URL specifying a CGI executable or a ColdFusion page. Since multiple CFHTTPPARAM tags can be nested in one CFHTTP tag, you can construct a multipart/form-data style post. A file content variable is created and this can be used in a CFOUTPUT. If PATH and FILE are specified, the data returned from the server is saved to the specified location.
- **Authentication** — CFHTTP supports Windows NT Basic Authentication for both GET and POST operations. However, Basic Authentication will not work if your Web server has enabled Windows NT Challenge/Response (Microsoft IIS).
- **Encryption** — To verify the encryption version used with SSL, access the schannel.dll file in the windows\system directory and display its properties. The Version tab lists a Description item, which displays one of the following:
  - US and Canada — 128 bit encryption
  - Export — 40 bit

  SSL is not supported by ColdFusion running on UNIX.

Examples

```
<!--- This example shows the use of CFHTTP to pull information
dynamically from the snippets directory --->
<HTML>
<HEAD>
<TITLE>
CFHTTP Example
</TITLE>
</HEAD>

<BODY>
<H3>CFHTTP Example</H3>

<P>This example shows the ability of CFHTTP to pull
the contents of a web resource from the Internet
or from a local directory.

<FORM ACTION="cfhttp.cfm" METHOD="POST">
Try entering a URL for the tag to return:
<INPUT TYPE="Text" size=25 NAME="urladdress"
  VALUE="http://www.allaire.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>Your file was of type: #CFHTTP.MimeType#
<P>#HTMLCodeFormat(CFHTTP.FileContent)#
</CFOUTPUT>

</BODY>
</HTML>
```

CFHTTPPARAM

Required for CFHTTP POST operations, CFHTTPPARAM is used to specify the parameters necessary to build a CFHTTP POST.

Syntax

```
<CFHTTPPARAM NAME="name"
    TYPE="type"
    VALUE="transaction type"
    FILE="filename">
```

NAME
Required. A variable name for the data being passed.

TYPE
Required. The transaction type. Valid entries are:

- URL
- FormField
- Cookie
- CGI
- File

VALUE
Optional for TYPE="File". Specifies the value of the URL, FormField, Cookie, File, or CGI variable being passed.

FILE
Required for TYPE="File".

Example

```
<!--- This example shows the use of CFHTTPPARAM --->
<HTML>
<HEAD>
<TITLE>CFHTTPPARAM Example</TITLE>
</HEAD>

<BODY bgcolor=silver>
<H3>CFHTTPPARAM Example</H3>

<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 simply 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>

<H3>For CFHTTPPARAM code, see right frame</H3>
<!--- <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="Coookie" VALUE="This is a cookie">
</CFHTTP>

<CFOUTPUT>
        #CFHTTP.FileContent#
</CFOUTPUT> --->

</BODY>
</HTML>
```

CFIF/CFELSE/CFELSEIF

Used with CFELSE and CFELSEIF, CFIF lets you create simple and compound conditional statements in CFML. The value in the CFIF tag can be any expression.

Syntax

```
<CFIF expression>
    HTML and CFML tags
<CFELSE>
    HTML and CFML tags
<CFELSEIF expression>
    HTML and CFML tags
</CFIF>
```

Usage

Note that when testing for the return value of any function that returns a Boolean, you do not need to explicitly define the TRUE condition. The following code uses IsArray as an example:

```
<CFIF IsArray(myarray)>
```

When successful, `IsArray` evaluates to YES, the string equivalent of the Boolean TRUE. This method is preferred over explicitly defining the TRUE condition:

```
<CFIF IsArray(myarray) IS TRUE>
```

Example

```
<!--- This example shows the interaction of CFIF, CFELSE,
and CFELSEIF --->
...
<H3>CFIF Example</H3>

<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>
the city = San Diego.
<hr>
<!--- use CFIF to test a condition when outputting a query --->
<P>The following are centers in San Diego:

<CFOUTPUT QUERY="getCenters" >
<CFIF city is "San Diego">
    <BR><B>Name/Address:</B>#Name#, #Address1#, #City#, #State#
    <BR><B>Contact:</B> #Contact#<BR>
</CFIF>
</CFOUTPUT>

<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 --->
<hr>
<P>Complete information is shown for centers in San Diego
or Santa Ana.  All other centers are listed in italics:

<CFOUTPUT QUERY="getCenters">
<CFIF city is "San Diego" OR city is "Santa Ana">
    <H4>Featured Center in <CFIF 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>

<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.
<hr>
<P>
<!--- use CFIF in conjunction with CFELSEIF to specify
more than one branch in a conditional situation --->
<CFOUTPUT QUERY="getCenters">
<CFIF city is "San Diego" OR city is "Santa Ana">
    <BR><I>#Name#, #City#</I> (this one is in <CFIF city is "San
       Diego">San Diego<CFELSE>Santa Ana</CFIF>)
<CFELSEIF city is "San Francisco">
    <BR><I>#Name#, #City#</I> (this one is in San Francisco)
<CFELSEIF 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>

</BODY>
</HTML>
```

CFINCLUDE

CFINCLUDE lets you embed references to ColdFusion pages in your CFML. If necessary, you can embed CFINCLUDE tags recursively.

For an additional method of encapsulating CFML, see the CFMODULE tag, used to create custom tags in CFML.

Syntax

```
<CFINCLUDE TEMPLATE="template_name">
```

TEMPLATE
A logical path to an existing page.

Usage

When ColdFusion searches for included files as follows:

- Checks the directory in which the current page lives.
- Searches directories explicitly mapped in the ColdFusion Administrator for the included file.

Example

```
<!--- This example shows the use of CFINCLUDE to paste
pieces of CFML or HTML code into another page dynamically --->
<HTML>
<HEAD>
    <TITLE>CFINCLUDE Example</TITLE>
</HEAD>

<BODY>
<H3>CFINCLUDE Example</H3>

<H4>This example includes the main.htm page from the CFDOCS
directory.  The images do not show up correctly 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">

</BODY>
</HTML>
```

CFINDEX

Use the CFINDEX tag to populate collections with indexed data. CFINDEX and CFSEARCH encapsulate the Verity indexing and searching utilities. Verity collections can be populated from either text files in a directory you specify, or from a query generated by any ColdFusion query. Before you can populate a Verity collection, you need to create the collection using either the CFCOLLECTION tag or the ColdFusion Administrator. Use CFSEARCH to search collections you populate with CFINDEX.

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/No"
    EXTERNAL="Yes/No"
    LANGUAGE="language">
```

COLLECTION

Required. Specifies a collection name. If you are indexing an external collection (EXTERNAL is "Yes"), specify the collection name, including fully qualified path:

```
COLLECTION="e:\collections\personnel"
```

You cannot combine internal and external collections in the same indexing operation.

ACTION

Optional. Specifies the index action. Valid entries are:

- Update — Updates the index and adds the key specified in KEY to the index if it is not already defined.
- Delete — Deletes the key specified in KEY in the specified collection.
- Purge — Deletes data in the specified collection leaving the collection intact for re-population.
- Refresh — Clears data in the specified collection prior to re-populating it with new data.
- Optimize — Optimizes the specified collection of files. This action is deprecated; use CFCOLLECTION instead.

TYPE

Optional. Specifies the type of entity being indexed. Default is CUSTOM. Valid entries are:

- File — Indexes files.
- Path — Indexes all files in specified path that pass EXTENSIONS filter.
- Custom — Indexes custom entities from a ColdFusion query.

TITLE

Required when TYPE="Custom". Specifies one of the following:

- A title for the collection
- A query column name for any TYPE and a valid query name

The TITLE attribute allows searching collections by title or displaying a separate title from the actual key.

KEY

Optional. A unique identifier reference that specifies one of the following:

- Document filename when TYPE="File"
- Fully qualified path when TYPE="Path"
- A unique identifier when TYPE="Custom", such as the table column holding the primary key
- A query column name for any other TYPE argument

BODY

Optional. ASCII text to index or a query column name. Required if TYPE="Custom". Ignored for TYPE="File" and TYPE="Path". Invalid if TYPE="Delete". Specifies one of the following:

- The ASCII text to be indexed
- A query column name when a valid query name is specified in QUERY

Multiple columns can be specified in a comma-separated list:

```
BODY="employee_name, dept_name, location"
```

CUSTOM1

Optional. A custom field you can use to store data during an indexing operation. Specify a query column name for any TYPE and a valid query name.

CUSTOM2

Optional. A second custom field you can use to store data during an indexing operation. Usage is the same as for CUSTOM1.

URLPATH

Optional. Specifies the URL path for files when TYPE="File" and TYPE="Path". When the collection is searched with CFSEARCH, this path name will automatically be prepended to all file names and returned as the URL attribute.

EXTENSIONS

Optional. Specifies the comma-separated list of file extensions that ColdFusion uses to index files when TYPE="Path". Default is HTM, HTML, CFM, CFML, DBM, DBML. An entry of "*." returns files with no extension:

```
EXTENSIONS=".htm, .html, .cfm, .cfml, *."
```

Returns files with the specified extensions as well as files with no extension.

QUERY

Optional. Specifies the name of the query against which the collection is being generated.

RECURSE

Optional. Yes or No. Yes specifies that directories below the path specified in KEY when TYPE="Path" will be included in the indexing operation.

EXTERNAL

Optional. Yes or No. Yes indicates that the collection specified in COLLECTION was created outside of ColdFusion using native Verity indexing tools.

LANGUAGE

Optional. To use the LANGUAGE attribute you must have the ColdFusion International Search Pack installed. Valid entries are:

- English (default)
- German
- Finnish
- French
- Danish
- Dutch
- Italian
- Norwegian
- Portuguese
- Spanish
- Swedish

Example

```
<!--- This example shows how to utilize CFINDEX
to populate an existing collection with content --->
<HTML>
<HEAD>
<TITLE>
CFINDEX Example
</TITLE>
</HEAD>
<BODY bgcolor=silver>
<H3>CFINDEX Example</H3>
```

```
<!--- To index the collection, select the check box on the form --->
<CFIF IsDefined("form.IndexCollection")>
<CFINDEX ACTION="UPDATE" COLLECTION="Snippets"
  KEY="c:\inetpub\wwwroot\cfdocs\snippets" TYPE="PATH" TITLE="Test"
    URLPATH="http://127.0.0.1/cfdocs/snippets/" EXTENSIONS=".cfm"
      RECURSE="Yes">
...
```

CFINPUT

CFINPUT is used inside CFFORM to place radio buttons, checkboxes, or text boxes. Provides input validation for the specified control type.

You can add standard HTML FORM tag attributes and their values to the CFINPUT tag. These attributes and values are passed directly through ColdFusion to the browser in creating a form. For example, FORM tag attributes like TARGET can be used to enhance your CFFORM features.

CFINPUT supports the JavaScript onClick event in the same manner as the HTML INPUT tag:

```
<CFINPUT TYPE="radio"
    NAME="radio1"
    onClick="JavaScript_function">
```

Syntax

```
<CFINPUT TYPE="input_type"
    NAME="name"
    VALUE="initial_value"
    REQUIRED="Yes/No"
    RANGE="min_value, max_value"
    VALIDATE="data_type"
    ONVALIDATE="javascript_function"
    MESSAGE="validation_msg"
    ONERROR="text"
    SIZE="integer"
    MAXLENGTH="integer"
    CHECKED="Yes/No">
```

TYPE
Optional. Valid entries are:

- Text — Creates a text entry box control (default).
- Radio — Creates a radio button control.
- Checkbox — Creates a checkbox control.
- Password — Creates a password entry control.

NAME
Required. A name for the form input element.

VALUE
Optional. An initial value for the form input element.

REQUIRED
Optional. Enter Yes or No. Default is No.

RANGE
Optional. Enter a minimum value, maximum value range separated by a comma. Valid only for numeric data.

VALIDATE
Optional. Valid entries are:

- date — Verifies US date entry in the form mm/dd/yyy.
- eurodate — Verifies valid European date entry in the form dd/mm/yyyy.
- time — Verifies a time entry in the form *hh:mm:ss*.
- float — Verifies a floating point entry.
- integer — Verifies an integer entry.
- telephone — Verifies a telephone entry. Telephone data must be entered as ###-###-####. The hyphen separator (-) can be replaced with a blank. The area code and exchange must begin with a digit between 1 and 9.
- zipcode — (U.S. formats only) Number can be a 5-digit or 9-digit zip in the form #####-####. The hyphen separator (-) can be replaced with a blank.
- creditcard — Blanks and dashes are stripped and the number is verified using the mod10 algorithm.
- social_security_number — Number must be entered as ###-##-####. The hyphen separator (-) can be replaced with a blank.

ONVALIDATE
Optional. The name of a valid JavaScript function used 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 and false otherwise. When used, the VALIDATE attribute is ignored.

MESSAGE
Optional. Message text to appear if validation fails.

ONERROR
Optional. The name of a valid JavaScript function you want to execute in the event of a failed validation.

SIZE
Optional. The size of the input control. Ignored if TYPE is Radio or Checkbox.

MAXLENGTH
Optional. The maximum length of text entered when TYPE is Text.

Examples

```
<!--- This example shows the use of CFINPUT to validate input --->
<HTML>
<HEAD>
<TITLE>
CFINPUT Example
</TITLE>
</HEAD>
```

```
<BODY bgcolor=silver>
<H3>CFINPUT Example</H3>

<!--- this example shows the use of CFINPUT within a CFFORM to
ensure simple validation of text items --->
<CFFORM ACTION="cfinput.cfm" METHOD="POST" ENABLECAB="Yes">

<!--- phone number validation --->
Phone Number Validation (enter a properly formatted phone number):
<BR><CFINPUT TYPE="Text" NAME="MyPhone" MESSAGE="Please 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="Please 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="Please enter a correctly
formatted date (dd/mm/yy)" VALIDATE="date" REQUIRED="No">

<INPUT TYPE="Submit" NAME="" VALUE="send my information">
</CFFORM>

</BODY>
</HTML>
```

CFINSERT

CFINSERT inserts new records in data sources.

Syntax

```
<CFINSERT DATASOURCE="ds_name"
    DBTYPE="type"
    DBSERVER="dbms"
    DBNAME="database name"
    TABLENAME="tbl_name"
    TABLEOWNER="owner"
    TABLEQUALIFIER="tbl_qualifier"
    USERNAME="username"
    PASSWORD="password"
    PROVIDER="COMProvider"
    PROVIDERDSN="datasource"
    FORMFIELDS="formfield1, formfield2, ...">
```

DATASOURCE
Required. Name of the data source that contains your table.

DBTYPE
Optional. The database driver type:

- ODBC (default) — ODBC driver.
- Oracle73 — Oracle 7.3 native database driver. Using this option, the ColdFusion Server computer must have Oracle 7.3.3 (or greater) client software installed.
- Oracle80 —Oracle 8.0 native database driver. Using this option, the ColdFusion Server computer must have Oracle 8.0 (or greater) client software installed.
- Sybase11 —Sybase System 11 native database driver. Using this option, the ColdFusion Server computer must have Sybase 11.1.1 (or greater) client software installed. Sybase patch ebf 7729 is recommended.

DBSERVER
Optional. For native database drivers, specifies the name of the database server machine. If specified, DBSERVER overrides the server specified in the data source.

DBNAME
Optional. The database name (Sybase System 11 driver only). If specified, DBNAME overrides the default database specified in the data source.

TABLENAME
Required. Name of the table you want the form fields inserted in. Note the following:

- ORACLE drivers — This specification must be in uppercase.
- Sybase driver — This specification is case-sensitive and must be in the same case as that used when the 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 the qualifier for the table. The purpose of table qualifiers varies across drivers. For SQL Server and Oracle, the qualifier refers to the name of the database that contains the table. For the Intersolv dBase driver, the qualifier refers to the directory where the DBF files are located.

USERNAME
Optional. If specified, USERNAME overrides the username value specified in the ODBC setup.

PASSWORD
Optional. If specified, PASSWORD overrides the password value specified in the ODBC setup.

PROVIDER
Optional. COM provider (OLE-DB only).

PROVIDERDSN
Optional. Data source name for the COM provider (OLE-DB only).

FORMFIELDS
Optional. A comma-separated list of form fields to insert. If this attribute is not specified, all fields in the form are included in the operation.

Examples

```
<!--- This example shows how to use CFINSERT instead of CFQUERY
to place data into a datasource. --->
<!--- if form.POSTED exists, we are inserting a new record,
so begin the 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>

<!--- 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>
<TITLE>
CFINSERT Example
</TITLE>
```

```
</HEAD>

<BODY bgcolor=silver>
<H3>CFINSERT Example</H3>

<P>First, we'll show a list of the available 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 your own 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>

</BODY>
</HTML>
```

CFLDAP

CFLDAP provides an interface to LDAP (Lightweight Directory Access Protocol) directory servers like the Netscape Directory Server. For complete examples of CFLDAP usage, refer to *Advanced ColdFusion Development*.

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="sort_order"
    DN="distinguished_name"
    STARTROW="row_number">
```

SERVER
Required. Host name ("biff.upperlip.com") or IP address ("192.1.2.225") of the LDAP server.

PORT
Optional. Port defaults to the standard LDAP port, 389.

USERNAME
Optional. If no user name is specified, the LDAP connection will be anonymous.

PASSWORD
Optional. Password corresponds to user name.

ACTION
Optional. Specifies the LDAP action. There are five possible values:

- Query — (Default) Returns LDAP entry information only. Requires NAME, START, ATTRIBUTES attributes.
- Add — Adds LDAP entries to the LDAP server. Requires ATTRIBUTES.
- Modify — Modifies LDAP entries on an LDAP server with the exception of the distinguished name ("DN") attribute. Requires DN, ATTRIBUTES.
- ModifyDN — Modifies the distinguished name attribute for LDAP entries on an LDAP server. Requires DN, ATTRIBUTES.
- Delete — Deletes LDAP entries on an LDAP server. Requires DN.

NAME

Required for ACTION="Query". The name you assign to the LDAP query.

TIMEOUT
Optional. Specifies the maximum amount of time in seconds to wait for LDAP processing. Defaults to 60 seconds.

MAXROWS
Optional. Specifies the maximum number of entries for LDAP queries.

START
Required for ACTION="Query". Specifies the distinguished name of the entry to be used to start the search.

SCOPE
Optional. Specifies the scope of the search from the entry specified in the Start attribute for ACTION="Query". There are three possible values:

- OneLevel — (Default) Searches all entries one level beneath the entry specified in the START attribute.
- Base — Searches only the entry specified in the START attribute.
- Subtree — Searches the entry specified in the START attribute as well all entries at all levels beneath it.

ATTRIBUTES
Required for ACTION="Query", Add, ModifyDN, and Modify. For queries, specifies the comma-separated list of attributes to be returned for queries. Can also be used to specify the list of update columns for ACTION="Add" or Modify. When used with ACTION="Add" and Action="Modify", separate multiple attributes with a semicolon. When used with ACTION="ModifyDN", ColdFusion passes attributes to the LDAP server without performing any syntax checking.

FILTER
Optional. Specifies the search criteria for ACTION="Query". Attributes are referenced in the form: "(attribute operator value)". Example: "(sn=Smith)". Default is "objectclass=*".

SORT
Optional. Specifies the attribute to sort query results by. Enter Asc for an ascending sort and Desc for a descending sort.

DN
Required for ACTION="Add", Modify, ModifyDN, and Delete. Specifies the distinguished name for update actions. Example: "cn=Barbara Jensen, o=Ace Industry, c=US".

Examples

```
<!---  This example shows the use of CFLDAP --->
<HTML>
<HEAD>
<TITLE>CFLDAP Example</TITLE>
```

```
</HEAD>

<BODY bgcolor=silver>
<H3>CFLDAP Example</H3>

<P>CFLDAP provides an interface to LDAP (Lightweight Directory Access
Protocol) 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>
</BODY>
</HTML>
```

CFLOCATION

CFLOCATION opens a specified ColdFusion page or HTML file. For example, you might use CFLOCATION to specify a standard message or response that you use in several different ColdFusion applications. Use the ADDTOKEN attribute to verify client requests.

Syntax

```
<CFLOCATION URL="url" ADDTOKEN="Yes/No">
```

URL
The URL of the HTML file or CFML page you want to open.

ADDTOKEN
Optional. Yes or No. CLIENTMANAGEMENT must be enabled (see CFAPPLICATION). A value of Yes appends client variable information to the URL you specify in the URL argument.

Example

```
<!--- This view only example shows the use of CFLOCATION --->
<HTML>
<HEAD>
<TITLE>CFLOCATION Example</TITLE>
</HEAD>

<BODY>
<H3>CFLOCATION Example</H3>
<P>CFLOCATION redirects the browser to a specified web resource;
normally, you would use this tag to go to another CF template or to
an HTML file on the same server.  The ADDTOKEN attribute allows you to
send client information to the target page.
<P>The following is example code to direct you back to
the CFDOCS home page (remove the comments and this information will
display within the frame):
<!--- <CFLOCATION URL="../../cfdocs/index.htm" ADDTOKEN="No"> --->

</BODY>
</HTML>
```

CFLOCK

The CFLOCK tag single-threads access to the CFML constructs in its body. Single-threaded access implies that the body of the tag can be executed by at most one request at a time. A request executing inside a CFLOCK tag has an "exclusive lock" on the tag. No other requests are allowed to start executing inside the tag while a request has an exclusive lock. ColdFusion issues exclusive locks on a first-come first-serve basis.

Syntax

```
<CFLOCK NAME="lockname"
    TIMEOUT="timeout in seconds "
    THROWONTIMEOUT="Yes/No">
    <!--- CFML to be synchronized --->
</CFLOCK>
```

NAME
Optional. Specifies the name of the lock. Only one request will be able to execute inside a CFLOCK tag with a given name. Therefore, providing the NAME attribute allows for synchronizing access to the same resources from different parts of an application. Lock names are global to a ColdFusion server. They are shared between applications and user sessions, but not across clustered servers. If the NAME attribute is not provided, ColdFusion creates an anonymous lock every time CFLOCK is used in a page.

TIMEOUT
Required. Specifies the maximum amount of time in seconds to wait to obtain an exclusive lock. If an exclusive lock can obtained within the specified period, execution will continue inside the body of the tag. Otherwise, the behavior depends on the value of the THROWONTIMEOUT attribute.

THROWONTIMEOUT
Optional. Yes or No. Specifies how timeout conditions should be handled. If the value is Yes an exception will be generated to provide notification of the timeout. If the value is No execution continues past the </CFLOCK> tag. Default is Yes.

Usage

ColdFusion Server is a multi-threaded web application server that can process multiple page requests at any given time. Use CFLOCK to guarantee that multiple concurrently executing requests do not manipulate shared data structures, files, or CFXs in an inconsistent manner. Note the following:

- Using CFLOCK around CFML constructs that modify shared data ensures that the modifications occur one after the other and not all at the same time.
- As a general rule, you should use the CFLOCK tag to perform updates to variables in the Application, Server, and Session scopes.
- Using CFLOCK around file manipulation constructs can guarantee that file updates do not fail due to files being open for writing by other applications or ColdFusion tags.

- Using CFLOCK around CFX invocations can guarantee that CFXs that are not implemented in a thread-safe manner can be safely invoked by ColdFusion. This usually only applies to CFXs developed in C++ using the CFAPI. Any C++ CFX that maintains and manipulates shared (global) data structures will have to be made thread-safe to safely work with ColdFusion. However, writing thread-safe C++ CFXs requires advanced knowledge. A CFML custom tag wrapper can be used around the CFX to make its invocation thread-safe.

Note: CFLOCK uses a kernel level synchronization object that is released automatically upon timeout and/or abnormal termination of the thread that owns it. Therefore, ColdFusion will never deadlock while processing a CFLOCK tag. However, very large timeouts can block request threads for long periods of time and thus radically decrease throughput. Always use the minimum timeout value allowed.

Example

```
<!--- This example shows how CFLOCK can be used to guarantee the
consistency of data updates to variables in the Application, Server, and
Session scopes. The following code might be part of Application.cfm. --->
<HTML>
<HEAD>CFLOCK Example</HEAD>

<BODY>
<H3>CFLOCK Example</H1>

<CFLOCK NAME="ApplicationData" TIMEOUT=30>
    <CFIF NOT IsDefined("Application.IsApplicationDataInitialized")>
        <CFSET Application.IsApplicationDataInitialized = TRUE>
        <CFSET Application.ImportantValue = 5>
    </CFIF>
</CFLOCK>
<CFOUTPUT>
    Important value is #Application.ImportantValue#
</CFOUTPUT>

</BODY>
</HTML>
```

CFLOOP

Looping is a very powerful programming technique that lets you repeat a set of instructions or display output over and over until one or more conditions are met. CFLOOP supports five different types of loops:

The type of loop is determined by the attributes of the CFLOOP tag.

Index Loops

An index loop repeats for a number of times determined by a range of numeric values. Index loops are commonly known as FOR loops, as in "loop FOR this range of values."

Syntax

```
<CFLOOP INDEX="parameter_name"
    FROM="beginning_value"
    TO="ending_value"
    STEP="increment">
    ...
    HTML or CFML code to execute
    ...
</CFLOOP>
```

INDEX
Required. Defines the parameter that is the index value. The index value will be set to the FROM value and then incremented by 1 (or the STEP value) until it equals the TO value.

FROM
Required. The beginning value of the index.

TO
Required. The ending value of the index.

STEP
Optional. Default is 1. Sets the value by which the loop INDEX value is incremented each time the loop is processed.

Examples

In this example, the INDEX variable is incremented for each iteration of the loop. The following code loops five times, displaying the INDEX value of the loop each time:

```
<CFLOOP INDEX="LoopCount"
```

```
    FROM="1" TO="5">
The loop index is <CFOUTPUT>#LoopCount#</CFOUTPUT>.<BR>

</CFLOOP>
```

The result of this loop in a browser looks like this:

```
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 STEP value has a default value of 1. But you can set the STEP value to change the way the INDEX value is incremented. The following code counts backwards from 5:

```
<CFLOOP INDEX="LoopCount"
    FROM="5"
    TO="1"
    STEP="-1">
The loop index is <CFOUTPUT>#LoopCount#</CFOUTPUT>.<BR>

</CFLOOP>
```

The result of this loop in a browser looks like this:

```
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.
```

Conditional Loops

A conditional loop iterates over a set of instructions while a given condition is TRUE. To use this type of loop correctly, the instructions must change the condition every time the loop iterates until the condition evaluates as FALSE. Conditional loops are commonly known as WHILE loops, as in "loop WHILE this condition is true."

Syntax

```
<CFLOOP CONDITION="expression">
```

CONDITION
Required. Sets the condition that controls the loop. The loop will repeat as long as the condition evaluates as TRUE. When the condition is FALSE, the loop stops.

Example

The following example increments the parameter "CountVar" from 1 to 5. The results look exactly like the Index loop example.

```
<!-- 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 result of this loop in a browser would look something like:

```
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.
```

Looping over a Query

A loop over a query repeats for every record in the query record set. The CFLOOP results are just like a CFOUTPUT. During each iteration of the loop, the columns of the current row will be available for output. CFLOOP allows you to loop over tags that can not be used inside CFOUTPUT.

Syntax

```
<CFLOOP QUERY="query_name"
    STARTROW="row_num"
    ENDROW="row_num">
```

QUERY
Required. Specifies the query that will control the loop.

STARTROW
Optional. Specifies the first row of the query that will be included in the loop.

ENDROW
Optional. Specifies the last row of the query that will be included in the loop.

Examples

Example 1

The following example shows a CFLOOP looping over a query that works in the same way as a CFOUTPUT tag using the QUERY attribute:

```
<CFQUERY NAME="MessageRecords"
    DATASOURCE="cfsnippets">
    SELECT * FROM Messages
</CFQUERY>

<CFLOOP QUERY="MessageRecords">
    <CFOUTPUT>#Message_ID#</CFOUTPUT><BR>
</CFLOOP>
```

Example 2

CFLOOP also provides iteration over a recordset with dynamic starting and stopping points. Thus you can begin at the tenth row in a query and end at the twentieth. This mechanism provides a simple means to get the next *n* sets of records from a query. The following example loops from the tenth through the twentieth 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 is done when there are no more records or when the current record is greater than the value of the ENDROW attribute.

Example 3

The advantage of looping over a query is that you can use CFML tags that are not allowed in a CFOUTPUT. The following example combines the pages returned by a query of a list of page names into a single document using the CFINCLUDE tag.

```
<CFQUERY NAME="GetTemplate"
    DATASOURCE="Library"
    MAXROWS="5">
    SELECT TemplateName FROM Templates
</CFQUERY>

<CFLOOP QUERY="TemplateName">
    <CFINCLUDE TEMPLATE="#TemplateName#">
</CFLOOP>
```

Looping over a List

Looping over a list offers the option of walking through elements contained within a variable or value returned from an expression. In a list loop, the INDEX attribute specifies the name of a variable to receive the next element of the list, and the LIST attribute holds a list or a variable containing a list.

Syntax

```
<CFLOOP INDEX="index_name"
    LIST="list_items"
    DELIMITERS="item_delimiter">
</CFLOOP>
```

INDEX

Required. Defines the parameter that is the index value. The index value will be set to the FROM value and then incremented by 1 (or the STEP value) until it equals the TO value.

LIST
Required. The list items in the loop, provided directly or with a variable.

DELIMITERS
Optional. Specifies the delimiter characters used to separate items in the LIST.

Example This loop will display the names of each of the Beatles:

```
<CFLOOP INDEX="ListElement"
    LIST="John,Paul,George,Ringo">
        <CFOUTPUT>#ListElement#</CFOUTPUT><BR>
</CFLOOP>
```

Although CFLOOP expects elements in the list to be separated by commas by default, you are free to specify your own element boundaries in the DELIMITER attribute. Here's the same loop as before, only this time CFLOOP will treat commas, colons, or slashes as list element delimiters:

```
<CFLOOP INDEX="ListElement"
    LIST="John/Paul,George::Ringo"
    DELIMITERS=",:/">
        <CFOUTPUT>#ListElement#</CFOUTPUT><BR>
</CFLOOP>
```

Delimiters need not be specified in any particular order. Note that consecutive delimiters are treated as a single delimiter; thus the two colons in the previous example are treated as a single delimiter between "George" and "Ringo."

Looping over a COM Collection or Structure

The CFLOOP COLLECTION attribute allows you to loop over a structure or a COM/DCOM collection object:

- A COM/DCOM collection object is a set of similar items referenced as a group rather than individually. For example, the group of open documents in an application is a type of collection.
- A structure can contain either a related set of items or be used as an associative array. Looping is particularly useful when using a structure as an associative array.

Each collection item is referenced in the CFLOOP by the variable name you supply in the ITEM attribute. This type of an iteration is generally used to access every object within a COM/DCOM collection or every element in the structure. The loop is executed until all objects have been accessed.

The COLLECTION attribute is used with the ITEM attribute in a CFLOOP. 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.

Examples

This example employs 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>
```

This example loops through a structure (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>

<!--- In CFLOOP, use ITEM to create a variable
    called 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

CFMAIL allows you to generate email messages and post them to a specified server.

Syntax

```
<CFMAIL TO="recipient"
    FROM="sender"
    CC="copy_to"
    SUBJECT="msg_subject"
    TYPE="msg_type"
    QUERY="query_name"
    MAXROWS="max_msgs"
    MIMEATTACH="path"
    GROUP="query_column"
    STARTROW="query_row"
    SERVER="servername"
    PORT="port_ID"
    MAILERID="headerid"
    TIMEOUT="seconds">
```

TO

Required. The name of the recipient(s) of the email message. This can be either a static address (as in, TO="support@allaire.com"), a variable that contains an address (such as, TO="#Form.Email#"), or the name of a query column that contains address information (such as, TO="#EMail#"). In the latter case, an individual email message is sent for every row returned by the query.

FROM

Required. The sender of the email message. This attribute may be either static (e.g., FROM="support@allaire.com") or dynamic (as in, FROM="#GetUser.EMailAddress#").

CC

Optional. Indicates additional addresses to copy the email message to. This attribute may also be static or dynamic.

SUBJECT

Required. The subject of the mail message. This field may be driven dynamically on a message-by-message basis. For example, if you want to do a mailing that updates customers on the status of their orders, you might use a subject attribute like SUBJECT="Status for Order Number #Order_ID#".

TYPE

Optional. Specifies extended type attributes for the message. Currently, the only valid value for this attribute is "HTML". Specifying TYPE= "HTML" informs the receiving email client that the message has embedded HTML tags that need to be processed. This is only useful when sending messages to mail clients that understand HTML (such as Netscape 2.0 and above email clients).

QUERY

Optional. The name of the CFQUERY from which you want to draw data for message(s) you want to send. Specify this attribute to send more than one mail message, or to send the results of a query within a single message.

MAXROWS
Optional. Specifies the maximum number of email messages you want to send.

MIMEATTACH
Optional. Specifies the path of the file to be attached to the email message. Attached file is MIME-encoded.

GROUP
Optional. Specifies the query column to use when you group sets of records together to send as a single email message. For example, if you send a set of billing statements out to your customers, you might group on "Customer_ID." The GROUP parameter, which is case sensitive, eliminates adjacent duplicates in the case where the data is sorted by the specified field.

STARTROW
Optional. Specifies the row in the query to start from.

SERVER
Required. The address of the SMTP server to use for sending messages. The server name specified in the ColdFusion Administrator is used if no server is specified.

PORT
The TCP/IP port on which the SMTP server listens for requests. This is almost always 25.

MAILERID
Optional. Specifies a mailer ID to be passed in the X-Mailer SMTP header, which identifies the mailer application. The default is Allaire ColdFusion Application Server.

TIMEOUT
Optional. The number of seconds to wait before timing out the connection to the SMTP server.

Example

```
<!--- This view-only example shows the use of CFMAIL --->
<HTML>
<HEAD>
<TITLE>CFMAIL Example</TITLE>
</HEAD>

<BODY bgcolor=silver>
<H3>CFMAIL Example</H3>
<P>This view-only example shows the use of CFMAIL.  If your CFAS mail
settings are configured successfully and the comments are removed,
you will be able to use this code to send simple email.
<!---  <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" METHOD="POST">
<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 for this message">
<INPUT TYPE="Hidden" NAME="MailFrom_required" VALUE="You must enter a
  sender for this message">
<INPUT TYPE="Hidden" NAME="Subject_required" VALUE="You must enter a
  subject for this message">
<INPUT TYPE="Hidden" NAME="Body_required" VALUE="You must enter some text
  for this message">

<P><INPUT TYPE="Submit" NAME="">
</FORM>  --->

</BODY>
</HTML>
```

CFMODULE

Use CFMODULE to invoke a custom tag for use in your ColdFusion application pages. CFMODULE can help deal with any custom tag name conflicts that might arise.

Use the TEMPLATE attribute to name a ColdFusion page containing the custom tag definition, including its path. Use the NAME attribute to refer to the custom tag using a dot notation scheme indicating the location of the custom tag in the ColdFusion installation directory.

Syntax

```
<CFMODULE TEMPLATE="template"
    NAME="tag_name"
    ATTRIBUTE="value"
    ATTRIBUTE="value"
    ...>
```

TEMPLATE
Used in place of NAME, defines a path to the application page (.cfm file) implementing the tag. Relative paths are expanded from the current page. Physical paths are not allowed. Absolute paths are expanded using the ColdFusion mappings.

NAME
Used in place of TEMPLATE, defines the name of the custom tag in the form "Name.Name.Name..." that uniquely identifies a subdirectory containing the custom tag page under the root directory for CF custom tags. For example:

`<CFMODULE Name="Allaire.Forums40.GetUserOptions">`
Identifies the page `GetUserOptions.cfm` in the directory `CustomTags\Allaire\Forums40` under the root directory of the ColdFusion installation.

ATTRIBUTE
Optional. Attributes you want your custom tag to use.

Example

```
<!--- This view-only example shows the use of CFMODULE --->
<HTML>
<HEAD>
<TITLE>CFMODULE Example</TITLE>
</HEAD>

<BODY>
<H3>CFMODULE Example</H3>

<P>This example shows the use of CFMODULE to call a sample custom
tag inline.
<P>To activate this example, you will need to save the sample
custom tag into a CustomTags/SampleTag directory under ColdFusion.
<!---
<!--- sample custom tag (save this in the cfusion\CustomTags
```

```
directory --->
<!--- this tag takes two parameters, adds them and
returns the result --->
<CFSET X = attributes.x>
<CFSET Y = attributes.y>
<CFSET caller.result = x + y>
<!--- end sample tag --->

<!--- Call the tag with CFMODULE--->
<CFMODULE
    NAME="sampleTag.myTag"
    X=3
    Y=4>

<!--- show the code --->
<CFOUTPUT>#HTMLCodeFormat("<CFMODULE
    NAME='sampleTag.myTag'
    X=3
    Y=4>")#
</CFOUTPUT>
<P>The result: <CFOUTPUT>#result#</CFOUTPUT>

--->

</BODY>
</HTML>
```

CFOBJECT

The CFOBJECT tag allows you to call methods in COM and CORBA objects.

Note ColdFusion administrators can disable the CFOBJECT tag in the ColdFusion Administrator Basic Security page.

CFOBJECT topics

- CFOBJECT Type="COM"
- CFOBJECT Type="CORBA"

CFOBJECT TYPE attributes

Depending on the value you assign to the TYPE attribute of CFOBJECT, there are several additional attributes you can set. This table shows which attributes you can use with each CFOBJECT TYPE.

Attributes Used with CFOBJECT TYPEs	
TYPE	**Attributes**
COM	ACTION CLASS NAME CONTEXT SERVER
CORBA	CONTEXT CLASS NAME

Sections that follow describe these values and attributes in greater detail.

CFOBJECT Type="COM"

CFOBJECT allows you to create and use COM (Component Object Model) objects. Any automation server object type that is currently registered on a machine can be invoked. You can use a utility like Microsoft's OLEView to browse COM objects. OLEView, as well as information about COM and DCOM, can be found at Microsoft's OLE Development web site `http://www.microsoft.com/oledev/`.

To use CFOBJECT, you need to know the program ID or filename of the object, the methods and properties available through the IDispatch interface, and the arguments and return types of the object's methods. The OLEView utility can give you this information for most COM objects.

Syntax

```
<CFOBJECT TYPE="COM"
    ACTION="action"
    CLASS="program_ID"
    NAME="text"
    CONTEXT="context"
    SERVER="server_name">
```

ACTION
Required. One of the following:

- Create — Use Create to instantiate a COM object (typically a DLL) prior to invoking methods or properties.
- Connect — Use Connect to connect to a COM object (typically an EXE) that is already running on the server specified in SERVER.

CLASS
Required. Enter the component ProgID for the object you want to invoke.

NAME
Required. Enter a name for the object.

CONTEXT
Optional. InProc, Local, or Remote. Uses Registry setting when not specified.

SERVER
Required when CONTEXT="Remote". Enter a valid server name using UNC (Universal Naming Convention) or DNS (Domain Name Server) conventions, in one of the following forms:

```
SERVER="\\lanserver"
SERVER="lanserver"
SERVER="http://www.servername.com"
SERVER="www.servername.com"
SERVER="127.0.0.1"
```

Example

```
<HTML>
<HEAD>
<TITLE>CFOBJECT (COM) Example</TITLE>
</HEAD>

<BODY>
<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.
Note that methods that expect no arguments should
be called using empty parenthesis.
--->
<CFSET obj.Init()>

<!---
This object is a collection object, and 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>
...
</BODY>
</HTML>
```

CFOBJECT Type="CORBA"

CFOBJECT allows you to call methods in CORBA objects. These CORBA objects must already have been defined and registered for use.

Syntax

```
<CFOBJECT TYPE="CORBA"
    CONTEXT="context"
    CLASS="file or naming service"
    NAME="text">
```

CONTEXT
Required. Specifies one of the following:

- IOR — ColdFusion uses the Interoperable Object Reference (IOR) to access the CORBA server.
- NameService — ColdFusion uses the naming service to access server.

CLASS
Required. Specifies different information, depending on the CONTEXT specification:

- If CONTEXT is IOR — Specifies the name of a file that contains the stringified version of the IOR. ColdFusion must be able to read this file at all times; it should be local to ColdFusion server or on the network in an open, accessible location.
- If CONTEXT is NameService — Specifies a period-delimited naming context for the naming service, such as Allaire.Department.Doc.empobject.

NAME
Required. Enter a name for the object. Your application uses this to reference the CORBA object's methods and attributes.

Usage

ColdFusion Enterprise version 4.0 supports CORBA through the Dynamic Invocation Interface (DII). To use CFOBJECT with CORBA objects, you need to know either the name of the file containing a stringified version of the IOR or the object's naming context in the naming service. You also need to know 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">
```

CFOUTPUT

Displays the results of a database query or other operation. You cannot nest CFOUTPUT tags.

Syntax

```
<CFOUTPUT QUERY="query_name"
    MAXROWS="max_rows_output"
    GROUP="parameter"
    STARTROW="start_row">

</CFOUTPUT>
```

QUERY
Optional. The name of the CFQUERY from which you want to draw data for the output section.

MAXROWS
Optional. Specifies the maximum number of rows you want displayed in the output section.

GROUP
Optional. Specifies the parameter around which to group output. The GROUP parameter eliminates sequential duplicates in the case where data is sorted by the specified field.

STARTROW
Optional. Specifies the row from which to start output.

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>
<HTML>
<HEAD>
<TITLE>CFOUTPUT</TITLE>
</HEAD>
<BODY>
<H3>CFOUTPUT Example</H3>

<P>CFOUTPUT simply 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>

<P>
</BODY>
</HTML>
```

CFPARAM

CFPARAM is used to test for a parameter's existence and optionally provide a default if it is not found.

Syntax

```
<CFPARAM NAME="param_name"
    DEFAULT="value">
```

NAME
The name of the parameter you are testing (such as "Client.Email" or "Cookie.BackgroundColor"). If you omit the DEFAULT attribute, an error occurs if the specified parameter does not exist.

DEFAULT
Optional. Default value to set the parameter to if it does not exist.

Usage

There are two ways to use CFPARAM:

- Test for a required variable — Use CFPARAM with only the NAME attribute to test that a required variable exists. If the variable does not exist, ColdFusion server stops processing the page and returns an error.
- Test for an optional variable — Use CFPARAM with both the NAME and DEFAULT attributes to test for the existence of an optional variable. If the variable exists, processing continues and the value is not changed. If the variable does not exist, it is created and set to the value of the DEFAULT attribute.

Example

```
<!--- This example shows how CFPARAM operates --->
<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>

<HTML>
<HEAD>
<TITLE>
CFPARAM Example
</TITLE>
</HEAD>

<BODY bgcolor=silver>
```

```
<H3>CFPARAM Example</H3>
<P>CFPARAM is used to set default values so that
the developer does not need 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>

</BODY>
</HTML>
```

CFPOP

CFPOP retrieves and deletes email messages from a POP mail server. See also CFMAIL.

Syntax

```
<CFPOP SERVER="servername"
    PORT="port_number"
    USERNAME="username"
    PASSWORD="password"
    ACTION="action"
    NAME="queryname"
    MESSAGENUMBER="number"
    ATTACHMENTPATH="path"
    TIMEOUT="seconds"
    MAXROWS="number"
    STARTROW="number">
```

SERVER
Required. Host name (biff.upperlip.com) or IP address (192.1.2.225) of the POP server.

PORT
Optional. Defaults to the standard POP port, 110.

USERNAME
Optional. If no user name is specified, the POP connection is anonymous.

PASSWORD
Optional. Password corresponds to user name.

ACTION
Optional. Specifies the mail action. There are three possible values:

- GetHeaderOnly — (Default) Returns message header information only.
- GetAll — Returns message header information, message text, and attachments if ATTACHMENTPATH is specified.
- Delete — Deletes messages on the POP server.

Note: Two retrieve options are offered to maximize performance. 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.

NAME
Optional. The name you assign to the index query. Required for ACTION="GetHeaderOnly" and ACTION="GetAll".

MESSAGENUMBER
Optional. Specifies the message number(s) for the given action. MESSAGENUMBER is required for ACTION="Delete". If it is provided for ACTION="GetHeaderOnly" or ACTION="GetAll", only referenced messages will

be retrieved. If it is omitted for ACTION="GetHeaderOnly"or ACTION="GetAll", all messages available on the server are returned.

MESSAGENUMBER can contain individual message numbers or a comma-separated list of message numbers. Invalid message numbers will be ignored.

ATTACHMENTPATH
Optional. Allows attachments to be written to the specified directory when ACTION="GetAll". If an invalid ATTACHMENTPATH is specified, no attachment files are written to the server.

TIMEOUT
Optional. Specifies the maximum amount of time in seconds to wait for mail processing. Defaults to 60 seconds.

MAXROWS
Optional. Specifies the maximum number of entries for mail queries. This attribute is ignored if MESSAGENUMBER is specified.

STARTROW
Optional. Specifies the first row number to be retrieved. Default is 1. This attribute is ignored if MESSAGENUMBER is specified.

Usage

For complete usage information on CFPOP, see *Developing Web Applications with ColdFusion*.

Example

```
<!--- This view-only example shows the use of CFPOP --->
<HTML>
<HEAD>
<TITLE>CFPOP Example</TITLE>
</HEAD>

<BODY>
<H3>CFPOP Example</H3>
<P>CFPOP allows you to retrieve and manipulate mail
in a POP3 mailbox.  This view-only example shows how to
create one feature of a mail client, allowing you to display
the mail headers in a POP3 mailbox.

<P>Simply uncomment this code and run with a mail-enabled CF Server to
see this feature in action.
<!---
<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>
     <UL>
     <CFOUTPUT QUERY=”GetHeaders”>
```

```
     <LI>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>
--->

</BODY>
</HTML>
```

CFPROCPARAM

The CFPROCPARAM tag is nested within a CFSTOREDPROC tag. You use it to specify parameter information, including type, name, value, and length.

Syntax

```
<CFPROCPARAM TYPE="IN/OUT/INOUT"
    VARIABLE="variable name"
    DBVARNAME="DB variable name"
    VALUE="parameter value"
    CFSQLTYPE="parameter datatype"
    MAXLENGTH="length"
    SCALE="decimal places"
    NULL="yes/no">
```

TYPE
Optional. Indicates whether the passed variable is an input, output or input/output variable. Default is IN.

VARIABLE
Required for OUT and INOUT parameters. This is the ColdFusion variable name that you use to reference the value that the output parameter represents after the call is made to the stored procedure.

DBVARNAME
Required if named notation is desired. This is the parameter name. This corresponds to the name of the parameter in the stored procedure.

VALUE
Required for IN and INOUT parameters. This corresponds to the actual value that ColdFusion passes to the stored procedure.

CFSQLTYPE
Required. This is the SQL type that the parameter (any type) will be bound to. The CFSQLTypes are as follows:

CF_SQL_BIGINT	CF_SQL_IDSTAMP	CF_SQL_REAL
CF_SQL_CHAR	CF_SQL_INTEGER	CF_SQL_SMALLINT
CF_SQL_DATE	CF_SQL_LONGVARCHAR	CF_SQL_TIME
CF_SQL_DECIMAL	CF_SQL_MONEY	CF_SQL_TIMESTAMP
CF_SQL_DOUBLE	CF_SQL_MONEY4	CF_SQL_TINYINT
CF_SQL_FLOAT	CF_SQL_NUMERIC	CF_SQL_VARCHAR

MAXLENGTH
Optional. Maximum length of the parameter.

SCALE
Optional. Number of decimal places of the parameter.

NULL
Optional. Specify Yes or No. Indicates whether the parameter is passed as a NULL. If you specify Yes, the tag ignores the VALUE attribute.

Usage

Use this tag to identify stored procedure parameters and their data type. Code one CFPROCPARAM tag for each parameter. The parameters you code vary, based on parameter type and DBMS. Additionally, the order in which you code CFPROCPARAM tags matters, depending on whether the stored procedure was coded using positional notation or named notation:

- Positional notation — Order is very important if the stored procedure was defined using positional notation. ColdFusion passes these parameters to the stored procedure in the order in which they are defined.
- Named notation — If named notation is used, the DBVarName for the parameter must correspond to the variable name in the stored procedure on the server.

Output variables will be scoped with the name of the VARIABLE attribute that was passed to the tag.

Example

```
...
<!--- The following 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 tag --->
<CFSTOREDPROC PROCEDURE="foo_proc"
    DATASOURCE="MY_SYBASE_TEST"USERNAME="sa"
    PASSWORD=""DBSERVER="scup"DBNAME="pubs2"
    RETURNCODE="YES"DEBUG>
<!--- 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>
...
```

CFPROCRESULT

The CFPROCRESULT tag is nested within a CFSTOREDPROC tag. This tag's NAME parameter specifies a result set name that other ColdFusion tags, such as CFOUTPUT and CFTABLE, use to access the result set. It also allows you to optionally identify which of the stored procedure's result sets to return.

Syntax

```
<CFPROCRESULT NAME="query_name"
    RESULTSET="1-n"
    MAXROWS="maxrows">
```

NAME
Required. Name for the query result set.

RESULTSET
Optional. Specify this parameter to identify the desired result set if the stored procedure returns multiple result sets. Default is 1.

MAXROWS
Optional. Specifies the maximum number of rows returned in the result set. The default is to return all rows in the result set.

Usage

Specify one or more CFPROCRESULT tags to enable access to data returned by the stored procedure.

RESULTSET must be unique within the scope of the CFSTOREDPROC tag. If you specify the same result set twice, the second occurrence overwrites the first.

Example

```
...
<!--- The following 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 tag --->
<CFSTOREDPROC PROCEDURE="foo_proc"
    DATASOURCE="MY_SYBASE_TEST"USERNAME="sa"
    PASSWORD=""DBSERVER="scup"DBNAME="pubs2"
    RETURNCODE="YES"DEBUG>
<!--- 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>
...
```

CFQUERY

CFQUERY passes SQL statements for any purpose to your data source. Not limited to queries.

Syntax

```
<CFQUERY NAME="query_name"
    DATASOURCE="ds_name"
    DBTYPE="type"
    DBSERVER="dbms"
    DBNAME="database name"
    USERNAME="username"
    PASSWORD="password"
    MAXROWS="number"
    BLOCKFACTOR="blocksize"
    TIMEOUT="milliseconds"
    CACHEDAFTER="date"
    CACHEDWITHIN="timespan"
    PROVIDER="COMProvider"
    PROVIDERDSN="datasource"
    DEBUG="Yes/No">

SQL statements

</CFQUERY>
```

NAME

Required. The name you assign to the query. Query names must begin with a letter and may consist of letters, numbers, and the underscore character (spaces are not allowed). The query name is used later in the page to reference the query's record set.

DATASOURCE

Required. The name of the data source from which this query should retrieve data.

DBTYPE

Optional. The database driver type:

- ODBC (default) — ODBC driver.
- Oracle73 — Oracle 7.3 native database driver. Using this option, the ColdFusion Server computer must have Oracle 7.3.3 (or greater) client software installed.
- Oracle80 —Oracle 8.0 native database driver. Using this option, the ColdFusion Server computer must have Oracle 8.0 (or greater) client software installed.
- Sybase11 —Sybase System 11 native database driver. Using this option, the ColdFusion Server computer must have Sybase 11.1.1 (or greater) client software installed. Sybase patch ebf 7729 is recommended.

DBSERVER

Optional. For native database drivers, specifies the name of the database server machine. If specified, DBSERVER overrides the server specified in the data source.

DBNAME
Optional. The database name (Sybase System 11 driver only). If specified, DBNAME overrides the default database specified in the data source.

USERNAME
Optional. If specified, USERNAME overrides the username value specified in the data source setup.

PASSWORD
Optional. If specified, PASSWORD overrides the password value specified in the data source setup.

MAXROWS
Optional. Specifies the maximum number of rows you want returned in the record set.

BLOCKFACTOR
Optional. Specifies the maximum number of rows to fetch at a time from the server. The range is 1 (default) to 100. This parameter applies to ORACLE native database drivers and to ODBC drivers. Certain ODBC drivers may dynamically reduce the block factor at runtime.

TIMEOUT
Optional. Lets you specify a maximum number of milliseconds for the query to execute before returning an error indicating that the query has timed-out. This attribute is not supported by most ODBC drivers. TIMEOUT is supported by the SQL Server 6.x or above driver. The minimum and maximum allowable values vary, depending on the driver.

CACHEDAFTER
Optional. Specify a date value (for example, 4/16/98, April 16, 1998, 4-16-98). ColdFusion uses cached query data if the date of the original query is after the date specified. Effective only if query caching has been enabled in the ColdFusion Administrator. To use cached data, the current query must use the same SQL statement, data source, query name, user name, password, and DBTYPE. Additionally, for native drivers it must have the same DBSERVER and DBNAME (Sybase only).

Years from 0 to 29 are interpreted as 21st century values. Years 30 to 99 are interpreted as 20th century values.

When specifying a date value as a string, make sure it is enclosed in quotes.

CACHEDWITHIN
Optional. Enter a timespan using the ColdFusion CreateTimeSpan function. Cached query data will be used if the original query date falls within the time span you define. The CreateTimeSpan function is used to define a period of time from

the present backwards. Effective only if query caching has been enabled in the ColdFusion Administrator. To use cached data, the current query must use the same SQL statement, data source, query name, user name, password, and DBTYPE. Additionally, for native drivers it must have the same DBSERVER and DBNAME (Sybase only).

PROVIDER
Optional. COM provider (OLE-DB only).

PROVIDERDSN
Optional. Data source name for the COM provider (OLE-DB only).

DEBUG
Optional. Used for debugging queries. Specifying this attribute causes the SQL statement actually submitted to the data source and the number of records returned from the query to be output.

Example

```
<!--- This example shows the use of CFQUERY --->
<HTML>
<HEAD>
    <TITLE>CFQUERY Example</TITLE>
</HEAD>

<BODY bgcolor=silver>
<H3>CFQUERY Example</H3>

<!--- define startrow and maxrows to facilitate
'next N' style browsing --->
<CFPARAM name="Startrow" default="1">
<CFPARAM name="MAXROWS" default="10">

<!--- query database for information --->
<CFQUERY NAME="GetParks" DATASOURCE="cfsnippets">
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>
<!--- Set a counter variable equal to startrow
```

```
so that we can show visually where we are in the
total recordset --->
<CFPARAM name="Counter" Default="#StartRow#">
<!--- Output the query and define the startrow and maxrows
parameters --->
<CFOUTPUT query="GetParks" StartRow="#startrow#" MAXROWS="#maxrows#">
<TR>
    <TD valign=top bgcolor=ffffed>
    <B>#Counter#</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>
<!--- increment the counter each row through the
query output --->
<CFSET #Counter# = #Counter# + 1>
</CFOUTPUT>

<!--- If the total amount of records is less than or equal
to the startrow + maxrows value, 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="cfquery.cfm?startrow=
      <CFOUTPUT>#Evaluate("#startrow#+#maxrows#")#
       </CFOUTPUT>">See next <CFOUTPUT>#MaxRows#</CFOUTPUT> rows</A>
    </CFIF>

    </TD>
</TR>
</TABLE>

</BODY>
</HTML>
```

CFREGISTRY

The CFREGISTRY tag reads, writes, and deletes keys and values in the system registry. CFREGISTRY is supported on all platforms, including Solaris.

Note ColdFusion administrators can disable CFREGISTRY processing in the ColdFusion Administrator Basic Security page.

CFREGISTRY topics

- CFREGISTRY ACTION="GetAll"
- CFREGISTRY ACTION="Get"
- CFREGISTRY ACTION="Set"
- CFREGISTRY ACTION="Delete"

CFREGISTRY ACTION attributes

Depending on the value you assign to the ACTION attribute of CFREGISTRY, there are several additional attributes you set. This table shows which attributes you can use with each CFREGISTRY ACTION.

Attributes Used with CFREGISTRY ACTIONs

ACTION	Attributes
GetAll	BRANCH TYPE NAME SORT
Get	BRANCH ENTRY TYPE VARIABLE
Set	BRANCH ENTRY TYPE VALUE
Delete	BRANCH ENTRY

Sections that follow describe these values and attributes in greater detail.

CFREGISTRY ACTION="GetAll"

Use CFREGISTRY with the GetAll action to return all registry keys and values defined in a branch. You can access these values as you would any record set.

Syntax

```
<CFREGISTRY ACTION="GetAll"
    BRANCH="branch"
    TYPE="data type"
    NAME="query name"
    SORT="criteria">
```

BRANCH
Required. The name of the registry branch containing the keys or values you want to access.

TYPE
Optional. The type of data you want to access:

- String — Return string values (default).
- DWord — Return DWord values.
- Key — Return keys.
- Any — Return keys and values.

NAME
Required. The name of the record set to contain returned keys and values.

SORT
Optional. Sorts query column data (case-insensitive). Sorts on Entry, Type, and Value columns as text. Specify any combination of columns from query output in a comma separated list. ASC (ascending) or DESC (descending) can be specified as qualifiers for column names. ASC is the default. For example:

```
Sort="value DESC, entry ASC"
```

Usage

CFREGISTRY 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 Any for TYPE, 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  --->

<HTML>
<HEAD>
<TITLE>CFREGISTRY ACTION="GetAll"</TITLE>
</HEAD>
```

```
<BODY>
<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>
</BODY>
</HTML>
```

CFREGISTRY ACTION="Get"

Use CFREGISTRY with the Get action to access a registry value and store it in a ColdFusion variable.

Syntax

```
<CFREGISTRY ACTION="Get"
    BRANCH="branch"
    ENTRY="key or value"
    TYPE="data type"
    VARIABLE="variable">
```

BRANCH
Required. The name of the registry branch containing the value you want to access.

ENTRY
Required. The registry value to be accessed.

TYPE
Optional. The type of data you want to access:

- String — Return a string value (default).
- DWord — Return a DWord value.
- Key — Return a key's default value.

VARIABLE
Required. Variable into which CFREGISTRY places the value.

Usage

CFREGISTRY does not create the variable if the value does not exist.

Example

```
<!--- This example uses CFREGISTRY with
      the Get Action  --->
```

```
<HTML>
<HEAD>
<TITLE>CFREGISTRY ACTION="Get"</TITLE>
</HEAD>
<BODY>
<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>
</BODY>
</HTML>
```

CFREGISTRY ACTION="Set"

Use CFREGISTRY with the Set action to add a registry key, add a new value, or update value data.

Syntax

```
<CFREGISTRY ACTION="Set"
    BRANCH="branch"
    ENTRY="key or value"
    TYPE="value type"
    VALUE="data">
```

BRANCH
Required. The name of the registry branch containing the key or value to be set.

ENTRY
Required. The key or value to be set.

TYPE
Optional. The type of data you want to set:

- String — Set a string value (default).
- DWord — Set a DWord value.
- Key — Create a key.

VALUE
Optional. The value data to be set. If you omit this attribute, CFREGISTRY creates default value data, as follows:

- String — Default value is an empty string: ""
- DWord — Default value is 0 (zero)

Usage CFREGISTRY creates the key or value if it does not exist.

Example

```
<!--- This example uses CFREGISTRY with
      the Set Action to modify
      registry value data --->

<HTML>
<HEAD>
<TITLE>CFREGISTRY ACTION="Set"</TITLE>
</HEAD>
<BODY>
<!--- 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="LastCFM01" TYPE="String" VALUE="#FileName#">
<H1>CFREGISTRY ACTION="Set"</H1>
</BODY>
</HTML>
```

CFREGISTRY ACTION="Delete"

Use CFREGISTRY with the Delete action to delete a registry key or value.

Syntax

```
<CFREGISTRY ACTION="Delete"
    BRANCH="branch"
    ENTRY="keyorvalue">
```

BRANCH
Required. Specifies one of the following:

- For key deletion — The name of the registry key to be deleted. To delete a key, do not specify ENTRY.
- For value deletion — The name of the registry branch containing the value to be deleted. To delete a value, you must specify ENTRY.

ENTRY
Required for value deletion. The value to be deleted.

Usage If you delete a key, CFREGISTRY also deletes values and subkeys defined beneath the key.

Example

```
<!--- This example uses CFREGISTRY with
      the Delete Action to remove
      a key from the registry  --->
<HTML>
<HEAD>
```

```
<TITLE>CFREGISTRY ACTION="Delete"</TITLE>
</HEAD>
<BODY>
<CFREGISTRY ACTION="Delete"
  BRANCH="HKEY_LOCAL_MACHINE\Software\cflangref\tempkey"
  ENTRY="LastCFM01">
<H1>CFREGISTRY ACTION="Delete"</H1>
</BODY>
</HTML>
```

CFREPORT

CFREPORT runs a predefined Crystal Reports report.

Syntax

```
<CFREPORT REPORT="report_path"
    ORDERBY="result_order"
    USERNAME="username"
    PASSWORD="password"
    FORMULA="formula">

</CFREPORT>
```

REPORT
Required. Specifies the path of the report. Store your Crystal Reports files in the same directories that you store your ColdFusion page files.

ORDERBY
Optional. Orders results according to your specifications.

USERNAME
Optional. The username required for entry into the database from which the report is created. Overrides the default settings for the data source in the ColdFusion Administrator.

PASSWORD
Optional. The password that corresponds to a username required for database access. Overrides the default settings for the data source in the ColdFusion Administrator.

FORMULA
Optional. Specifies one or more named formulas. Terminate each formula specification with a semicolon. Use the following format:

```
FORMULA="formulaname1=formula1;formulaname2=formula2;"
```

Example

```
<!--- This view-only example shows the use of CFREPORT --->
<HTML>
<HEAD>
<TITLE>CFREPORT Example</TITLE>
</HEAD>

<BODY>
<H3>CFREPORT Tag<H3>
<P>CFREPORT allows reports from the Crystal Reports Professional
report writer to be displayed through a ColdFusion interface.
The CFREPORT tag requires the name of the report to run;
CFREPORT can also pass information to the report
file being 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 certain subset of the report.

<CFREPORT REPORT=’/reports/monthlysales.rpt’>
          {Departments.Department} = ‘International’
</CFREPORT>

<P>Substitute your own report files and filters for this code
and CFREPORT can place your existing Crystal Reports into web pages.

</BODY>
</HTML>
```

CFSCHEDULE

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 allows you to offer users access to pages that publish data, such as reports, without forcing users to wait while a database transaction is performed in order to populate the data on the page.

ColdFusion scheduled events are registered using the ColdFusion Administrator. In addition, execution of CFSCHEDULE can be disabled in the Administrator. Information supplied by the user includes the scheduled ColdFusion page to execute, the time and frequency for executing the page, and if the output from the task should be published. If the output is to be published then a path and file is specified.

The event submission and its success or failure status is written to the `\cfusion\log\schedule.log` file.

Syntax

```
<CFSCHEDULE ACTION="Update"
    TASK="taskname"
    OPERATION="HTTPRequest"
    FILE="filename"
    PATH="path_to_file"
    STARTDATE="date"
    STARTTIME="time"
    URL="URL"
    PUBLISH="Yes/No"
    ENDDATE="date"
    ENDTIME="time"
    INTERVAL="seconds"
    REQUESTTIMEOUT="seconds"
    USERNAME="username"
    PASSWORD="password"
    RESOLVEURL="Yes/No"
    PROXYSERVER="hostname">

<CFSCHEDULE ACTION="Delete" TASK="TaskName">
<CFSCHEDULE ACTION="Run" TASK="TaskName">
```

ACTION

Required. Valid entries are:

- Delete – Deletes task specified by TASK.
- Update – Creates a new task if one does not exist.
- Run – Executes task specified by TASK.

TASK

Required. The name of the task to delete, update, or run.

OPERATION

Required when creating tasks with ACTION="Update". Specify the type of operation the scheduler should perform when executing this task. For now only OPERATION="HTTPRequest" is supported for static page generation.

FILE
Required with PUBLISH="Yes." A valid filename for the published file.

PATH
Required with PUBLISH="Yes." The path location for the published file.

STARTDATE
Required when ACTION="Update". The date when scheduling of the task should start.

STARTTIME
Required when creating tasks with ACTION="Update". Enter a value in seconds. The time when scheduling of the task should start.

URL
Required when ACTION="Update". The URL to be executed.

PUBLISH
Optional. Yes or No. Specifies whether the result should be saved to a file.

ENDDATE
Optional. The date when the scheduled task should end.

ENDTIME
Optional. The time when the scheduled task should end. Enter a value in seconds.

INTERVAL
Required when creating tasks with ACTION="Update". Interval at which task should be scheduled. Can be set in seconds or as Once, Daily, Weekly, Monthly, and Execute. The default interval is one hour and the minimum interval is one minute.

REQUESTTIMEOUT
Optional. Customizes the REQUESTTIMEOUT for the task operation. Can be used to extend the default timeout for operations that require more time to execute.

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. Yes or No. Specifies whether to resolve links in the result page to absolute references.

Example

```
<!--- This example shows an example of CFSCHEDULE --->
<HTML>
<HEAD>
<TITLE>CFSCHEDULE Example</TITLE>
</HEAD>

<BODY>
<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 allows you to 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.

<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">

</BODY>
</HTML>
```

CFSCRIPT

The CFSCRIPT tag encloses a code segment containing CFScript.

Syntax

```
<CFSCRIPT>
 CFScript code goes here
</CFSCRIPT>
```

Usage

Use CFSCRIPT to perform processing in CFScript instead of CFML. Note the following regarding CFScript:

- CFScript uses ColdFusion functions, expressions, and operators
- You can read and write ColdFusion variables inside of CFScript

One use of CFSCRIPT is to wrap a series of assignment functions that would otherwise require CFSET statements.

For more information on CFScript, see *Advanced ColdFusion Development*.

Example

```
<!--- This example shows the use of CFSCRIPT --->
<HTML>
<HEAD>
<TITLE>CFSCRIPT Example</TITLE>
</HEAD>

<BODY bgcolor=silver>
<H3>CFSCRIPT Example</H3>

<P>CFSCRIPT adds a simple scripting language to ColdFusion
for those developers who are more comfortable with JavaScript
or VBScript syntax.

<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

Use the CFSEARCH tag to execute searches against data indexed in Verity collections. Collections can be created by calling the CFCOLLECTION tag, by using the ColdFusion Administrator, or through native Verity indexing tools. Collections are populated with data either with the CFINDEX tag, or externally, using native Verity indexing tools. Collections must be created and populated before any searches can be executed.

Syntax

```
<CFSEARCH NAME="search_name"
    COLLECTION="collection_name"
    TYPE="criteria"
    CRITERIA="search_expression"
    MAXROWS="number"
    STARTROW="row_number"
    EXTERNAL="Yes/No"
    LANGUAGE="language">
```

NAME

Required. A name for the search query.

COLLECTION

Required. Specifies the logical collection name that is the target of the search operation or an external collection with fully qualified path. Collection names are defined either through the CFCOLLECTION tag or in the ColdFusion Administrator, Verity page.

Multiple ColdFusion collections can be specified in a comma-separated list:

```
COLLECTION="CFUSER, CFLANG"
```

If you are searching an external collection (EXTERNAL="Yes") specify the collection name, including fully qualified path:

```
COLLECTION="e:\collections\personnel"
```

If multiple collections are specified in COLLECTION and EXTERNAL is Yes, the specified collections must all be externally generated. You cannot combine internal and external collections in the same search operation.

TYPE

Optional. Specifies the criteria type for the search. Valid entries are:

- SIMPLE — By default the STEM and MANY operators are used.
- EXPLICIT — All operators must be invoked explicitly.

CRITERIA

Optional. Specifies the criteria for the search following the syntactic rules specified by TYPE.

MAXROWS

Optional. Specifies the maximum number of entries for index queries. If omitted, all rows are returned.

STARTROW
Optional. Specifies the first row number to be retrieved. Default is 1.

EXTERNAL
Optional. Yes or No. Yes indicates that the collection you are searching was created outside of ColdFusion using native Verity indexing tools. The default is No.

LANGUAGE
Optional. To use the LANGUAGE attribute you must have the ColdFusion International Search Pack installed. Valid entries are:

- English (default)
- German
- Finnish
- French
- Danish
- Dutch
- Italian
- Norwegian
- Portuguese
- Spanish
- Swedish

Usage

In the CRITERIA attribute, if you pass a mixed case entry (mixed upper and lower case), case sensitivity is applied to the search. If you pass all upper or all lower case, case insensitivity is assumed.

Every search conducted with the CFSEARCH tag returns, as part of the record set, a number of result columns you can reference in your CFOUTPUT:

- URL — Returns the value of the URLPATH attribute defined in the CFINDEX tag used to populate the collection. This value is always empty when you populate the collection with CFINDEX when TYPE="Custom".
- KEY — Returns the value of the KEY attribute defined in the CFINDEX tag used to populate the collection.
- TITLE — Returns whatever was placed in the TITLE attribute in the CFINDEX operation used to populate the collection.
- SCORE — Returns the relevancy score of the document based on the search criteria.
- CUSTOM1 and CUSTOM2 — Returns whatever was placed in the custom fields in the CFINDEX operation used to populate the collection.

- SUMMARY — Returns the contents of the automatic summary generated by CFINDEX. The default summarization selects the best three matching sentences, up to a maximum of 500 characters.

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

```
#DocSearch.URL#
#DocSearch.KEY#
#DocSearch.TITLE#
#DocSearch.SCORE#
```

Example

```
<!--- This example shows how to utilize CFSEARCH
to search an existing, populated collection  --->
<HTML>
<HEAD>
<TITLE>
CFSEARCH Example
</TITLE>
</HEAD>

<BODY bgcolor=silver>
<H3>CFSEARCH Example</H3>

<!--- To index the collection, select the check box
on the form --->
<CFIF IsDefined("form.IndexCollection")>
<!--- Change KEY and URLPATH to reflect accurate key and URL
<CFINDEX ACTION="UPDATE" COLLECTION="Snippets"
   KEY="c:\inetpub\wwwroot\cfdocs\snippets" TYPE="PATH"
     TITLE="This is my test" URLPATH="http://127.0.0.1/cfdocs/snippets/"
       EXTENSIONS=".cfm" RECURSE="Yes">
<H3>Collection re-indexed</H3>
</CFIF>
<CFIF IsDefined("form.source") AND
IsDefined("form.type") AND IsDefined("form.searchstring")>

<!--- actually conduct the search --->
    <CFSEARCH NAME="SearchSnippets"
    COLLECTION="#form.source#"
    TYPE="#form.type#"
    CRITERIA="#form.searchstring#">

<!--- print out the search results --->
    <CFOUTPUT>
    <H2>#form.type# Search Results</H2>

    <P>#SearchSnippets.RecordCount# "hit
      <CFIF SearchSnippets.recordcount is not 1>s</CFIF>" found
       out of #SearchSnippets.RecordsSearched# total record
         <CFIF SearchSnippets.recordcount is not 1>s</CFIF>
          searched.

    <P><I><B>#form.maxrows# records returned ...</B></I>
```

```
    <CFTABLE QUERY="SearchSnippets" MAXROWS="#maxrows#"
      STARTROW="1" COLHEADERS HTMLTABLE>
        <CFCOL HEADER="SCORE" TEXT="#score#">
        <CFCOL HEADER="TITLE"
           TEXT="<a href='#url#' target='blank'>#title#</A>">
        <CFCOL HEADER="SUMMARY" TEXT="#summary#">
    </CFTABLE>
    </CFOUTPUT>

</CFIF>
...
```

CFSELECT

Used inside CFFORM, CFSELECT allows you to construct a drop-down list box form control. You can populate the drop-down list box from a query, or using the OPTION tag. Use OPTION elements to populate lists. Syntax for the OPTION tag is the same as for its HTML counterpart.

Syntax

```
<CFSELECT NAME="name"
    REQUIRED="Yes/No"
    MESSAGE="text"
    ONERROR="text"
    SIZE="integer"
    MULTIPLE="Yes/No"
    QUERY="queryname"
    SELECTED="column_value"
    VALUE="text"
    DISPLAY="text">

</CFSELECT>
```

NAME
Required. A name for the form you are creating.

SIZE
Required. Size of the drop-down list box in number of entries.

REQUIRED
Optional. Yes or No. If Yes, a list element must be selected when the form is submitted. Default is No.

MESSAGE
Optional. Message that appears if REQUIRED="Yes" and no selection is made.

ONERROR
Optional. The name of a valid JavaScript function you want to execute in the event of a failed validation.

MULTIPLE
Optional. Yes or No. Yes permits selection of multiple elements in the drop-down list box. The default is No.

QUERY
Optional. Name of the query to be used to populate the drop-down list box.

SELECTED
Optional. Enter a value matching at least one entry in VALUE to preselect the entry in the drop-down list box.

VALUE

Optional. The query column value for the list element. Used with the QUERY attribute.

DISPLAY
Optional. The query column displayed. Defaults to the value of VALUE. Used with the QUERY attribute.

Usage

You can add standard FORM tag attributes and their values to the CFSELECT tag. These attributes and values are passed directly through ColdFusion to the browser in creating a form. For example, FORM tag attributes, like TARGET can be used to enhance your CFFORM features.

CFSELECT supports the JavaScript *onClick* event in the same manner as the HTML INPUT tag:

```
<CFSELECT NAME="dept"
    MESSAGE="You must select a department name"
    QUERY="get_dept_list"
    VALUE="dept_name"
    onClick="JavaScript_function">
```

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 the results of the query.  In addition, 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") 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 lastname
         <CFIF #employeeNames# is not "">= '#employeeNames#'</CFIF>
</CFQUERY>

<HTML>
<HEAD>
<TITLE>
CFSELECT Example
</TITLE>
</HEAD>

<BODY>

<H3>CFSELECT Example</H3>
```

```
<!--- Use CFFORM when using other CFINPUT tools --->
<CFFORM ACTION="cfselect.cfm" METHOD="POST" ENABLECAB="Yes">

<!--- 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>
...
```

CFSET

Use the CFSET tag to define a ColdFusion variable. If the variable already exists, CFSET resets it to the specified value.

Syntax `<CFSET variable_name=expression>`

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>
```

Examples

```
<!--- This example shows how to use CFSET --->
<CFQUERY NAME="GetMessages" DATASOURCE="cfsnippets">
SELECT   *
FROM     Messages
</CFQUERY>
<HTML>
<HEAD>
<TITLE>
CFSET Example
</TITLE>
</HEAD>

<BODY bgcolor=silver>
<H3>CFSET Example</H3>

<P>CFSET allows you to set and reassign values to local or
global variables within a CF template.

<CFSET NumRecords = GetMessages.RecordCount>
<P>For example, the variable NumRecords has been declared on
this template to hold the amount of records returned from
our 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>Finally, CFSET can also be used to collect environmental
variables, such as the time, the IP address of the user, or any
other function or expression possible in ColdFusion.

<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)#")>
...
```

CFSETTING

CFSETTING is used to control various aspects of page processing, such as controlling the output of HTML code in your pages. One benefit of this option is managing whitespace that can occur in output pages that are served by ColdFusion.

Syntax

```
<CFSETTING ENABLECFOUTPUTONLY="Yes/No"
    SHOWDEBUGOUTPUT="Yes/No">
```

ENABLECFOUTPUTONLY
Required. Yes or No. When set to Yes, CFSETTING blocks output of all HTML that resides outside CFOUTPUT tags.

SHOWDEBUGOUTPUT
Optional. Yes or No. When set to No, SHOWDEBUGOUTPUT suppresses debugging information that would otherwise display at the end of the generated page. Default is Yes.

Usage

When nesting CFSETTING tags, you must match each ENABLECFOUTPUTONLY="Yes" setting with an ENABLECFOUTPUTONLY="No" setting for ordinary HTML text to be visible to a user. For example, if you have five ENABLECFOUTPUTONLY="Yes" statements, you must also have five corresponding ENABLECFOUTPUTONLY="No" statements for HTML text to be displayed again.

If at any point the output of plain HTML is enabled (no matter how many ENABLECFOUTPUTONLY="No" statements have been processed) the first ENABLECFOUTPUTONLY="YES" statement will block output.

Example

```
...
<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>

</BODY>
</HTML>
```

CFSLIDER

Used inside CFFORM, CFSLIDER allows you to place a slider control in a ColdFusion form. A slider control is like a sliding volume control. The slider groove is the area over which the slider moves.

Note CFSLIDER incorporates a Java applet, so a browser must be Java-enabled for CFSLIDER to work properly.

Syntax

```
<CFSLIDER NAME="name"
    LABEL="text"
    REFRESHLABEL="Yes/No"
    IMG="filename"
    IMGSTYLE="style"
    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"
    GROOVECOLOR="color"
    BGCOLOR="color"
    TEXTCOLOR="color"
    FONT="font_name"
    FONTSIZE="integer"
    ITALIC="Yes/No"
    BOLD="Yes/No"
    NOTSUPPORTED="text">
```

NAME
Required. A name for the CFSLIDER control.

LABEL
Optional. A label that appears with the slider control, for example:

```
LABEL="Volume %value%"
```

You can use %value% to reference the slider value. If % is omitted, the slider value appears immediately following the label.

REFRESHLABEL
Optional. Yes or No. If Yes, the label is not refreshed when the slider is moved. Default is Yes.

IMG
Optional. Filename of the image to be used in the slider groove.

IMGSTYLE
Optional. Style of the image to appear in the slider groove. Valid entries are:

- Centered
- Tiled
- Scaled

Default is Scaled.

RANGE
Optional. Determines the values of the left and right slider range. The slider value appears as the slider is moved.

Separate values by a comma, for example:

```
RANGE="1,100"
```

Default is "0,100". Valid only for numeric data.

SCALE
Optional. An unsigned integer. SCALE defines the slider scale within the value of RANGE. For example, if RANGE=0,1000 and SCALE=100, the incremental values for the slider would be 0, 100, 200, 300, and so on.

VALUE
Optional. Determines the default slider setting. Must be within the values specified in RANGE. Defaults to the minimum value specified in RANGE.

ONVALIDATE
Optional. The name of a valid JavaScript function used to validate user input, in this case, a change to the default slider value.

MESSAGE
Optional. Message text to appear if validation fails.

ONERROR
Optional. The name of a valid JavaScript function you want to execute in the event of a failed validation.

HEIGHT
Optional. Height value of the slider control, in pixels.

WIDTH
Optional. Width value of the slider control, in pixels.

VSPACE
Optional. Vertical margin spacing above and below slider control, in pixels.

HSPACE
Optional. Horizontal margin spacing to the left and right of slider control, in pixels.

ALIGN
Optional. Alignment value. Valid entries are:

- Top
- Left
- Bottom
- Baseline
- TextTop
- AbsBottom
- Middle
- AbsMiddle
- Right

GROOVECOLOR
Optional. Color value of the slider groove. The slider groove is the area in which the slider box moves. Valid entries are:

- black
- magenta
- cyan
- orange
- darkgray
- pink
- gray
- white
- lightgray
- yellow

A hex value can be entered in the form:

GROOVECOLOR="##*xxxxxx*"
Where *x* is 0–9 or A–F. Use either two pound signs or no pound signs.

BGCOLOR
Optional. Background color of slider label. See GROOVECOLOR for color options.

TEXTCOLOR
Optional. Slider label text color. See GROOVECOLOR for color options.

FONT
Optional. Font name for label text.

FONTSIZE

Optional. Font size for label text measured in points.

ITALIC
Optional. Enter Yes for italicized label text, No for normal text. Default is No.

BOLD
Optional. Enter Yes for bold label text, No for medium text. Default is No.

NOTSUPPORTED
Optional. The text you want to display if the page containing 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>"
```

By default, if no message is specified, the following message appears:

```
<B>Browser must support Java to <BR>
view ColdFusion Java Applets!</B>
```

Example

```
<!--- This example shows how to use CFSLIDER
within CFFORM --->
<HTML>

<HEAD>
<TITLE>
    CFSLIDER Example
</TITLE>
</HEAD>

<BODY bgcolor=silver>

<H3>CFSLIDER Example</H3>
<P>CFSLIDER, used within a CFFORM, can provide
additional 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 CF template.

<P>

<CFIF IsDefined("form.mySlider") is True>
<H3>You slid to a value of <CFOUTPUT>#mySlider#</CFOUTPUT></H3>

Try again!
</CFIF>

<CFFORM ACTION="cfslider.cfm" METHOD="POST" ENABLECAB="Yes">

1 <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="20"
```

```
    WIDTH="150" FONT="Verdana" BGCOLOR="Silver" GROOVECOLOR="Lime"
    BOLD="No" ITALIC="Yes" REFRESHLABEL="Yes"> 100

<P><INPUT TYPE="Submit" NAME="" VALUE="Show the Result">
</CFFORM>

</BODY>
</HTML>
```

CFSTOREDPROC

The CFSTOREDPROC tag is the main tag used for executing stored procedures via an ODBC or native connection to a server database. It specifies database connection information and identifies the stored procedure.

Syntax

```
<CFSTOREDPROC PROCEDURE="procedure name"
    DATASOURCE="ds_name"
    USERNAME="username"
    PASSWORD="password"
    DBSERVER="dbms"
    DBNAME="database name"
    BLOCKFACTOR="blocksize"
    PROVIDER="COMProvider"
    PROVIDERDSN="datasource"
    DEBUG="Yes/No"
    RETURNCODE="Yes/No">
```

PROCEDURE
Required. Specifies the name of the stored procedure on the database server.

DATASOURCE
Required. The name of an ODBC or native data source that points to the database containing the stored procedure.

USERNAME
Optional. If specified, USERNAME overrides the username value specified in the data source setup.

PASSWORD
Optional. If specified, PASSWORD overrides the password value specified in the data source setup.

DBSERVER
Optional. For native database drivers, specifies the name of the database server machine. If specified, DBSERVER overrides the server specified in the data source.

DBNAME
Optional. The database name (Sybase System 11 driver only). If specified, DBNAME overrides the default database specified in the data source.

BLOCKFACTOR
Optional. Specifies the maximum number of rows to fetch at a time from the server. The range is 1 (default) to 100. The ODBC driver may dynamically reduce the block factor at runtime.

PROVIDER
Optional. COM provider (OLE-DB only).

PROVIDERDSN
Optional. Data source name for the COM provider (OLE-DB only).

DEBUG
Optional. Yes or No. Specifies whether debug info will be listed on each statement. Default is No.

RETURNCODE
Optional. Yes or No. Specifies whether the tag populates CFSTOREDPROC.STATUSCODE with the status code returned by the stored procedure. Default is No.

Usage

Within a CFSTOREDPROC tag, you code CFPROCRESULT and CFPROCPARAM tags as necessary.

If you set the ReturnCode parameter to "YES", CFSTOREDPROC sets a variable called CFSTOREDPROC.STATUSCODE, which indicates the status code for the stored procedure. Stored procedure status code values vary by DBMS. Refer to your DBMS-specific documentation for the meaning of individual status code values.

Stored procedures represent an advanced feature, found in high-end database management systems. You should be familiar with stored procedures and their usage before implementing these tags.

Example

```
...
<!--- The following 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 tag --->
<CFSTOREDPROC PROCEDURE="foo_proc"
    DATASOURCE="MY_SYBASE_TEST"USERNAME="sa"
    PASSWORD=""DBSERVER="scup"DBNAME="pubs2"
    RETURNCODE="YES"DEBUG>
<!--- 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>
...
```

CFSWITCH/CFCASE/CFDEFAULTCASE

Used with CFCASE and CFDEFAULTCASE, the CFSWITCH tag 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.

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>
```

EXPRESSION
Required. Any 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 specified expression (case-insensitive comparison). If a value matches the expression, CFSWITCH executes the code between the CFCASE start and end tags.

Separate multiple values with a comma or an alternative delimiter, as specified in the DELIMITERS parameter. Duplicate value attributes are not allowed and will cause a runtime error.

DELIMITERS
Optional. Specifies the character that separates multiple entries in a list of values. The default delimiter is the comma (,).

Usage

Use CFSWITCH followed by one or more CFCASE tags, optionally ending with a CFDEFAULTCASE tag. The CFSWITCH tag selects the matching alternative from the specified CFCASE and CFDEFAULTCASE tags and jumps to the matching tag, executing the code between the CFCASE start and end tags. There is no need to explicitly break out of the CFCASE tag, as there is in some other languages.

You can specify only one CFDEFAULTCASE tag within a CFSWITCH tag. CFCASE tags cannot appear after the CFDEFAULTCASE tag.

CFSWITCH provides better performance than a series of CFIF/CFELSEIF tags and the resulting code is easier to read.

Example

```
<!--- This example illustrates the use of CFSWITCH and
CFCASE to exercise a case statement in CFML --->
```

```
<!--- query to get some information --->
<CFQUERY NAME="GetEmployees" DATASOURCE="cfsnippets">
SELECT   Emp_ID, FirstName, LastName, EMail,
         Phone, Department
FROM     Employees
</CFQUERY>

<HTML>
<HEAD>
<TITLE>
CFSWITCH Example
</TITLE>
</HEAD>

<BODY bgcolor=silver>
<H3>CFSWITCH Example</H3>

<!--- By outputting the query and using CFSWITCH,
we can classify the output without using a CFLOOP construct.
 --->
<CFOUTPUT query="GetEmployees">
<CFSWITCH EXPRESSION=#Department#>
<!--- each time the case is fulfilled, the specific
information is printed; if the case is not fulfilled,
the default case is output --->
    <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>
    </CFDEFAULTCASE>
</CFSWITCH>
</CFOUTPUT>

</BODY>
</HTML>
```

CFTABLE

Builds a table in your ColdFusion page. Use the CFCOL tag to define column and row characteristics for a table. CFTABLE renders data either as preformatted text, or, with the HTMLTABLE attribute, as an HTML table. Use CFTABLE to create tables if you don't want to write your own HTML TABLE tag code, or if your data can be well presented as preformatted text.

Syntax

```
<CFTABLE QUERY="query_name"
    MAXROWS="maxrows_table"
    COLSPACING="number_of_spaces"
    HEADERLINES="number_of_lines"
    HTMLTABLE
    BORDER
    COLHEADERS
    STARTROW="row_number">

</CFTABLE>
```

QUERY
Required. The name of the CFQUERY from which you want to draw data.

MAXROWS
Optional. Specifies the maximum number of rows you want to display in the table.

COLSPACING
Optional. Indicates the number of spaces to insert between columns (default is 2).

HEADERLINES
Optional. Indicates the number of lines to use for the table header (the default is 2, which leaves one line between the headers and the first row of the table).

HTMLTABLE
Optional. Renders the table as an HTML 3.0 table.

BORDER
Optional. Adds a border to the table. Use only when you specify the HTMLTABLE attribute for the table.

COLHEADERS
Optional. Displays headers for each column, as specified in the CFCOL tag.

STARTROW
Optional. Specifies the query row from which to start processing.

Example

```
<!--- This example shows the use of CFCOL and CFTABLE
to align information returned from a query --->

<!--- This query selects employee information from the
```

```
cfsnippets datasource --->
<CFQUERY NAME="GetEmployees" DATASOURCE="cfsnippets">
SELECT Emp_ID, FirstName, LastName, EMail, Phone, Department
FROM Employees
</CFQUERY>

<HTML>
<HEAD>
<TITLE>CFTABLE Example</TITLE>
</HEAD>

<BODY>
<H3>CFTABLE Example</H3>

<!--- Note the use of the HTMLTABLE attribute to display the
CFTABLE as an HTML table, rather simply as PRE formatted information --->
<CFTABLE QUERY="GetEmployees" STARTROW="1" COLSPACING="3" HTMLTABLE>
<!--- each CFCOL tag sets the width of a column in the table,
as well as specifying the header information and the text/CFML
with which to fill 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>

</BODY>
</HTML>
```

CFTEXTINPUT

The CFTEXTINPUT form custom control allows you to place a single-line text entry box in a CFFORM. In addition to input validation, the tag gives you control over all font characteristics.

Note CFTEXTINPUT incorporates a Java applet, so a browser must be Java-enabled for CFTEXTINPUT to work properly.

Syntax

```
<CFTEXTINPUT NAME="name"
    VALUE="text"
    REQUIRED="Yes/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/No"
    BOLD="Yes/No"
    HEIGHT="integer"
    WIDTH="integer"
    VSPACE="integer"
    HSPACE="integer"
    ALIGN="alignment"
    BGCOLOR="color"
    TEXTCOLOR="color"
    MAXLENGTH="integer"
    NOTSUPPORTED="text">
```

NAME
Required. A name for the CFTEXTINPUT control.

VALUE
Optional. Initial value that appears in the text control.

REQUIRED
Optional. Yes or No. If Yes, the user must enter or change text. Default is No.

RANGE
Optional. Enter a minimum value, maximum value range separated by a comma. Valid only for numeric data.

VALIDATE
Optional. Valid entries are:

- date — Verifies US date entry in the form *mm/dd/yy*.

- eurodate — Verifies valid European date entry in the form *dd/mm/yyyy*.
- time — Verifies a time entry in the form *hh:mm:ss*.
- float — Verifies a floating point entry.
- integer — Verifies an integer entry.
- telephone — Verifies a telephone entry. Telephone data must be entered as ###-###-####. The hyphen separator (-) can be replaced with a blank. The area code and exchange must begin with a digit between 1 and 9.
- zipcode — (U.S. formats only) Number can be a 5-digit or 9-digit zip in the form #####-####. The hyphen separator (-) can be replaced with a blank.
- creditcard — Blanks and dashes are stripped and the number is verified using the mod10 algorithm.
- social_security_number — Number must be entered as ###-##-####. The hyphen separator (-) can be replaced with a blank.

ONVALIDATE
Optional. The name of a valid JavaScript function used 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 and FALSE otherwise. When used, the VALIDATE attribute is ignored.

MESSAGE
Optional. Message text to appear if validation fails.

ONERROR
Optional. The name of a valid JavaScript function you want to execute in the event of a failed validation.

SIZE
Optional. Number of characters displayed before horizontal scroll bar appears.

FONT
Optional. Font name for text.

FONTSIZE
Optional. Font size for text.

ITALIC
Optional. Enter Yes for italicized text, No for normal text. Default is No.

BOLD
Optional. Enter Yes for boldface text, No for medium text. Default is No.

HEIGHT
Optional. Height value of the control, in pixels.

WIDTH

Optional. Width value of the control, in pixels.

VSPACE
Optional. Vertical spacing of the control, in pixels.

HSPACE
Optional. Horizontal spacing of the control, in pixels.

ALIGN
Optional. Alignment value. Valid entries are:

- Top
- Left
- Bottom
- Baseline
- TextTop
- AbsBottom
- Middle
- AbsMiddle
- Right

BGCOLOR
Optional. Background color of the control. Valid entries are:

- black
- magenta
- cyan
- orange
- darkgray
- pink
- gray
- white
- lightgray
- yellow

A hex value can also be entered in the form:

`BGCOLOR="##xxxxxx"`
Where *x* is 0–9 or A–F. Use either two pound signs or no pound signs.

TEXTCOLOR
Optional. Text color for the control. See BGCOLOR for color options.

MAXLENGTH
Optional. The maximum length of text entered.

NOTSUPPORTED
Optional. The text you want to display if the page containing 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>"
```

By default, if no message is specified, the following message appears:

```
<B>Browser must support Java to <BR>
view ColdFusion Java Applets!</B>
```

Example

```
<!--- This example shows the use of CFTEXTINPUT --->

<HTML>
<HEAD>
<TITLE>
CFTEXTINPUT Example
</TITLE>
</HEAD>

<BODY bgcolor=silver>

<H3>CFTEXTINPUT Example</H3>

CFTEXTINPUT can be used to provide simple validation for text
fields in CFFORM and to have control over font information
displayed in CFFORM input boxes for text.  For example, the field
provided below must not
be blank, and provides a client-side message upon erring.

<CFFORM ACTION="cftextinput.cfm" METHOD="POST" ENABLECAB="Yes">

<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>

</BODY>
</HTML>
```

CFTHROW

The CFTHROW tag raises a developer-specified exception that can be caught with CFCATCH TYPE="APPLICATION" or CFCATCH TYPE="ANY".

Syntax

```
<CFTHROW MESSAGE="message">
```

MESSAGE
Optional. A message that describes the exceptional event.

Usage

Use CFTHROW within a CFTRY block to raise an error condition. The CFCATCH block can access this message through CFCATCH.message.

Example

```
<!--- This example shows how to use CFTHROW to
create an error --->
<HTML>
<HEAD>
<TITLE>
CFTHROW Example
</TITLE>
</HEAD>

<BODY bgcolor=silver>
<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 ColdFusion Server --->
    <P>#CFCATCH.message#
</CFOUTPUT>
</CFCATCH>
</CFTRY>

</BODY>
</HTML>
```

CFTRANSACTION

Use CFTRANSACTION to group multiple queries into a single unit. CFTRANSACTION can also provide rollback processing.

Syntax

```
<CFTRANSACTION ISOLATION="ODBC_lock">
...
</CFTRANSACTION>
```

ISOLATION
Optional. ODBC lock type. Valid entries are:

- Read_Uncommitted
- Read_Committed
- Repeatable_Read
- Serializable

Usage

Any queries executed with CFQUERY and placed between <CFTRANSACTION> and </CFTRANSACTION> tags are treated as a single transaction. Changes to data requested by these queries are not committed to the database until all actions within the transaction block have executed successfully. If an error occurs in a query, all changes made by previous queries within the transaction block are rolled back.

Use the ISOLATION attribute for additional control over how the database engine performs locking during the transaction.

Example

```
<!--- This example shows the use of CFTRANSACTION --->
<HTML>
<HEAD>
<TITLE>CFTRANSACTION Example</TITLE>
</HEAD>

<BODY>
<H3>CFTRANSACTION Example</H3>

<P>CFTRANSACTION can be used to group multiple queries
using CFQUERY into a single business event.  Changes to data
requested by these queries are not committed to the datasource
until all actions within the transaction block have executed
successfully.
<P>The following is a sample listing (see code in right pane):

<CFTRANSACTION>
    <CFQUERY NAME='makeNewCourse' DATASOURCE='cfsnippets'>
    INSERT INTO Courses
        (Number, Descript)
    VALUES
        ('#myNumber#', '#myDescription#')
    </CFQUERY>
```

```
    <CFQUERY NAME='insertNewCourseToList' DATASOURCE='cfsnippets'>
    INSERT INTO CourseList
        (CorNumber, CorDesc, Dept_ID,
        CorName, CorLevel, LastUpdate)
    VALUES
        ('#myNumber#', '#myDescription#', '#myDepartment#',
        '#myDescription#', '#myCorLevel#', #Now()#)
    </CFQUERY>
</CFTRANSACTION>

</BODY>
</HTML>
```

CFTREE

The CFTREE form custom control allows you to place a tree control in a CFFORM. User selections can be validated. Individual tree items are created with CFTREEITEM tags inside the CFTREE tag block.

Note CFTREE incorporates a Java applet, so a browser must be Java-enabled for CFTREE to work properly.

Syntax

```
<CFTREE NAME="name"
    REQUIRED="Yes/No"
    DELIMITER="delimiter"
    COMPLETEPATH="Yes/No"
    APPENDKEY="Yes/No"
    HIGHLIGHTHREF="Yes/No"
    ONVALIDATE="script_name"
    MESSAGE="text"
    ONERROR="text"
    FONT="font"
    FONTSIZE="size"
    ITALIC="Yes/No"
    BOLD="Yes/No"
    HEIGHT="integer"
    WIDTH="integer"
    VSPACE="integer"
    HSPACE="integer"
    ALIGN="alignment"
    BORDER="Yes/No"
    HSCROLL="Yes/No"
    VSCROLL="Yes/No"
    NOTSUPPORTED="text">

</CFTREE>
```

NAME
Required. A name for the CFTREE control.

REQUIRED
Optional. Yes or No. User must select an item in the tree control. Default is No.

DELIMITER
Optional. The character used to separate elements in the form variable PATH. The default is "\".

COMPLETEPATH
Optional. Yes passes the root level of the treename.path form variable when the CFTREE is submitted. If omitted or No, the root level of this form variable is not included.

APPENDKEY
Optional. Yes or No. When used with HREF, Yes passes the CFTREEITEMKEY variable along with the value of the selected tree item in the URL to the application page specified in the CFFORM ACTION attribute. The default is Yes.

HIGHLIGHTHREF
Optional. Yes highlights links associated with a CFTREEITEM with a URL attribute value. No disables highlight. Default is Yes.

ONVALIDATE
Optional. The name of a valid JavaScript function used 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 and false otherwise.

MESSAGE
Optional. Message text to appear if validation fails.

ONERROR
Optional. The name of a valid JavaScript function you want to execute in the event of a failed validation.

FONT
Optional. Font name to use for all data in the tree control.

FONTSIZE
Optional. Font size for text in the tree control, measured in points.

ITALIC
Optional. Yes or No. Yes presents all tree control text in italic. Default is No.

BOLD
Optional. Yes or No. Yes presents all tree control text in boldface. Default is No.

HEIGHT
Optional. Height value of the tree control, in pixels.

WIDTH
Optional. Width value of the tree control, in pixels.

VSPACE
Optional. Vertical margin spacing above and below the tree control in pixels.

HSPACE
Optional. Horizontal spacing to the left and right of the tree control, in pixels.

ALIGN
Optional. Alignment value. Valid entries are:

- Top

- Left
- Bottom
- Baseline
- TextTop
- AbsBottom
- Middle
- AbsMiddle
- Right

BORDER
Optional. Places a border around the tree. Default is Yes.

HSCROLL
Optional. Permits horizontal scrolling. Default is Yes.

VSCROLL
Optional. Permits vertical scrolling. Default is Yes.

NOTSUPPORTED
Optional. The text you want to display if the page containing 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>"
```

By default, if no message is specified, the following message appears:

```
<B>Browser must support Java to <BR>
view ColdFusion Java Applets!</B>
```

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 of the query.  In addition, 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>
<HEAD>
```

```
<TITLE>
CFTREE Example
</TITLE>
</HEAD>

<BODY>
<H3>CFTREE Example</H3>

<!--- Use CFFORM when using other CFINPUT tools --->
<CFFORM ACTION="cftree.cfm" METHOD="POST" ENABLECAB="Yes">
<!--- 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>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

Use CFTREEITEM to populate a tree control created with CFTREE with individual elements. You can use the IMG values supplied with ColdFusion or reference your own icons.

Note CFTREEITEM incorporates a Java applet, so a browser must be Java-enabled for CFTREE to work properly.

Syntax

```
<CFTREEITEM VALUE="text"
    DISPLAY="text"
    PARENT="parent_name"
    IMG="filename"
    IMGOPEN="filename"
    HREF="URL"
    TARGET="URL_target"
    QUERY="queryname"
    QUERYASROOT="Yes/No"
    EXPAND="Yes/No">
```

VALUE
Required. Value passed when the CFFORM is submitted. When populating a CFTREE with data from a CFQUERY, columns are specified in a comma-separated list:

```
VALUE="dept_id,emp_id"
```

DISPLAY
Optional. The label for the tree item. Default is VALUE. When populating a CFTREE with data from a CFQUERY, display names are specified in a comma-separated list:

```
DISPLAY="dept_name,emp_name"
```

PARENT
Optional. Value for tree item parent.

IMG
Optional. Image name or filename for the tree item. When populating a CFTREE with data from a CFQUERY, images or filenames for each level of the tree are specified in a comma-separated list.

The default image name is “Folder.” A number of images are supplied and can be specified using only the image name (no file extension):

- cd
- computer
- document
- element

- folder
- floppy
- fixed
- remote

Use commas to separate image names corresponding to tree level, for example:

```
IMG="folder,document"
IMG=",document
```

To specify your own custom image, specify the path and file extension:

```
IMG="../images/page1.gif"
```

IMGOPEN
Optional. Icon displayed with open tree item. You can specify the icon filename using a relative path. As with IMG, you can use an image supplied with ColdFusion.

HREF
Optional. URL to associate with the tree item or a query column for a tree that is populated from a query. If HREF is a query column, then the HREF value is the value populated by the query. If HREF is not recognized as a query column, it is assumed that the HREF text is an actual HTML HREF.

When populating a CFTREE with data from a CFQUERY, HREFs can be specified in a comma-separated list

```
HREF="http://dept_server,http://emp_server"
```

TARGET
Optional. Target attribute for HREF URL. When populating a CFTREE with data from a CFQUERY, targets are specified in a comma-separated list:

```
TARGET="FRAME_BODY,_blank"
```

QUERY
Optional. Query name used to generate data for the tree item.

QUERYASROOT
Optional. Yes or No. Defines specified query as the root level. As in Example 1, this option prevents having to create an additional parent CFTREEITEM.

EXPAND
Optional. Yes or No. Yes expands tree to show tree item children. No keeps tree item collapsed. Default is Yes.

Examples

```
<!--- This example shows the use of CFTREEITEM in a CFFORM.
The query takes a list of employees, and uses CFTREE and CFSELECT
to display the results of the query.  In addition, 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>
<HEAD>
<TITLE>
CFTREE Example
</TITLE>
</HEAD>

<BODY>
<H3>CFTREEITEM Example</H3>

<!--- Use CFFORM when using other CFINPUT tools --->
<CFFORM ACTION="cftreeitem.cfm" METHOD="POST" ENABLECAB="Yes">

<!--- 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>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>
...
```

CFTRY/CFCATCH

Used with one or more CFCATCH tags, the CFTRY tag allows developers to catch and process exceptions in ColdFusion pages. Exceptions include any event that disrupts the normal flow of instructions in a ColdFusion page such as failed database operations, missing include files, and developer-specified events.

Syntax

```
<CFTRY>
... Add code here
<CFCATCH TYPE="exceptiontype">
... Add exception processing code here
</CFCATCH>
... Additional CFCATCH blocks go here
</CFTRY>
```

TYPE

Optional. Specifies the type of exception to be handled by the CFCATCH block:

- Application
- Database
- Template
- Security
- Object
- Synchronization
- MissingInclude
- Expression
- Lock
- Any (default)

Usage

You must code at least one CFCATCH tag within a CFTRY block. Code CFCATCH tags at the end of the CFTRY block. Always code CFCATCH TYPE="ANY" last; ColdFusion tests CFCATCH tags in the order in which they appear in the page.

Applications can optionally use the CFTHROW tag to raise custom exceptions. Such exceptions are caught with either TYPE="APPLICATION" or TYPE="ANY".

The tags that throw an exception of TYPE="TEMPLATE" are CFINCLUDE, CFMODULE, and CFERROR.

An exception raised within a CFCATCH block cannot be handled by the CFTRY block that immediately encloses the CFCATCH tag.

You can use the CFCATCH variable to access exception information:

- Type — Exception type, as specified in CFCATCH
- Message — The exception's diagnostic message, if one was provided. If no diagnostic message is available, this is an empty string.

- Detail — A detailed message from the CFML interpreter. This message, which contains HTML formatting, can help determine which tag threw the exception.
- NativeErrorCode — TYPE=Database only. The native error code associated with this exception. Database drivers typically provide error codes to assist diagnosis of failing database operations. If no error code was provided, the value of NativeErrorCode is -1.
- SQLState — TYPE=Database only. The SQLState associated with this exception. Database drivers typically provide error codes to assist diagnosis of failing database operations. If no SQLState value was provided, the value of SQLState is -1.
- ErrNumber — TYPE=Expression only. Internal expression error number.
- MissingFileName — TYPE=MissingInclude only. Name of the file that could not be included.
- LockName — TYPE=Lock only. The name of the affected lock (set to anonymous if the lock was unnamed).
- LockOperation — TYPE=Lock only. The operation that failed (set to Timeout, Create MuteEx, or Unknown).

Example

```
<!--- CFTRY example, using CFTHROW to create
a sample error --->
<HTML>
<HEAD>
<TITLE>
CFTRY Example
</TITLE>
</HEAD>

<BODY bgcolor=silver>
<H3>CFTRY Example</H3>

<!--- open a CFTRY block --->
<CFTRY>

<!--- note that we have misspelled the 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 are fishing --->
<CFCATCH TYPE="Database">
<!--- the message to display --->
    <H3>You've Thrown a Database <B>Error</B></H3>

<CFOUTPUT>
<!--- and the diagnostic message from ColdFusion Server --->
```

```
    <P>#CFCATCH.message#
</CFOUTPUT>
</CFCATCH>
</CFTRY>

</BODY>
</HTML>
```

CFUPDATE

The CFUPDATE tag updates existing records in data sources.

Syntax

```
<CFUPDATE DATASOURCE="ds_name"
    DBTYPE="type"
    DBSERVER="dbms"
    DBNAME="database name"
    TABLENAME="table_name"
    TABLEOWNER="name"
    TABLEQUALIFIER="qualifier"
    USERNAME="username"
    PASSWORD="password"
    PROVIDER="COMProvider"
    PROVIDERDSN="datasource"
    FORMFIELDS="field_names">
```

DATASOURCE
Required. Name of the data source that contains your table.

DBTYPE
Optional. The database driver type:

- ODBC (default) — ODBC driver.
- Oracle73 — Oracle 7.3 native database driver. Using this option, the ColdFusion Server computer must have Oracle 7.3.3 (or greater) client software installed.
- Oracle80 —Oracle 8.0 native database driver. Using this option, the ColdFusion Server computer must have Oracle 8.0 (or greater) client software installed.
- Sybase11 —Sybase System 11 native database driver. Using this option, the ColdFusion Server computer must have Sybase 11.1.1 (or greater) client software installed. Sybase patch ebf 7729 is recommended.

DBSERVER
Optional. For native database drivers, specifies the name of the database server machine. If specified, DBSERVER overrides the server specified in the data source.

DBNAME
Optional. The database name (Sybase System 11 driver only). If specified, DBNAME overrides the default database specified in the data source.

TABLENAME
Required. Name of the table you want to update. Note the following:

- ORACLE drivers — This specification must be in uppercase.
- Sybase driver — This specification is case-sensitive and must be in the same case as that used when the table was created

TABLEOWNER

Optional. For data sources that support table ownership (for example, 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 the qualifier for the table. The purpose of table qualifiers varies across drivers. For SQL Server and Oracle, the qualifier refers to the name of the database that contains the table. For the Intersolv dBase driver, the qualifier refers to the directory where the DBF files are located.

USERNAME
Optional. If specified, USERNAME overrides the username value specified in the ODBC setup.

PASSWORD
Optional. If specified, PASSWORD overrides the password value specified in the ODBC setup.

PROVIDER
Optional. COM provider (OLE-DB only).

PROVIDERDSN
Optional. Data source name for the COM provider (OLE-DB only).

FORMFIELDS
Optional. A comma-separated list of form fields to update. If this attribute is not specified, all fields in the form are included in the operation.

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") is "True">
<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     Number, Course_ID, Descript
FROM       Courses
<CFIF IsDefined("url.course_ID") is True>
WHERE    Course_ID = #url.course_ID#
</CFIF>
ORDER by Number
</CFQUERY>
```

```
<HTML>
<HEAD>
<TITLE>CFUPDATE Example</TITLE>
</HEAD>

<BODY bgcolor=silver>
<H3>CFUPDATE Example</H3>

<!--- if we are updating a record, don't show
the entire list --->
<CFIF NOT IsDefined("url.course_ID")>
<P><H3><a href="cfupdate.cfm">Show Entire List</A></H3>

<FORM METHOD="POST" ACTION="cfupdate.cfm">

<H3>You can alter the contents of this
record, and then click "submit" to use
CFUPDATE and alter the database</H3>

<P>Course Number <INPUT TYPE="Text" NAME="number"
  VALUE="<CFOUTPUT>#GetCourseInfo.number#</CFOUTPUT>">
<P>Course Description<BR>
<TEXTAREA NAME="Descript" COLS="40" ROWS="5">
<CFOUTPUT>#GetCourseInfo.Descript#</CFOUTPUT>
</TEXTAREA>
<INPUT TYPE="Hidden" NAME="course_id"
  VALUE="<CFOUTPUT>#GetCourseInfo.Course_ID#</CFOUTPUT>">
<P><INPUT TYPE="Submit" NAME="">
</FORM>

<CFELSE>
<!--- Show the entire record set in CFTABLE form --->
<CFTABLE QUERY="GetCourseInfo" HTMLTABLE>
<CFCOL  TEXT="<a href='cfupdate.cfm?course_ID=#course_ID#'>Edit Me</a>"
  WIDTH=10 HEADER="Edit<br>this Entry">
<CFCOL  TEXT="#Number#" WIDTH="4" HEADER="Course Number">
<CFCOL  TEXT="#Descript#" WIDTH=100 HEADER="Course Description">
</CFTABLE>
</CFIF>

</BODY>
</HTML>
```

CFWDDX

The CFWDDX tag serializes and de-serializes CFML data structures to the XML-based WDDX format. You can also use it to generate JavaScript statements instantiating JavaScript objects equivalent to the contents of a WDDX packet or some CFML data structures.

Syntax

```
<CFWDDX ACTION="action"
    INPUT="inputdata"
    OUTPUT="resultvariablename"
    TOPLEVELVARIABLE="toplevelvariablenameforjavascript">
```

ACTION
Specifies the action taken by the CFWDDX tag. Use one of the following:

- CFML2WDDX — Serialize CFML to WDDX format
- WDDX2CFML — Deserialize WDDX to CFML
- CFML2JS — Serialize CFML to JavaScript format
- WDDX2JS — Deserialize WDDX to JavaScript

INPUT
Required. The value to be processed.

OUTPUT
The name of the variable to hold the output of the operation. This attribute is required for ACTION=WDDX2CFML. For all other actions, if this attribute is not provided, the result of the WDDX processing is outputted in the HTML stream.

TOPLEVELVARIABLE
Required when ACTION=WDDX2JS or ACTION=CFML2JS. The name of the top-level JavaScript object created by the deserialization process. The object created by this process is an instance of the WddxRecordset object, explained in "WddxRecordset Object" on page 449.

This is optional when ACTION=CFML2WDDX or ACTION=WDDX2CFML.

Usage

Use this tag to serialize and deserialize packets of data used to communicate with the browser.

For complete information on WDDX, see the "Programming with XML" chapter in *Advanced ColdFusion Development*.

Example

```
<!--- This snippet shows basic use of the CFWDDX tag. --->
<html>
<head>
    <title>CFWDDX Tag</title>
</head>
<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>
</body>
</html>
```

CHAPTER 3

ColdFusion Functions

This chapter describes each of the functions in the ColdFusion Markup Language (CFML). The introduction contains an alphabetical summary of ColdFusion functions, a list of new functions in ColdFusion 4.0, and a list of functions by category. The remainder of this chapter provides complete descriptions of each function, listed alphabetically.

Contents

Alphabetical List of ColdFusion Functions

Abs
ArrayAppend
ArrayAvg
ArrayClear
ArrayDeleteAt
ArrayInsertAt
ArrayIsEmpty
ArrayLen
ArrayMax
ArrayMin
ArrayNew
ArrayPrepend
ArrayResize
ArraySet
ArraySort
ArraySum
ArraySwap
ArrayToList
Asc
Atn
BitAnd
BitMaskClear
BitMaskRead
BitMaskSet
BitNot
BitOr
DaysInMonth
DaysInYear
DE
DecimalFormat
DecrementValue
Decrypt
DeleteClientVariable
DirectoryExists
DollarFormat
Encrypt
Evaluate
Exp
ExpandPath
FileExists
Find
FindNoCase
FindOneOf
FirstDayOfMonth
Fix
FormatBaseN
GetBaseTagData
GetBaseTagList
GetClientVariablesList
GetDirectoryFromPath
GetFileFromPath
GetLocale
IsStruct
LCase
Left
Len
ListAppend
ListChangeDelims
ListContains
ListContainsNoCase
ListDeleteAt
ListFind
ListFindNoCase
ListFirst
ListGetAt
ListInsertAt
ListLast
ListLen
ListPrepend
ListRest
ListSetAt
ListToArray
LJustify
Log
Log10
LSCurrencyFormat
LSDateFormat
LSIsCurrency
QuerySetCell
QuotedValueList
Rand
Randomize
RandRange
REFind
REFindNoCase
RemoveChars
RepeatString
Replace
ReplaceList
ReplaceNoCase
REReplace
REReplaceNoCase
Reverse
Right
RJustify
Round
RTrim
Second
SetLocale
SetVariable
Sgn
Sin
SpanExcluding
SpanIncluding

BitSHLN
BitSHRN
BitXor
Ceiling
Chr
CJustify
Compare
CompareNoCase
Cos
CreateDate
CreateDateTime
CreateODBCDate
CreateODBCDateTime
CreateODBCTime
CreateTime
CreateTimeSpan
DateAdd
DateCompare
DateDiff
DateFormat
DatePart
Day
DayOfWeek
DayOfWeekAsString
DayOfYear
GetTempDirectory
GetTempFile
GetTemplatePath
GetTickCount
GetToken
Hour
HTMLCodeFormat
HTMLEditFormat
IIf
IncrementValue
InputBaseN
Insert
Int
IsArray
IsAuthenticated
IsAuthorized
IsBoolean
IsDate
IsDebugMode
IsDefined
IsLeapYear
IsNumeric
IsNumericDate
IsQuery
IsSimpleValue
LSIsDate
LSIsNumeric
LSNumberFormat
LSParseCurrency
LSParseDateTime
LSParseNumber
LSTimeFormat
LTrim
Max
Mid
Min
Minute
Month
MonthAsString
Now
NumberFormat
ParagraphFormat
ParameterExists
ParseDateTime
Pi
PreserveSingleQuotes
Quarter
QueryAddRow
QueryNew
Sqr
StripCR
StructClear
StructCopy
StructCount
StructDelete
StructFind
StructInsert
StructIsEmpty
StructKeyExists
StructNew
StructUpdate
Tan
TimeFormat
Trim
UCase
URLEncodedFormat
Val
ValueList
Week
WriteOutput
Year
YesNoFormat

New Functions in ColdFusion 4.0

Decrypt
DirectoryExists
Encrypt
GetBaseTagData
GetBaseTagList
GetTickCount
IsAuthenticated
IsAuthorized
IsDebugMode
IsStruct
REFindNoCase
REReplaceNoCase
StructClear
StructCount
StructCopy
StructDelete
StructFind
StructInsert
StructIsEmpty
StructKeyExists
StructNew
WriteOutput

Array Functions

ArrayAppend
ArrayAvg
ArrayClear
ArrayDeleteAt
ArrayInsertAt
ArrayIsEmpty
ArrayLen
ArrayMax
ArrayMin
ArrayNew
ArrayPrepend
ArrayResize
ArraySet
ArraySort
ArraySum
ArraySwap
ArrayToList
IsArray
ListToArray

Date and Time Functions

CreateDate
CreateDateTime
CreateODBCDate
Day
DayOfWeek
DayOfWeekAsString
IsNumericDate
Minute
Month

CreateODBCDateTime
CreateODBCTime
CreateTime
CreateTimeSpan
DateAdd
DateCompare
DateDiff
DatePart
DayOfYear
DaysInMonth
DaysInYear
FirstDayOfMonth
Hour
IsDate
IsLeapYear
MonthAsString
Now
ParseDateTime
Quarter
Second
Week
Year

Decision Functions

IsArray
IsAuthenticated
IsAuthorized
IsBoolean
IsDate
IsDebugMode
IsDefined
IsLeapYear
IsNumeric
IsNumericDate
IsQuery
IsSimpleValue
LSIsCurrency
LSIsDate
LSIsNumeric
ParameterExists

Display and Formatting Functions

DateFormat
DecimalFormat
DollarFormat
FormatBaseN
LSDateFormat
LSNumberFormat
LSTimeFormat
NumberFormat

HTMLCodeFormat	ParagraphFormat
HTMLEditFormat	TimeFormat
LSCurrencyFormat	YesNoFormat

Dynamic Evaluation Functions

DE	IIf
Evaluate	SetVariable

List Functions

ArrayToList	ListGetAt
ListAppend	ListInsertAt
ListChangeDelims	ListLast
ListContains	ListLen
ListContainsNoCase	ListPrepend
ListDeleteAt	ListRest
ListFind	ListSetAt
ListFindNoCase	ListToArray
ListFirst	

Structure Functions

IsStruct	StructInsert
StructClear	StructIsEmpty

StructCopy	StructKeyExists
StructCount	StructNew
StructDelete	StructUpdate
StructFind	

International Functions

GetLocale	LSNumberFormat
LSCurrencyFormat	LSParseCurrency
LSDateFormat	LSParseDateTime
LSIsCurrency	LSParseNumber
LSIsDate	LSTimeFormat
LSIsNumeric	SetLocale

Mathematical Functions

Abs	Ceiling	Min
Atn	Cos	Pi
BitAnd	DecrementValue	Rand
BitMaskClear	Exp	Randomize
BitMaskRead	Fix	RandRange
BitMaskSet	IncrementValue	Round
BitNot	InputBaseN	Sgn
BitOr	Int	Sin
BitSHLN	Log	Sqr

BitSHRN
BitXor
Log10
Max
Tan

String Functions

Asc
Chr
CJustify
Compare
CompareNoCase
DayOfWeekAsString
FormatBaseN
Find
FindNoCase
FindOneOf
GetToken
Insert
LCase
Left
Len
LJustify
LSParseCurrency
LSParseDateTime
LSParseNumber
LTrim
Mid
MonthAsString
ParseDateTime
REFind
REFindNoCase
RemoveChars
RepeatString
Replace
ReplaceList
ReplaceNoCase
REReplace
REReplaceNoCase
Reverse
Right
RJustify
RTrim
SpanExcluding
SpanIncluding
Trim
UCase
Val

System Functions

DirectoryExists
ExpandPath
FileExists
GetDirectoryFromPath
GetFileFromPath
GetTempDirectory
GetTempFile
GetTemplatePath

Query Functions

IsQuery	QueryNew
QueryAddRow	QuerySetCell

Other Functions

Decrypt	ParameterExists
DeleteClientVariable	PreserveSingleQuotes
Encrypt	QuotedValueList
GetBaseTagData	StripCR
GetBaseTagList	URLEncodedFormat
GetClientVariablesList	ValueList
GetTickCount	WriteOutput

Abs

Returns the absolute value of a number. The absolute value of a number is the number without its sign.

See also Sgn.

Syntax

Abs(*number*)

number
Any number.

Examples

```
<!--- This example shows how to use the ABS function --->
<HTML>
<HEAD>
<TITLE>
Abs Example
</TITLE>
</HEAD>

<BODY bgcolor=silver>
<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.

</BODY>
</HTML>
```

ArrayAppend

Appends an array index to the end of the specified array. Returns a Boolean TRUE on successful completion.

See also ArrayPrepend.

Syntax

ArrayAppend(*array*, *value*)

array
Name of the array to which you want to append an index.

value
The value you want to place into the specified array in the last index position.

Example

```
<!--- This example shows ArrayAppend --->
<HTML>
<HEAD>
<TITLE>ArrayAppend Example</TITLE>
</HEAD>

<BODY>
<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 and 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>

</BODY>
</HTML>
```

ArrayAvg

Returns the average of the values in the specified array.

Syntax

ArrayAvg(*array*)

array
Name of the array containing values you want to average.

Example

```
<!--- This example shows the use of ArrayAvg --->
<HTML>
<HEAD>
<TITLE>ArrayAvg Example</TITLE>
</HEAD>

<BODY>
<H3>ArrayAvg Example</H3>
<!--- create an array --->
<CFSET myNumberArray = ArrayNew(1)>
<!--- run a standard search --->
 <CFSEARCH NAME="SearchSnippets"
        COLLECTION="snippets"
        TYPE="simple"
        CRITERIA="Array*">
<!--- set a temp variable --->
<CFSET temp = 1>
<!--- output the results of the search --->
<CFOUTPUT QUERY="SearchSnippets">
    <CFSET myNumberArray[temp] = score>
<CFSET temp = temp + 1>
</CFOUTPUT>
<!--- use ArrayAvg to get the average score --->
<P>The average score for your search on the term
"Array*" in the collection "Snippets" is
<CFOUTPUT>#ArrayAvg(myNumberArray)# for
#SearchSnippets.RecordCount# "hits"<BR>
</CFOUTPUT>

</BODY>
</HTML>
```

ArrayClear

Deletes all data in the specified array. Returns a Boolean TRUE on successful completion.

See also ArrayDeleteAt.

Syntax

ArrayClear(*array*)

array
Name of the array in which you want to delete data.

Example

```
<!--- This example shows ArrayClear --->
<HTML>
<HEAD>
<TITLE>ArrayClear Example</TITLE>
</HEAD>

<BODY>
<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>

</BODY>
</HTML>
```

ArrayDeleteAt

Deletes data from the specified array at the specified index position. Note that when an array index is deleted, index positions in the array are recalculated. For example, in an array containing the months of the year, deleting index position [5] removes the entry for May. If you then want to delete the entry for November, you delete index position [10], not [11], since the index positions were recalculated after index position [5] was removed.

Returns a Boolean TRUE on successful completion.

See also ArrayInsertAt.

Syntax

ArrayDeleteAt(*array*, *position*)

array
Name of the array in which you want to delete index data specified in position.

position
Array position containing the data you want to delete.

Example

```
<!--- This example shows ArrayDeleteAt --->
<HTML>
<HEAD>
<TITLE>ArrayDeleteAt Example</TITLE>
</HEAD>

<BODY>
<H3>ArrayDeleteAt 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] = "Wednesday">
<!--- delete the second element --->
<P>Is the second element gone?:
  <CFOUTPUT>#ArrayDeleteAt(DaysArray,2)#</CFOUTPUT>
<!--- note that the formerly third element, "Wednesday"
is now the second element --->
<P>The second element is now: <CFOUTPUT>#DaysArray[2]#</CFOUTPUT>

</BODY>
</HTML>
```

ArrayInsertAt

Inserts data in the specified array at the specified index position. All array elements with indexes greater than the new position are shifted right by one. The length of the array increases by one index.

Returns a Boolean TRUE on successful completion.

See also ArrayDeleteAt.

Syntax

ArrayInsertAt(*array*, *position*, *value*)

array
Name of the array in which you want to insert data.

position
The index position in the specified array where you want to insert the data specified in value.

value
The value of the data you want to insert into the array.

Example

```
<!--- This example shows ArrayInsertAt --->
<HTML>
<HEAD>
<TITLE>ArrayInsertAt Example</TITLE>
</HEAD>

<BODY>
<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>
<!--- Notice how the array now has four elements, and that
element 3, "Thursday", has now become element four --->
</BODY>
</HTML>
```

ArrayIsEmpty

Determines whether the specified array is empty of data.

Returns a Boolean TRUE if specified array is empty, FALSE if not empty.

See also ArrayLen.

Syntax

ArrayIsEmpty(*array*)

array
Name of the array you want to check for data.

Example

```
<!--- This example shows ArrayIsEmpty --->
<HTML>
<HEAD>
<TITLE>ArrayIsEmpty Example</TITLE>
</HEAD>

<BODY>
<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>

</BODY>
</HTML>
```

ArrayLen

Returns the length of the specified array.

See also ArrayIsEmpty.

Syntax

ArrayLen(*array*)

array
Name of the array whose length you want to return.

Example

```
<!--- This example shows ArrayLen --->
<HTML>
<HEAD>
<TITLE>ArrayLen Example</TITLE>
</HEAD>

<BODY>
<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>

</BODY>
</HTML>
```

ArrayMax

Returns the largest numeric value in the specified array.

Syntax

ArrayMax(*array*)

array
Name of the array from which you want to return the largest numeric value.

Example

```
<!--- This example shows the use of ArrayMax and ArrayMin --->
<HTML>
<HEAD>
<TITLE>ArrayMax Example</TITLE>
</HEAD>

<BODY>
<H3>ArrayMax Example</H3>

<!--- create an array --->
<CFSET myNumberArray = ArrayNew(1)>
<!--- run a standard search --->
 <CFSEARCH NAME="SearchSnippets"
        COLLECTION="snippets"
        TYPE="simple"
        CRITERIA="Array*">
<!--- set a temp variable --->
<CFSET temp = 1>
<!--- output the results of the search --->
<CFOUTPUT QUERY="SearchSnippets">
    <CFSET myNumberArray[temp] = score>
<CFSET temp = temp + 1>
</CFOUTPUT>
<CFOUTPUT>
The largest numeric element of the
#ArrayLen(myNumberArray)# elements
is #ArrayMax(myNumberArray)#.  The smallest element
is #ArrayMin(myNumberArray)#.
</CFOUTPUT>

</BODY>
</HTML>
```

ArrayMin

Returns the smallest numeric value in the specified array.

Syntax

ArrayMin(*array*)

array
Name of the array from which you want to return the smallest numeric value.

Example

```
<!--- This example shows the use of ArrayMax and ArrayMin --->
<HTML>
<HEAD>
<TITLE>ArrayMin Example</TITLE>
</HEAD>

<BODY>
<H3>ArrayMin Example</H3>

<!--- create an array --->
<CFSET myNumberArray = ArrayNew(1)>
<!--- run a standard search --->
 <CFSEARCH NAME="SearchSnippets"
        COLLECTION="snippets"
        TYPE="simple"
        CRITERIA="Array*">
<!--- set a temp variable --->
<CFSET temp = 1>
<!--- output the results of the search --->
<CFOUTPUT QUERY="SearchSnippets">
    <CFSET myNumberArray[temp] = score>
<CFSET temp = temp + 1>
</CFOUTPUT>
<CFOUTPUT>
The largest numeric element of the
#ArrayLen(myNumberArray)# elements
is #ArrayMax(myNumberArray)#.  The smallest element
is #ArrayMin(myNumberArray)#.
</CFOUTPUT>

</BODY>
</HTML>
```

ArrayNew

Creates an array of between 1 and 3 dimensions. Array elements are indexed with square brackets: [ ].

Note that ColdFusion arrays expand dynamically as data is added.

Syntax

ArrayNew(*dimension*)

dimension
An integer value between 1 and 3.

Examples

```
<!--- This example shows ArrayNew --->
<HTML>
<HEAD>
<TITLE>ArrayNew Example</TITLE>
</HEAD>

<BODY>
<H3>ArrayNew Example</H3>

<!--- Make an array --->
<CFSET MyNewArray = ArrayNew(1)>
<!--- Note that ArrayToList will not function properly
if the Array has not been 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)#
<!--- Note that the array has expanded dynamically
to six elements with the use of ArraySet, even though
we only set three values --->

</CFOUTPUT>
</BODY>
</HTML>
```

ArrayPrepend

Adds an array element to the beginning of the specified array. Returns a Boolean TRUE on successful completion.

See also ArrayAppend.

Syntax

ArrayPrepend(*array*, *value*)

array
Name of the array to which you want to prepend data.

value
The value you want to add to the beginning of the specified array.

Examples

```
<!--- This example shows ArrayPrepend --->
<HTML>
<HEAD>
<TITLE>ArrayPrepend Example</TITLE>
</HEAD>

<BODY>
<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 the query and append these names
successively before the last element (this list will
reverse 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>
</BODY>
</HTML>
```

ArrayResize

Resets an array to a specified minimum number of elements. ArrayResize can provide some performance gains if used to size an array to its expected maximum. Use ArrayResize immediately after creating an array with ArrayNew for arrays greater than 500 elements.

Note that ColdFusion arrays expand dynamically as data is added.

Returns a Boolean TRUE on successful completion.

Syntax

ArrayResize(*array*, *minimum_size*)

array
Name of the array you want to resize.

minimum_size
Minimum size of the specified array.

Example

```
<!--- This example shows the use of ArrayResize --->
<HTML>
<HEAD>
<TITLE>ArrayResize Example</TITLE>
</HEAD>

<BODY>
<H3>ArrayResize Example</H3>

<!--- perform a query to get the list of course --->
<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>

</BODY>
</HTML>
```

ArraySet

In a one-dimensional array, sets the elements in a specified range to the specified value. Useful in initializing an array after a call to ArrayNew. Returns a Boolean TRUE on successful completion.

See also ArrayNew.

Syntax

ArraySet(*array*, *start_pos*, *end_pos*, *value*)

array
Name of the array you want to change.

start_pos
Starting position in the specified array.

end_pos
Ending position in the specified array. If this value exceeds the array length, elements are accordingly added to the array.

value
The value you want to add to the range of elements in the specified array.

Example

```
<!--- This example shows ArraySet --->
<HTML>
<HEAD>
<TITLE>ArraySet Example</TITLE>
</HEAD>

<BODY>
<H3>ArraySet Example</H3>

<!--- Make an array --->
<CFSET MyNewArray = ArrayNew(1)>
<!--- Note that ArrayToList will 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

Returns the specified array with elements numerically or alphanumerically sorted.

Syntax **ArraySort**(*array*, *sort_type* [, *sort_order*])

array
Name of the array you want to sort.

sort_type
The type of sort to execute. Sort type can be:

- `numeric` — Sorts numerically
- `text` — Sorts text alphabetically, uppercase before lowercase
- `textnocase` — Sorts text alphabetically; case is ignored

sort_order
The sort order you want to enforce:

- `asc` — (Default) Ascending sort order
- `desc` — Descending sort order

Example

```
<!--- This example shows ArraySort --->

<HTML>
<HEAD>
<TITLE>ArraySort Example</TITLE>
</HEAD>

<BODY>
<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 myAlphaArray = ArraySort(myArray, "textnocase", "desc")>
...
```

ArraySum

Returns the sum of values in the specified array.

Syntax

ArraySum(*array*)

array
Name of the array containing values you want to add together.

Example

```
<!--- This example shows the use of ArraySum --->

<HTML>
<HEAD>
<TITLE>ArraySum Example</TITLE>
</HEAD>

<BODY>
<H3>ArraySum Example</H3>
<!--- create an array --->
<CFSET myNumberArray = ArrayNew(1)>
<!--- run a standard search --->
 <CFSEARCH NAME="SearchSnippets"
        COLLECTION="snippets"
        TYPE="simple"
        CRITERIA="Array*">
<!--- set a temp variable --->
<CFSET temp = 1>
<!--- output the results of the search --->
<CFOUTPUT QUERY="SearchSnippets">
    <CFSET myNumberArray[temp] = score>
<CFSET temp = temp + 1>
</CFOUTPUT>
<!--- use ArrayAvg to get the average score --->
<P>The average score for your search on the term
"Array*" in the collection "Snippets" is
<CFOUTPUT>
#ArrayAvg(myNumberArray)# for
#SearchSnippets.RecordCount# "hits"<BR>
<P>The sum of the elements in the array is
#ArraySum(myNumberArray)#
</CFOUTPUT>

</BODY>
</HTML>
```

ArraySwap

Swaps array values for the specified array at the specified positions. ArraySwap can be used with greater efficiency than multiple CFSETs.

Returns a Boolean TRUE on successful completion.

Syntax

ArraySwap(*array*, *position1*, *position2*)

array
Name of the array whose elements you want to swap.

position1
Position of the first element you want to swap.

position2
Position of the second element you want to swap.

Example

```
<!--- This example shows ArraySwap --->
<HTML>
<HEAD>
<TITLE>ArraySwap Example</TITLE>
</HEAD>

<BODY>
<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>

</BODY>
</HTML>
```

ArrayToList

Converts the specified one dimensional array to a list, delimited with the character you specify.

Syntax `ArrayToList(array [, delimiter ])`

array
Name of the array containing elements you want to use to build a list.

delimiter
Specify the character(s) you want to use to delimit elements in the list. Default is comma (,).

Example

```
<!--- This example shows ArrayToList --->

<HTML>
<HEAD>
<TITLE>ArrayToList Example</TITLE>
</HEAD>

<BODY>
<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 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>
</BODY>
</HTML>
```

Asc

Returns the ASCII value (character code) of the first character of a string. Returns 0 if string is empty.

See also Chr.

Syntax

`Asc(`*`string`*`)`

string
Any string.

Examples

```
<!--- This code illustrates ASC --->
<HTML>
<HEAD>
<TITLE>
Asc Example
</TITLE>
</HEAD>

<BODY bgcolor=silver>
<H3>Asc Example</H3>

<!--- if the character string is not empty, then
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>Please enter a character</H4>
    </CFIF>
</CFIF>

<FORM ACTION="asc.cfm" METHOD="POST">
<P>Type in 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>

</BODY>
</HTML>
```

Atn

Returns the arctangent of a number. The arctangent is the angle whose tangent is *number*.

See also Tan, Sin, Cos, and Pi.

Syntax

Atn(*number*)

number
Tangent of the angle you want.

Usage

The range of the result is $-\pi/2$ to $\pi/2$ radians. To convert degrees to radians, multiply degrees by $\pi/180$. To convert radians to degrees, multiply radians by $180/\pi$.

Examples

```
<!--- This snippet shows how to use Atn --->
<HTML>
<HEAD>
<TITLE>
Atn Example
</TITLE>
</HEAD>

<BODY bgcolor=silver>
<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>Please enter a number</H4>
    </CFIF>
</CFIF>

<FORM ACTION="atn.cfm" METHOD="POST">
<P>Type in 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>

</BODY>
</HTML>
```

BitAnd

Returns the bitwise AND of two long integers.

See also BitNot, BitOr, and BitXor.

Syntax **BitAnd**(*number1, number2*)

number1, number2
Any long integers.

Usage Bit functions operate on 32-bit integers.

Examples

```
<!--- This example shows BitAnd --->
<HTML>
<HEAD>
<TITLE>BitAnd Example</TITLE>
</HEAD>

<BODY>
<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>

</BODY>
</HTML>
```

BitMaskClear

Returns *number* bitwise cleared with *length* bits beginning from *start*.

See also BitMaskRead and BitMaskSet.

Syntax

BitMaskClear(*number*, *start*, *length*)

number
Long integer to be masked.

start
Integer specifying the starting bit for masking.

length
Integer specifying the length of mask.

Usage

Parameters *start* and *length* must be in the range from 0 to 31.

Bit functions operate on 32-bit integers.

Examples

```
<!--- This example shows BitMaskClear --->
<HTML>
<HEAD>
<TITLE>BitMaskClear Example</TITLE>
</HEAD>

<BODY>
<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>

</BODY>
</HTML>
```

BitMaskRead

Returns the integer created from *length* bits of *number* beginning from *start*.

See also BitMaskClear and BitMaskSet.

Syntax

BitMaskRead(*number*, *start*, *length*)

number
Long integer to be masked.

start
Integer specifying the starting bit for reading.

length
Integer specifying the length of mask.

Usage

Parameters *start* and *length* must be in the range from 0 to 31.

Bit functions operate on 32-bit integers.

Examples

```
<!--- This example shows BitMaskRead --->
<HTML>
<HEAD>
<TITLE>BitMaskRead Example</TITLE>
</HEAD>

<BODY>
<H3>BitMaskRead Example</H3>

<P>Returns integer created from length bits of
number beginning with start.

<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>

</BODY>
</HTML>
```

BitMaskSet

Returns *number* bitwise masked with *length* bits of *mask* beginning from *start*.

See also BitMaskClear and BitMaskRead.

Syntax

BitMaskSet(*number*, *mask*, *start*, *length*)

number
Long integer to be masked.

mask
Long integer specifying the mask.

start
Integer specifying the starting bit in number for masking.

length
Integer specifying the length of mask.

Usage

Parameters *start* and *length* must be in the range from 0 to 31.

Bit functions operate on 32-bit integers.

Examples

```
<!--- This example shows BitMaskSet --->
<HTML>
<HEAD>
<TITLE>BitMaskSet Example</TITLE>
</HEAD>

<BODY>
<H3>BitMaskSet Example</H3>

<P>Returns number bitwise masked with length
bits of mask beginning from 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>

</BODY>
</HTML>
```

BitNot

Returns the bitwise NOT of a long integer.

See also BitAnd, BitOr, and BitXor.

Syntax

BitNot(*number*)

number
Any long integer.

Usage

Bit functions operate on 32-bit integers.

Examples

```
<!--- This example shows BitNot --->
<HTML>
<HEAD>
<TITLE>BitNot Example</TITLE>
</HEAD>

<BODY>
<H3>BitNot Example</H3>

Returns the bitwise NOT of a long integer.

<P>BitNot(0): <CFOUTPUT>#BitNot(0)#</CFOUTPUT>

<P>BitNot(255): <CFOUTPUT>#BitNot(255)#</CFOUTPUT>

</BODY>
</HTML>
```

BitOr

Returns the bitwise OR of two long integers

See also BitAnd, BitNot, and BitXor.

Syntax **BitOr**(*number1, number2*)

number1, number2
Any long integers.

Usage Bit functions operate on 32-bit integers.

Examples

```
<!--- This example shows BitOr --->
<HTML>
<HEAD>
<TITLE>BitOr Example</TITLE>
</HEAD>

<BODY>
<H3>BitOr Example</H3>

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>

</BODY>
</HTML>
```

BitSHLN

Returns *number* bitwise shifted without rotation to the left by *count* bits.

See also BitSHRN.

Syntax **BitSHLN**(*number*, *count*)

number
Long integer to be shifted to the left.

count
Integer specifying number of bits the number should be shifted.

Usage Parameter *count* must be in the range from 0 to 31.

Bit functions operate on 32-bit integers.

Examples

```
<!--- This example shows BitSHLN --->
<HTML>
<HEAD>
<TITLE>BitSHLN Example</TITLE>
</HEAD>

<BODY>
<H3>BitSHLN Example</H3>

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>

</BODY>
</HTML>
```

BitSHRN

Returns *number* bitwise shifted without rotation to the right by *count* bits.

See also BitSHLN.

Syntax

BitSHRN(*number*, *count*)

number
Long integer to be shifted to the right.

count
Integer specifying number of bits the number should be shifted.

Usage

Parameter *count* must be in the range from 0 to 31.

Bit functions operate on 32-bit integers.

Examples

```
<!--- This example shows BitSHRN --->
<HTML>
<HEAD>
<TITLE>BitSHRN Example</TITLE>
</HEAD>

<BODY>
<H3>BitSHRN Example</H3>

Returns the 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>

</BODY>
</HTML>
```

BitXor

Returns bitwise XOR of two long integers.

See also BitAnd, BitNot, and BitOr.

Syntax **BitXor**(*number1, number2*)

number1, number2
Any long integers.

Usage Bit functions operate on 32-bit integers.

Examples

```
<!--- This example shows BitXOr --->
<HTML>
<HEAD>
<TITLE>BitXOr Example</TITLE>
</HEAD>

<BODY>
<H3>BitXOr Example</H3>

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>

</BODY>
</HTML>
```

Ceiling

Returns the closest integer greater than a given number.

See also Int, Fix, and Round.

Syntax **Ceiling**(*number*)

number
Any real number.

Examples

```
<!--- This example illustrates the CF function ceiling --->
<HTML>
<HEAD>
<TITLE>Ceiling Example</TITLE>
</HEAD>

<BODY bgcolor=silver>
<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>

</BODY>
</HTML>
```

Chr

Returns a character of a given ASCII value (character code).

See also Asc.

Syntax

Chr(*number*)

number
Any ASCII value (a number in the range 0 to 255 inclusive).

Usage

Numbers from 0 to 31 are the standard, nonprintable ASCII codes. For example, Chr(10) returns a linefeed character and Chr(13) returns a carriage return character. Therefore, the two-character string Chr(13) & Chr(10) is the newline string.

Examples

```
<!--- This code illustrates CHR --->
<HTML>
<HEAD>
<TITLE>
CHR Example
</TITLE>
</HEAD>

<BODY bgcolor=silver>
<H3>CHR Example</H3>
<!--- if the character string is not empty, then
output its CHR value --->
<CFIF IsDefined("form.charVals")>
        <CFOUTPUT>#form.charVals# = #CHR("#form.charVals#")#</CFOUTPUT>
</CFIF>

<CFFORM ACTION="chr.cfm" METHOD="POST">
<P>Type in a ASCII value to see its character
<BR><CFINPUT TYPE="Text"
       NAME="CharVals"
       RANGE="1,256"
       MESSAGE="Please enter an integer from 1 to 256"
       VALIDATE="integer"
       REQUIRED="No"
       SIZE="3"
       MAXLENGTH="3">
<P><INPUT TYPE="Submit" NAME=""> <INPUT TYPE="RESET">
</CFFORM>

</BODY>
</HTML>
```

CJustify

Centers a string in the specified field length.

See also LJustify and RJustify.

Syntax **Cjustify**(*string*, *length*)

string
Any string to be centered.

length
Length of field.

Examples

```
<!--- 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 bgcolor=silver>
<H3>CJustify</H3>

<P>Enter a string, and it will be center justified within
the sample field

<FORM ACTION="cjustify.cfm" METHOD="POST">
<P><INPUT TYPE="Text" VALUE="<CFOUTPUT>#jString#</CFOUTPUT>" SIZE=35
NAME="justifyString">

<P><INPUT TYPE="Submit" NAME=""> <INPUT TYPE="RESET">

</FORM>

</BODY>
</HTML>
```

Compare

Performs a case-sensitive comparison of two strings. Returns a negative number if *string1* is less than *string2*; 0 if *string1* is equal to *string2*; or a positive number if *string1* is greater than *string2*.

See also CompareNoCase and Find.

Syntax

Compare(*string1*, *string2*)

string1, string2
Strings to be compared.

Usage

The comparison is performed on the ASCII values (character codes) of corresponding characters in *string1* and *string2*.

If many strings are sorted in increasing order based on the Compare function, they appear listed in dictionary order.

Examples

```
<!--- This example shows the use of Compare --->
<HTML>
<HEAD>
<TITLE>
Compare Example
</TITLE>
</HEAD>

<BODY bgcolor=silver>
<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" 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>

</BODY>
</HTML>
```

CompareNoCase

Performs a case-insensitive comparison of two strings. Returns a negative number if *string1* is less than *string2*; 0 if *string1* is equal to *string2*; or a positive number if *string1* is greater than *string2*.

See also Compare and FindNoCase.

Syntax

CompareNoCase(*string1*, *string2*)

string1, string2
Strings to be compared.

Examples

```
<!--- This example shows the use of CompareNoCase --->
<HTML>
<HEAD>
<TITLE>
CompareNoCase Example
</TITLE>
</HEAD>

<BODY bgcolor=silver>
<H3>CompareNoCase Example</H3>

<P>The compare 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>
...
```

Cos

Returns the cosine of a given angle in radians.

See also Sin, Tan, and Pi.

Syntax

Cos(*number*)

number
Angle in radians for which you want 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.

Examples

```
<!--- This snippet shows how to use Cos --->
<HTML>
<HEAD>
<TITLE>
Cos Example
</TITLE>
</HEAD>

<BODY bgcolor=silver>
<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>Please enter a number between -1 and 1</H4>
    </CFIF>
</CFIF>

<FORM ACTION="cos.cfm" METHOD="POST">
<P>Type in 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>

</BODY>
</HTML>
```

CreateDate

Returns a valid date/time object.

See also CreateDateTime and CreateODBCDate.

Syntax

CreateDate(*year*, *month*, *day*)

year
Number representing the year in the range 100–9999.

month
Number representing the month of the year, ranging from 1 (January) to 12 (December).

day
Number representing the day of the month, ranging from 1 to 31.

Usage

CreateDate is a subset of CreateDateTime.

Time in the returned object is set to 00:00:00.

Years less than 100 are interpreted as 20th century values.

Examples

```
<!--- This example shows how to use CreateDate,
CreateDateTime, and CreateODBCDate --->
<HTML>
<HEAD>
<TITLE>
CreateDate Example
</TITLE>
</HEAD>

<BODY bgcolor=silver>
<H3>CreateDate Example</H3>

<CFIF IsDefined("form.year")>
Your date value, presented using different CF date functions:
<CFSET yourDate =
    #CreateDate("#form.year#","#form.month#","#form.day#")#>

<CFOUTPUT>
<UL>
    <LI>Built with CreateDate:
#CreateDate("#form.year#","#form.month#","#form.day#")#
    <LI>Built with CreateDateTime:
#CreateDateTime("#form.year#","#form.month#","#form.day#", 12,13,0)#
    <LI>Built with CreateODBCDate: #CreateODBCDate("#yourDate#")#
    <LI>Built with CreateODBCDateTime: #CreateODBCDateTime("#yourDate#")#
</UL>
```

```
<P>The same value can be formatted with dateFormat:
<UL>
    <LI>Built with CreateDate:
      #DateFormat
        ("#CreateDate("#form.year#","#form.month#","#form.day#")#",
          "mmm-dd-yyyy")#
    <LI>Built with CreateDateTime:
       #DateFormat
          ("#CreateDateTime("#form.year#","#form.month#","#form.day#",
             12,13,0)#")#
    <LI>Built with CreateODBCDate:
      #DateFormat("#CreateODBCDate("#yourDate#")#", "mmmm d, yyyy")#
    <LI>Built with CreateODBCDateTime:
       #DateFormat("#CreateODBCDateTime("#yourDate#")#", "d/m/yy")#
</UL>
</CFOUTPUT>
</CFIF>

<CFFORM ACTION="createdate.cfm" METHOD="POST">
<P>Please enter the year, month and day in integer format for
the date value you would like to view:
<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>

</BODY>
</HTML>
```

CreateDateTime

Returns a valid date/time object.

See also CreateDate, CreateTime, CreateODBCDateTime, and Now.

Syntax

CreateDateTime(*year*, *month*, *day*, *hour*, *minute*, *second*)

year
Number representing the year in the range 100–9999.

month
Number representing the month of the year, ranging from 1 (January) to 12 (December).

day
Number representing the day of the month, ranging from 1 to 31.

hour
Number representing the hour, ranging from 0 to 23.

minute
Number representing the minute, ranging from 0 to 59.

second
Number representing the second, ranging from 0 to 59.

Usage

Years less than 100 are interpreted as 20th century values.

Examples

```
<!--- This example shows how to use CreateDateTime --->
<HTML>
<HEAD>
<TITLE>
CreateDateTime Example
</TITLE>
</HEAD>

<BODY bgcolor=silver>
<H3>CreateDateTime Example</H3>

<CFIF IsDefined("form.year")>
Your date value, presented using different CF date functions:
<CFSET yourDate = #CreateDateTime("#form.year#",
  "#form.month#","#form.day#",  "#form.hour#", "#form.minute#",
    "#form.second#")#>
<CFOUTPUT>
<UL>
    <LI>Built with CreateDate:
      #CreateDate("#form.year#","#form.month#","#form.day#")#
```

```
    <LI>Built with CreateDateTime:
       #DateFormat("#CreateDateTime
         ("#form.year#","#form.month#","#form.day#", "#form.hour#",
           "#form.minute#","#form.second#")#")#
    <LI>Built with CreateODBCDate: #CreateODBCDate("#yourDate#")#
    <LI>Built with CreateODBCDateTime: #CreateODBCDateTime("#yourDate#")#
</UL>

<P>The same value can be formatted with dateFormat:
<UL>
    <LI>Built with CreateDate:
       #DateFormat("#CreateDate
         ("#form.year#","#form.month#","#form.day#")#", "mmm-dd-yyyy")#
    <LI>Built with CreateDateTime:
        #DateFormat("#CreateDateTime
          ("#form.year#","#form.month#","#form.day#", "#form.hour#",
            "#form.minute#","#form.second#")#")#
    <LI>Built with CreateODBCDate:
       #DateFormat("#CreateODBCDate("#yourDate#")#", "mmmm d, yyyy")#
    <LI>Built with CreateODBCDateTime:
       #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
the date value you would like 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>

</BODY>
</HTML>
```

CreateODBCDate

Returns a date in ODBC date format.

See also CreateDate and CreateODBCDateTime.

Syntax

CreateODBCDate(*date*)

date
Date/time object in the period from 100 AD to 9999 AD.

Examples

```
<!--- This example shows how to use CreateDate,
CreateDateTime, CreateODBCDate  --->
<HTML>
<HEAD>
<TITLE>
CreateODBCDate Example
</TITLE>
</HEAD>

<BODY bgcolor=silver>
<H3>CreateODBCDate Example</H3>

<CFIF IsDefined("form.year")>
Your date value, presented using different CF date functions:
<CFSET yourDate = #CreateDateTime("#form.year#","#form.month#",
  "#form.day#", "#form.hour#","#form.minute#","#form.second#")#>

<CFOUTPUT>
<UL>
    <LI>Built with CreateDate:
      #CreateDate("#form.year#","#form.month#","#form.day#")#
    <LI>Built with CreateDateTime: #DateFormat("#CreateDateTime
      ("#form.year#","#form.month#","#form.day#", "#form.hour#",
        "#form.minute#","#form.second#")#")#
    <LI>Built with CreateODBCDate: #CreateODBCDate("#yourDate#")#
    <LI>Built with CreateODBCDateTime: #CreateODBCDateTime("#yourDate#")#
</UL>

<P>The same value can be formatted with DateFormat:
<UL>
    <LI>Built with CreateDate: #DateFormat("#CreateDate
      ("#form.year#","#form.month#","#form.day#")#", "mmm-dd-yyyy")#
    <LI>Built with CreateDateTime: #DateFormat("#CreateDateTime
      ("#form.year#","#form.month#","#form.day#", "#form.hour#",
        "#form.minute#","#form.second#")#")#
    <LI>Built with CreateODBCDate: #DateFormat("#CreateODBCDate
      ("#yourDate#")#", "mmmm d, yyyy")#
    <LI>Built with CreateODBCDateTime: #DateFormat("#CreateODBCDateTime
      ("#yourDate#")#", "d/m/yy")#
</UL>
```

```
</CFOUTPUT>
</CFIF>

<CFFORM ACTION="createODBCdate.cfm" METHOD="POST">
<P>Please enter the year, month and day in integer format for
the date value you would like 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>

</BODY>
</HTML>
```

CreateODBCDateTime

Returns a date/time object in ODBC timestamp format.

See also CreateDateTime, CreateODBCDate, CreateODBCTime, and Now.

Syntax

CreateODBCDateTime(*date*)

date
Date/time object in the period from 100 AD to 9999 AD.

Usage

When passing a date/time value as a string, make sure it is enclosed in quotes. Otherwise, it is interpreted as a number representation of a date/time object, returning undesired results.

Examples

```
<!--- This example shows how to use CreateDate,
CreateDateTime, CreateODBCDate and CreateODBCDateTime --->
<HTML>
<HEAD>
<TITLE>
CreateODBCDateTime Example
</TITLE>
</HEAD>

<BODY bgcolor=silver>
<H3>CreateODBCDateTime Example</H3>

<CFIF IsDefined("form.year")>
Your date value, presented using different CF date functions:
<CFSET yourDate = #CreateDateTime("#form.year#", "#form.month#",
  "#form.day#", "#form.hour#","#form.minute#","#form.second#")#>

<CFOUTPUT>
<UL>
    <LI>Built with CreateDate: #CreateDate("#form.year#",
      "#form.month#","#form.day#")#
    <LI>Built with CreateDateTime: #DateFormat("#CreateDateTime
      ("#form.year#","#form.month#","#form.day#", "#form.hour#",
        "#form.minute#","#form.second#")#")#
    <LI>Built with CreateODBCDate: #CreateODBCDate("#yourDate#")#
    <LI>Built with CreateODBCDateTime: #CreateODBCDateTime("#yourDate#")#
</UL>

<P>The same value can be formatted with dateFormat:
<UL>
    <LI>Built with CreateDate: #DateFormat("#CreateDate
      ("#form.year#","#form.month#","#form.day#")#", "mmm-dd-yyyy")#
    <LI>Built with CreateDateTime: #DateFormat("#CreateDateTime
      ("#form.year#","#form.month#","#form.day#", "#form.hour#",
        "#form.minute#","#form.second#")#")#
    <LI>Built with CreateODBCDate: #DateFormat("#CreateODBCDate
```

```
      ("#yourDate#")#", "mmmm d, yyyy")#
    <LI>Built with CreateODBCDateTime: #DateFormat("#CreateODBCDateTime
      ("#yourDate#")#", "d/m/yy")#
</UL>
</CFOUTPUT>
</CFIF>

<CFFORM ACTION="createODBCdatetime.cfm" METHOD="POST">
<P>Please enter the year, month and day in integer format for
the date value you would like 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>

</BODY>
</HTML>
```

CreateODBCTime

Returns a time object in ODBC time format.

See also CreateODBCDateTime and CreateTime.

Syntax

CreateODBCTime(*date*)

date
Date/time object in the period from 100 AD to 9999 AD.

Usage

When passing a date/time value as a string, make sure it is enclosed in quotes. Otherwise, it is interpreted as a number representation of a date/time object, returning undesired results.

Examples

```
<!--- This code illustrates CreateTime and CreateODBCTime --->
<HTML>
<HEAD>
<TITLE>
CreateODBCTime Example
</TITLE>
</HEAD>

<BODY bgcolor=silver>
<H3>CreateODBCTime Example</H3>

<CFIF IsDefined("form.hour")>
Your time value, presented using different CF time functions:
<CFSET yourTime =
#CreateTime("#form.hour#","#form.minute#","#form.second#")#>

<CFOUTPUT>
<UL>
    <LI>Built with CreateTime: #TimeFormat("#yourTime#")#
    <LI>Built with CreateODBCTime: #CreateODBCTime("#yourTime#")#
</UL>

<P>The same value can be formatted with timeFormat:
<UL>
    <LI>Built with CreateTime: #TimeFormat("#yourTime#", 'hh:mm:ss')#
    <LI>Built with CreateODBCTime:
       #TimeFormat("#yourTime#", 'hh:mm:sstt')#
</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>
</FORM>

</BODY>
</HTML>
```

CreateTime

Returns a valid time variable in ColdFusion.

See also CreateODBCTime and CreateDateTime.

Syntax

CreateTime(*hour*, *minute*, *second*)

hour
Number representing the hour, ranging from 0 to 23.

minute
Number representing the minute, ranging from 0 to 59.

second
Number representing the second, ranging from 0 to 59.

Usage

CreateTime is a subset of CreateDateTime.

Time variables are special cases of date/time variables. The date portion of a time variable is set to December 30, 1899.

Examples

```
<!--- This code illustrates CreateTime and CreateODBCTime --->
<HTML>
<HEAD>
<TITLE>
CreateTime Example
</TITLE>
</HEAD>

<BODY bgcolor=silver>
<H3>CreateTime Example</H3>

<CFIF IsDefined("form.hour")>
Your time value, presented using different CF time functions:
<CFSET yourTime = #CreateTime("#form.hour#",
  "#form.minute#","#form.second#")#>

<CFOUTPUT>
<UL>
    <LI>Built with CreateTime: #TimeFormat("#yourTime#")#
    <LI>Built with CreateODBCTime: #CreateODBCTime("#yourTime#")#
</UL>

<P>The same value can be formatted with timeFormat:
<UL>
    <LI>Built with CreateTime: #TimeFormat("#yourTime#", 'hh:mm:ss')#
    <LI>Built with CreateODBCTime:
       #TimeFormat("#yourTime#", 'hh:mm:sstt')#
</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>

</BODY>
</HTML>
```

CreateTimeSpan

Creates a date/time object for adding and subtracting other date/time objects.

See also CreateDateTime, DateAdd, and DateDiff.

Syntax

CreateTimeSpan(*days*, *hours*, *minutes*, *seconds*)

days
Number representing the number of days.

hours
Number representing the number of hours.

minutes
Number representing the number of minutes.

seconds
Number representing the number of seconds.

Usage

The CreateTimeSpan function creates a special date/time object that should only be used to add and subtract from other date/time objects or with the CFQUERY CACHEDWITHIN attribute.

Examples

```
<!--- This example shows how to CreateTimeSpan --->
...
<CFIF IsDefined("form.year")>

<!--- set variables for the date and for the time span --->
<CFSET yourDate = #CreateDateTime("#form.year#",
  "#form.month#","#form.day#",
"#form.hour#","#form.minute#","#form.second#")#>
<CFSET yourTimeSpan = #CreateTimeSpan("#form.tsday#",
  "#form.tshour#","#form.tsminute#","#form.tssecond#")#>

<CFOUTPUT>
<P>Your original date value: #yourDate#
<P>The date of your timespan, formatted:

<!--- output the results of the form --->
<P>#yourTimeSpan# days <CFIF #yourTimeSpan# LTE 0>before
  your<CFELSE>after your</CFIF> original date:
<BR>#DateFormat("#yourDate#" + "#yourTimeSpan#")#
 #TimeFormat("#yourDate#" + "#yourTimeSpan#")#
</CFOUTPUT>
</CFIF>
...
```

DateAdd

Returns a date to which a specified time interval has been added.

See also DateDiff, DatePart, and CreateTimeSpan.

Syntax

DateAdd(*datepart*, *number*, *date*)

datepart
One of the following strings:

- 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 or negative to get dates in the past).

date
Date/time object in the period from 100 AD to 9999 AD.

Usage

The *datepart* specifiers "y," "d," and "w" perform the same function — add a certain number of days to a given date.

When passing a date/time value as a string, make sure it is enclosed in quotes. Otherwise, it is interpreted as a number representation of a date/time object, returning undesired results.

Examples

```
<!--- 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 will expire.  (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>
    <TD>#Subject#</TD>
    <TD>#Posted# <CFIF DateAdd("#type#", "#value#",
       "#posted#") LT #Now()#>EXPIRED</CFIF></TD>
</TR>
</CFOUTPUT>
</TABLE>
...
```

DateCompare

Performs a full date/time comparison of two dates. Returns -1 if *date1* is less than *date2*; returns 0 if *date1* is equal to *date2*; returns 1 if *date1* is greater than *date2*.

See also CreateDateTime and DatePart.

Syntax

DateCompare(*date1*, *date2*)

date1
Date/time object in the period from 100 AD to 9999 AD.

date2
Date/time object in the period from 100 AD to 9999 AD.

Usage

When passing a date/time value as a string, make sure it is enclosed in quotes. Otherwise, it is interpreted as a number representation of a date/time object, returning undesired results.

Examples

```
<!--- This example shows the use of DateCompare --->
...
<CFIF IsDefined("form.date1")>
<CFIF IsDate("#form.date1#") and IsDate("#form.date2#")>
<CFSET comparison = DateCompare("#form.date1#", "#form.date2#")>
<!--- switch on the variable to give various responses --->
<CFSWITCH EXPRESSION=#comparison#>
<CFCASE value="-1">
    <H3><CFOUTPUT>#DateFormat("#form.date1#")#</CFOUTPUT>
      (Date 1) is earlier than <CFOUTPUT>#DateFormat("#form.date2#")#
         </CFOUTPUT> (Date 2)</H3>
    <I>The dates are not equal</I>
</CFCASE>
<CFCASE value="0">
    <H3><CFOUTPUT>#DateFormat("#form.date1#")#</CFOUTPUT>
      (Date 1) is equal to <CFOUTPUT>#DateFormat("#form.date2#")#
         </CFOUTPUT> (Date 2)</H3>
    <I>The dates are equal!</I>
</CFCASE>
<CFCASE value="1">
    <H3><CFOUTPUT>#DateFormat("#form.date1#")#</CFOUTPUT>
      (Date 1) is later than <CFOUTPUT>#DateFormat("#form.date2#")#
         </CFOUTPUT> (Date 2)</H3>
    <I>The dates are not equal</I>
</CFCASE>
<CFDEFAULTCASE>
    <H3>This is the default case</H3>
</CFDEFAULTCASE>
</CFSWITCH>
...
```

DateDiff

Returns the number of intervals in whole units of type *Datepart* by which *Date1* is less than *Date2*.

See also DateAdd, DatePart, and CreateTimeSpan.

Syntax `DateDiff(datepart, date1, date2)`

datepart
One of the following strings:

- `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 period from 100 AD to 9999 AD.

date2
Date/time object in the period from 100 AD to 9999 AD.

Usage If you want to know the number of days between *date1* and *date2*, you can use either Day of Year ("y") or Day ("d").

When *datepart* is Weekday ("w"), DateDiff returns the number of weeks between the two dates. If *date1* falls on a Monday, DateDiff counts the number of Mondays until *date2*. It counts *date2* but not *date1*. If interval is Week ("ww"), however, the DateDiff function returns the number of calendar weeks between the two dates. It counts the number of Sundays between *date1* and *date2*. DateDiff counts *date2* if it falls on a Sunday; but it doesn't count *date1*, even if it does fall on a Sunday.

If *Date1* refers to a later point in time than *date2*, the DateDiff function returns a negative number.

When passing date/time value as a string, make sure it is enclosed in quotes. Otherwise, it is interpreted as a number representation of a date/time object returning undesired results.

Examples

```
<!--- 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

Returns a formatted date/time value. If no mask is specified, DateFormat function returns date value using the *dd-mmm-yy* format.

See also Now, CreateDate, and ParseDateTime.

Syntax `DateFormat(date [, mask ])`

date
Date/time object in the period from 1601 AD to 9999 AD.

mask
Set of characters that are used to show how ColdFusion should display the date:

- `d` — Day of the month as digits with no leading zero for single-digit days.
- `dd` — Day of the month as digits with a 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 with no leading zero for single-digit months.
- `Mm` — Month as digits with a 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 with no leading zero for years less than 10.
- `yy` — Year as last two digits with a leading zero for years less than 10.
- `yyyy` — Year represented by four digits.
- `gg` — Period/era string. Currently ignored, but reserved for future use

Usage When passing a date/time value as a string, make sure it is enclosed in quotes. Otherwise, it is interpreted as a number representation of a date/time object, returning undesired results.

Examples

```
<!--- This example shows the various types of output
possible with DateFormat --->
<HTML>
<HEAD>
<TITLE>
DateFormat Example
</TITLE>
</HEAD>

<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 a number of
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>

</BODY>
</HTML>
```

DatePart

Returns the specified part of a date as an integer.

See also DateAdd and DateDiff.

Syntax

DatePart(*datepart*, *date*)

datepart
One of the following strings:

- yyyy — Year
- q — Quarter
- m — Month
- y — Day of year
- d — Day
- w — Weekday
- ww — Week
- h — Hour
- n — Minute
- s — Second

date
Any date.

Usage

Years less than 100 are interpreted as 20th century values.

When passing a date/time value as a string, make sure it is enclosed in quotes. Otherwise, it is interpreted as a number representation of a date/time object, returning undesired results.

Examples

```
<!--- This example shows information available from DatePart --->
<HTML>
<HEAD>
<TITLE>
DatePart Example
</TITLE>
</HEAD>

<CFSET todayDate = #Now()#>
<BODY>
<H3>DatePart Example</H3>

<P>Today's date is <CFOUTPUT>#todayDate#</CFOUTPUT>.

<P>Using datepart, we can extract an integer representing
```

```
the various 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>

</BODY>
</HTML>
```

Day

Returns the ordinal for the day of the month, ranging from 1 to 31.

See also DayOfWeek, DayOfWeekAsString, DayOfYear, DaysInMonth, DaysInYear, and FirstDayOfMonth.

Syntax

Day(*date*)

date
Any date.

Usage

Years less than 100 are interpreted as 20th century values.

When passing a date/time value as a string, make sure it is enclosed in quotes. Otherwise, it is interpreted as a number representation of a date/time object, returning undesired results.

Examples

```
<!--- This example shows the value of the Day function --->
<HTML>
<HEAD>
<TITLE>
Day Example
</TITLE>
</HEAD>

<BODY bgcolor=silver>
<H3>Day 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>
...
```

DayOfWeek

Returns the ordinal for the day of the week. The day is given as an integer ranging from 1 (Sunday) to 7 (Saturday).

See also Day, DayOfWeekAsString, DayOfYear, DaysInMonth, DaysInYear, and FirstDayOfMonth.

Syntax

DayOfWeek(*date*)

date
Any date.

Usage

Years less than 100 are interpreted as 20th century values.

When passing a date/time value as a string, make sure it is enclosed in quotes. Otherwise, it is interpreted as a number representation of a date/time object, returning undesired results.

Examples

```
<!--- This example shows the value of the DayOfWeek function --->
<HTML>
<HEAD>
<TITLE>
DayofWeek Example
</TITLE>
</HEAD>

<BODY bgcolor=silver>
<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

Returns the day of the week corresponding to *day_of_week*, an integer ranging from 1 (Sunday) to 7 (Saturday).

See also Day, DayOfWeek, DayOfYear, DaysInMonth, DaysInYear, and FirstDayOfMonth.

Syntax

DayOfWeekAsString(*day_of_week*)

day_of_week
Integer representing the day of the week, where 1 is Sunday, 2 is Monday, and so on.

Usage

Years less than 100 are interpreted as 20th century values.

Examples

```
<!--- shows the value of the dayOfWeekAsString function --->
<HTML>
<HEAD>
<TITLE>
DayOfWeekAsString Example
</TITLE>
</HEAD>

<BODY bgcolor=silver>
<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>
...
```

DayOfYear

Returns the ordinal for the day of the year.

See also Day, DayOfWeek, DayOfWeekAsString, DaysInMonth, DaysInYear, and FirstDayOfMonth.

Syntax

DayOfYear(*date*)

date
Any date.

Usage

DayofYear is aware of leap years.

Years less than 100 are interpreted as 20th century values.

When passing a date/time value as a string, make sure it is enclosed in quotes. Otherwise, it is interpreted as a number representation of a date/time object, returning undesired results.

Examples

```
<!--- shows the value of the DayOfYear function --->
<HTML>
<HEAD>
<TITLE>
DayOfYear Example
</TITLE>
</HEAD>

<BODY bgcolor=silver>
<H3>DayOfYear 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>
...
```

DaysInMonth

Returns the number of days in the specified month (*Date*).

See also Day, DayOfWeek, DayOfWeekAsString, DayOfYear, DaysInYear, and FirstDayOfMonth.

Syntax

DaysInMonth(*date*)

date
Any date.

Usage

Years less than 100 are interpreted as 20th century values.

When passing a date/time value as a string, make sure it is enclosed in quotes. Otherwise, it is interpreted as a number representation of a date/time object, returning undesired results.

Examples

```
<!--- shows the value of the DaysInMonth function --->
<HTML>
<HEAD>
<TITLE>
DaysInMonth Example
</TITLE>
</HEAD>

<BODY bgcolor=silver>
<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>
...
```

DaysInYear

Returns the number of days in a year.

See also Day, DayOfWeek, DayOfWeekAsString, DayOfYear, DaysInMonth, DaysInYear, FirstDayOfMonth, and IsLeapYear.

Syntax

DaysInYear(*date*)

date
Any date.

Usage

DaysInYear is aware of leap years.

Years less than 100 are interpreted as 20th century values.

When passing a date/time value as a string, make sure it is enclosed in quotes. Otherwise, it is interpreted as a number representation of a date/time object, returning undesired results.

Examples

```
<!--- shows the value of the DaysInYear function --->
<HTML>
<HEAD>
<TITLE>
DaysInYear Example
</TITLE>
</HEAD>

<BODY bgcolor=silver>
<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#")#).
...
```

DE

Returns its argument with double quotes wrapped around it and all double quotes inside it escaped. The DE (Delay Evaluation) function prevents the evaluation of a string as an expression when it is passed as an argument to IIf or Evaluate.

See also Evaluate and IIf.

Syntax

DE(*string*)

string
String to be evaluated with delay.

Examples

```
<!--- This shows the use of DE and Evaluate --->
<HTML>
<HEAD>
<TITLE>
DE Example
</TITLE>
</HEAD>

<BODY bgcolor=silver>
<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 fishing --->
<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

Returns *number* as a string formatted with two decimal places and thousands separator.

See also DollarFormat and NumberFormat.

Syntax

DecimalFormat(*number*)

number
Number being formatted.

Examples

```
<!--- This code shows the use of DecimalFormat --->
<HTML>
<HEAD>
<TITLE>
DecimalFormat Example
</TITLE>
</HEAD>

<BODY bgcolor=silver>
<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>

</BODY>
</HTML>
```

DecrementValue

Returns integer part of *number* decremented by one.

See also IncrementValue.

Syntax **DecrementValue**(*number*)

number
Number being decremented.

Examples

```
<!--- This shows the use of DecrementValue --->
<HTML>
<HEAD>
<TITLE>
DecrementValue Example
</TITLE>
</HEAD>

<BODY>
<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>

</BODY>
</HTML>
```

Decrypt

Decrypts an encrypted string.

See also Encrypt.

Syntax

Decrypt(*encrypted_string*, *key*)

encrypted_string
String to be decrypted.

key
String specifying the key used to encrypt *encrypted_string*.

Examples

```
<!--- This example shows the use of Encrypt and Decrypt --->
<HTML>
<HEAD>
<TITLE>Decrypt Example</TITLE>
</HEAD>

<BODY bgcolor=silver>
<H3>Decrypt Example</H3>

<P>This function allows for the encryption and decryption of a
string. Try it out by entering your own string and a key of your
own choosing and seeing 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" method="post">
<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>
</BODY>
</HTML>
```

DeleteClientVariable

Deletes the client variable specified by *name*. Returns a Boolean TRUE when variable is successfully deleted, even if variable did not previously exist. To test for the existence of a variable, use IsDefined.

See also GetClientVariablesList.

Syntax

DeleteClientVariable("*name*")

name
Name of a client variable to be deleted, surrounded by double quotes.

Usage

If the client variable specified by *name* does not exist, an error is returned.

Example

```
<!--- This view-only example shows DeleteClientVariable --->
<HTML>
<HEAD>
<TITLE>DeleteClientVariable Example</TITLE>
</HEAD>

<BODY>
<!--- this example is view only --->
<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 existance 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>
--->

</BODY>
</HTML>
```

DirectoryExists

Returns YES if the directory specified in the argument does exist; otherwise, it returns NO.

See also FileExists.

Syntax

DirectoryExists(*absolute_path*)

absolute_path
Any absolute path.

Examples

```
<!--- This example shows the use of DirectoryExists --->
<HTML>
<HEAD>
<TITLE>
DirectoryExists Example
</TITLE>
</HEAD>

<BODY bgcolor=silver>
<H3>DirectoryExists Example</H3>

<CFSET thisPath="#ExpandPath("*.*")#">
<CFSET thisDirectory="#GetDirectoryFromPath("#thisPath#")#">
<CFSET thisDirectory=#Left("#thisDirectory#",
  "#Evaluate("#Len("#thisDirectory#")# - 1")#")#>

<CFOUTPUT>
The current directory is: #GetDirectoryFromPath("#thisPath#")#
<CFIF IsDefined("form.yourDirectory")>
<CFIF form.yourDirectory is not "">
<CFSET yourDirectory = form.yourDirectory>
    <CFIF DirectoryExists(#yourdirectory#)>
    <P>Your directory exists.  You entered
    a valid directory name, #yourdirectory#

...
```

DollarFormat

Returns *number* as a string formatted with two decimal places, thousands separator, dollar sign. Parentheses are used if *number* is negative.

See also DecimalFormat and NumberFormat.

Syntax

DollarFormat(*number*)

number
Number being formatted.

Examples

```
<!--- This example shows the use of DollarFormat --->
<HTML>
<HEAD>
<TITLE>
DollarFormat Example
</TITLE>
</HEAD>

<BODY bgcolor=silver>
<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>
```

Encrypt

Encrypts a string.

See also Decrypt.

Syntax

Encrypt(*string*, *key*)

string
String to be encrypted.

key
String specifying the key used to encrypt *string*.

Examples

```
<!--- This example shows the use of Encrypt and Decrypt --->
<HTML>
<HEAD>
<TITLE>Encrypt Example</TITLE>
</HEAD>

<BODY bgcolor=silver>
<H3>Encrypt Example</H3>

<P>This function allows for the encryption and decryption of a
string. Try it out by entering your own string and a key of your
own choosing and seeing 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" method="post">
<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>
</BODY>
</HTML>
```

Evaluate

The function evaluates all of its arguments, left to right, and returns the result of evaluating the last argument.

See also DE and IIf.

Syntax **Evaluate**(*string_expression1* [, *string_expression2* [, …]])

string_expression1, string_expression2
Valid expressions to be evaluated.

Usage String expressions can be arbitrarily complex. Note, however, that they are somewhat more complicated to write because they are inside a string. In particular, if the string expression is double-quoted, double-quotes inside the expression must be escaped.

Examples

```
<!--- This shows the use of DE and Evaluate --->
<HTML>
<HEAD>
<TITLE>
Evaluate Example
</TITLE>
</HEAD>

<BODY bgcolor=silver>
<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

Returns e raised to the power of *number*. The constant e equals 2.71828182845904, the base of the natural logarithm.

See also Log and Log10.

Syntax

Exp(*number*)

number
Exponent applied to the base e.

Usage

To calculate powers of other bases, use ^ (the exponentiation operator). Exp is the inverse of Log, the natural logarithm of number.

Examples

```
<!--- This example shows how to use Exp --->
<HTML>
<HEAD>
<TITLE>
Exp Example
</TITLE>
</HEAD>

<BODY bgcolor=silver>
<H3>Exp 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="exp.cfm" METHOD="POST">
Enter a number to see its value raised to the E power,
the natural logarithm of that number, 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>
</BODY>
</HTML>
```

ExpandPath

Returns a path equivalent to the *relative_path* appended to the base template path. Note the following:

- ExpandPath creates a platform-appropriate path. You can use either a slash (/) or a back slash (\) in the specified relative path.
- The return value contains a trailing slash (or back slash) if the specified relative path contains a trailing slash (or back slash).

See also FileExists, GetDirectoryFromPath, and GetFileFromPath.

Syntax

ExpandPath(*relative_path*)

relative_path
Any relative path. ExpandPath converts relative directory references (.\ and ..\)to an absolute path. The function throws an error if this argument or the resulting absolute path is invalid.

Examples

```
<!--- This example shows the use of ExpandPath --->
<HTML>
<HEAD>
<TITLE>
ExpandPath Example
</TITLE>
</HEAD>

<BODY bgcolor=silver>
<H3>ExpandPath 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:
...
```

FileExists

Returns YES if the file specified in the argument does exist; otherwise, it returns NO.

See also DirectoryExists, ExpandPath, and GetTemplatePath.

Syntax

FileExists(*absolute_path*)

absolute_path
Any absolute path.

Examples

```
<!--- This example shows the use of FileExists --->
<HTML>
<HEAD>
<TITLE>
FileExists Example
</TITLE>
</HEAD>

<BODY bgcolor=silver>
<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

Returns the first index of an occurrence of a *substring* in a *string* from a specified starting position. Returns 0 if *substring* is not in *string*. The search is case-sensitive.

See also FindNoCase, Compare, FindOneOf, REFind, and Replace.

Syntax

Find(*substring*, *string* [, *start*])

substring
String being sought.

string
String being searched.

start
Starting position for the search.

Examples

```
...
<!--- depending upon the action desired, perform
different function --->
<CFSWITCH EXPRESSION="#tag#">
    <CFCASE value="find">
        <CFSET TheAction = Find("#form.MyString#",
          "#CFHTTP.FileContent#", 1)>
    </CFCASE>
    <CFCASE value="findNoCase">
        <CFSET TheAction = FindNoCase("#form.MyString#",
          "#CFHTTP.FileContent#", 1)>
    </CFCASE>
    <CFCASE value="findOneof">
        <CFSET TheAction = FindOneOf("#form.MyString#",
          "#CFHTTP.FileContent#", 1)>
    </CFCASE>
    <CFDEFAULTCASE>
        <CFSET TheAction = Find("#form.MyString#",
          "#CFHTTP.FileContent#", 1)>
    </CFDEFAULTCASE>
</CFSWITCH>
...
```

FindNoCase

Returns the first index of an occurrence of a *substring* in a *string* from a specified starting position. Returns 0 if *substring* is not in *string*. The search is case-insensitive.

See also Find, CompareNoCase, FindOneOf, REFind, and Replace functions.

Syntax `FindNoCase(substring, string [, start ])`

substring
String being sought.

string
String being searched.

start
Starting position for the search.

Examples

```
...
<!--- depending upon the action desired, perform
different function --->
<CFSWITCH EXPRESSION="#tag#">
    <CFCASE value="find">
        <CFSET TheAction = Find("#form.MyString#",
          "#CFHTTP.FileContent#", 1)>
    </CFCASE>
    <CFCASE value="findNoCase">
        <CFSET TheAction = FindNoCase("#form.MyString#",
          "#CFHTTP.FileContent#", 1)>
    </CFCASE>
    <CFCASE value="findOneof">
        <CFSET TheAction = FindOneOf("#form.MyString#",
          "#CFHTTP.FileContent#", 1)>
    </CFCASE>
    <CFDEFAULTCASE>
        <CFSET TheAction = Find("#form.MyString#",
          "#CFHTTP.FileContent#", 1)>
    </CFDEFAULTCASE>
</CFSWITCH>
...
```

FindOneOf

Return the first index of the occurrence of any character from *set* in *string*. Returns 0 if no characters are found. The search is case-sensitive.

See also Find, Compare, and REFind functions.

Syntax

FindOneOf(*set*, *string* [, *start*])

set
String containing one or more characters being sought.

string
String being searched.

start
Starting position for the search.

Examples

```
...
<!--- depending upon the action desired, perform
different function --->
<CFSWITCH EXPRESSION="#tag#">
    <CFCASE value="find">
        <CFSET TheAction = Find("#form.MyString#",
          "#CFHTTP.FileContent#", 1)>
    </CFCASE>
    <CFCASE value="findNoCase">
        <CFSET TheAction = FindNoCase("#form.MyString#",
          "#CFHTTP.FileContent#", 1)>
    </CFCASE>
    <CFCASE value="findOneof">
        <CFSET TheAction = FindOneOf("#form.MyString#",
          "#CFHTTP.FileContent#", 1)>
    </CFCASE>
    <CFDEFAULTCASE>
        <CFSET TheAction = Find("#form.MyString#",
          "#CFHTTP.FileContent#", 1)>
    </CFDEFAULTCASE>
</CFSWITCH>
...
```

FirstDayOfMonth

Returns the ordinal (the day's number in the year) for the first day of the specified month.

See also Day, DayOfWeek, DayOfWeekAsString, DayOfYear, DaysInMonth, and DaysInYear.

Syntax

FirstDayOfMonth(*date*)

date
Any date.

Usage

Years less than 100 are interpreted as 20th century values.

When passing a date/time value as a string, make sure it is enclosed in quotes. Otherwise, it is interpreted as a number representation of a date/time object, returning undesired results.

Examples

```
<!--- This example shows the use of FirstDayOfMonth --->
<HTML>
<HEAD>
<TITLE>
FirstDayOfMonth Example
</TITLE>
</HEAD>

<BODY bgcolor=silver>
<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>

</BODY>
</HTML>
```

Fix

Returns the closest integer less than *number* if *number* is greater than or equal to 0. Returns the closest integer greater than *number* if *number* is less than 0.

See also Ceiling, Int, and Round.

Syntax

`Fix(`*number*`)`

number
Any number.

Examples

```
<!--- This example shows the use of Fix --->
<HTML>
<HEAD>
<TITLE>
Fix Example
</TITLE>
</HEAD>

<BODY>
<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>

</BODY>
</HTML>
```

FormatBaseN

Converts a *number* to a string in the base specified by *radix*.

See also InputBaseN.

Syntax **FormatBaseN**(*number*, *radix*)

number
Number to be converted.

radix
Base of the result.

Examples

```
<!--- This example shows FormatBaseN and InputBaseN--->
<HTML>
<HEAD>
<TITLE>
FormatBaseN Example
</TITLE>
</HEAD>

<BODY>
<H3>FormatBaseN 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, an integer ranging
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>

</BODY>
</HTML>
```

GetBaseTagData

Returns an object that contains data (variables, scopes, etc.) from a specified ancestor tag. By default the closest ancestor is returned. If there is no ancestor by the specified name, or if the ancestor does not expose any data (for example, CFIF), an exception will be thrown.

See also GetBaseTagList.

Syntax

GetBaseTagData(*tagname* [, *instancenumber*])

tagname
Required. Specifies the ancestor tag name for which the function returns data.

instancenumber
Optional. Specifies the number of ancestor levels to jump before returning data. The default is 1.

Example

```
<!--- This example illustrates usage of the GetBaseTagData
      function. This is 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 out the tag's execution mode --->
       Located inside the
       <cfif tagData.thisTag.executionMode neq 'inactive'>
           template
       <cfelse>
           body
       </cfif>
...
```

GetBaseTagList

Returns a comma-delimited list of uppercase ancestor tag names. The first element of the list is the parent tag. If you call this function for a top-level tag, it returns an empty string.

See also GetBaseTagData.

Syntax `GetBaseTagList()`

Example

```
<!--- This example illustrates usage of the GetBaseTagList
      function. This is 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>
...
```

GetClientVariablesList

Returns a comma-delimited list of non-readonly client variables available to a template.

See also DeleteClientVariable.

Syntax `GetClientVariablesList()`

Example

```
<!--- This view-only example shows GetClientVariablesList --->
<HTML>
<HEAD>
<TITLE>GetClientVariablesList Example</TITLE>
</HEAD>

<BODY>
<!--- 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 existance 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>
--->

</BODY>
</HTML>
```

GetDirectoryFromPath

Extracts the directory (with a \ (backslash)) from a fully specified path.

See also ExpandPath and GetFileFromPath.

Syntax

GetDirectoryFromPath(*path*)

path
Fully specified path (drive, directory, filename, and extension).

Examples

```
<!--- This example shows the use of GetDirectoryFromPath --->
<HTML>
<HEAD>
<TITLE>
GetDirectoryFromPath Example
</TITLE>
</HEAD>

<BODY bgcolor=silver>
<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 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#">
...
```

GetFileFromPath

Extracts the filename from a fully specified path.

See also ExpandPath and GetDirectoryFromPath.

Syntax

GetFileFromPath(*path*)

path
Fully qualified path (drive, directory, filename, and extension).

Examples

```
<!--- This example shows the use of GetFileFromPath--->
<HTML>
<HEAD>
<TITLE>
GetFileFromPath Example
</TITLE>
</HEAD>

<BODY bgcolor=silver>
<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#">
...
```

GetLocale

Returns the locale for the current request. Locales are determined by the native operating system.

A locale is an encapsulation of the set of attributes that govern the display and formatting of international date, time, number, and currency values.

See also SetLocale.

Syntax `GetLocale()`

Locale support

ColdFusion can be expected to support the following locales with a default Windows NT installation.

Locales Supported by ColdFusion		
Dutch (Belgian)	French (Canadian)	Norwegian (Bokmal)
Dutch (Standard)	French (Standard)	Norwegian (Nynorsk)
English (Australian)	French (Swiss)	Portuguese (Brazilian)
English (Canadian)	German (Austrian)	Portuguese (Standard)
English (New Zealand)	German (Standard)	Spanish (Mexican)
English (UK)	German (Swiss)	Spanish (Modern)
English (US)	Italian (Standard)	Spanish (Standard)
French (Belgian)	Italian (Swiss)	Swedish

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. GetLocale( ) will return an entry from that list. SetLocale will fail if called with a locale name not on that list.

Example

```
<!--- This example shows GetLocale --->
<HTML>
<HEAD>
<TITLE>GetLocale Example</TITLE>
</HEAD>

<BODY>
<H3>GetLocale Example</H3>

<P>GetLocale returns the locale for the current
```

```
request.  Locales are determined by the native
operating system.

<P>A locale is an encapsulation of the set of attributes
that govern the display and formatting of international
date, time, number, and currency values.

<P>The locale for this system is <CFOUTPUT>#GetLocale()#</CFOUTPUT>

</BODY>
</HTML>
```

GetTempDirectory

Returns the temporary directory (with a trailing backslash) of ColdFusion Server. In Windows NT, the temporary directory is determined by the value of the TMP or TEMP environment variables.

See also GetTempFile.

Syntax **GetTempDirectory()**

Example

```
<!--- This example uses GetTempDirectory to find
the temporary directory, and GetTempFile to place
a dummy file in that directory --->
<HTML>
<HEAD>
<TITLE>
GetTempDirectory Example
</TITLE>
</HEAD>

<BODY>
<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>

</BODY>
</HTML>
```

GetTempFile

Creates and returns the name of a temporary file in a directory whose name starts with (at most) the first three characters of *prefix*.

See also GetTempDirectory.

Syntax

GetTempFile(*dir*, *prefix*)

dir
Directory name.

prefix
Prefix of a temporary file to be created in the directory specified by dir.

Examples

```
<!--- This example uses GetTempDirectory to find
the temporary directory, and GetTempFile to place
a dummy file in that directory --->
<HTML>
<HEAD>
<TITLE>
GetTempFile Example
</TITLE>
</HEAD>

<BODY>
<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>

</BODY>
</HTML>
```

GetTemplatePath

Returns the fully specified path of the base template.

See also FileExists and ExpandPath.

Syntax `GetTemplatePath()`

Example

```
<!--- This example uses GetTemplatePath to show
the template path of the current page --->
<HTML>
<HEAD>
<TITLE>
GetTemplatePath Example
</TITLE>
</HEAD>

<BODY>
<H3>GetTemplatePath Example</H3>

<P>The template path of the current page is:
<CFOUTPUT>#GetTemplatePath()#</CFOUTPUT>

</BODY>
</HTML>
```

GetTickCount

Returns a millisecond clock counter that can be used for timing sections of CFML code or any other aspects of page processing.

Syntax `GetTickCount()`

Usage The absolute value of the counter has no meaning. Generate useful timing values by taking differences between the results of GetTickCount() at specified points during page processing.

Examples

```
<!--- This example calls the GetTickCount
      function to track execution time --->
<html>
<body>
<!--- 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>

</body>
</html>
```

GetToken

Returns the specified token in a string. Default delimiters are spaces, tabs, and newline characters. If *index* is greater than the number of tokens in *string*, GetToken returns an empty string.

See also Left, Right, Mid, SpanExcluding, and SpanIncluding.

Syntax

GetToken(*string*, *index* [, *delimiters*])

string
Any string.

index
Any integer > 0 that indicates position of a token.

delimiters
String containing sets of delimiters.

Examples

```
<!--- This example shows the use of GetToken --->
<HTML>
<HEAD>
<TITLE>
GetToken Example
</TITLE>
</HEAD>

<BODY bgcolor=silver>
<H3>GetToken Example</H3>

<CFIF IsDefined("form.yourString")>
<!--- set delimiter --->
<CFIF form.yourDelimiter is not "">
    <CFSET yourDelimiter = form.yourDelimiter>
<CFELSE>
    <CFSET yourDelimiter = " ">
</CFIF>
<!--- check that 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>
...
```

Hour

Returns the ordinal value for the hour, ranging from 0 to 23.

See also DatePart, Minute, and Second.

Syntax

Hour(*date*)

date
Any date.

Usage

Years less than 100 are interpreted as 20th century values.

When passing a date/time value as a string, make sure it is enclosed in quotes. Otherwise, it is interpreted as a number representation of a date/time object, returning undesired results.

Examples

```
<!--- This example shows the use of Hour, Minute, and Second --->
<HTML>
<HEAD>
<TITLE>
Hour Example
</TITLE>
</HEAD>

<BODY bgcolor=silver>
<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>

</BODY>
</HTML>
```

HTMLCodeFormat

Returns HTML escaped *string* enclosed in <PRE> and </PRE> tags. All carriage returns are removed from *string*, and all special characters (> < " &) are escaped.

See also HTMLEditFormat.

Syntax

HTMLCodeFormat(*string* [, *version*])

string
String being HTML escaped and preformatted.

version
The specific HTML version to use in replacing special characters with their entity references. Valid entries are:

- -1 — The latest implementation of HTML
- 2.0 — For HTML 2.0 (Default)
- 3.2 — For HTML 3.2

Example

```
<!--- This example shows the use of HTMLCodeFormat
and HTMLEditFormat --->

<HTML>
<HEAD>
<TITLE>
HTMLCodeFormat Example
</TITLE>
</HEAD>

<BODY bgcolor=silver>
<H3>HTMLCodeFormat Example</H3>

<FORM ACTION="htmlcodeformat.cfm" METHOD="POST">
Try entering a URL for the tag to return in HTMLCodeFormat
and HTMLEditFormat:
<INPUT TYPE="Text" size=25 NAME="urladdress"
  VALUE="http://www.allaire.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>

</BODY>
</HTML>
```

HTMLEditFormat

Returns HTML escaped *string*. All carriage returns are removed from *string*, and all special characters (> < " &) are escaped.

See also HTMLCodeFormat.

Syntax

HTMLEditFormat(*string* [, *version*])

string
String being HTML escaped.

version
The specific HTML version to use in replacing special characters with their entity references. Valid entries are:

- -1 – The latest implementation of HTML
- 2.0 – For HTML 2.0 (Default)
- 3.2 – For HTML 3.2

Usage

By escaping all special characters, this function increases the length of the specified string. This can cause unpredictable results when performing certain string functions (Left, Right, and Mid, for example) against the expanded string.

Example

```
<!--- This example shows the use of HTMLCodeFormat
and HTMLEditFormat --->
<HTML>
<HEAD>
<TITLE>
HTMLEditFormat Example
</TITLE>
</HEAD>

<BODY bgcolor=silver>
<H3>HTMLEditFormat Example</H3>

<FORM ACTION="htmleditformat.cfm" METHOD="POST">
Try entering a URL for the tag to return in HTMLCodeFormat
and HTMLEditFormat:
<INPUT TYPE="Text" size=25 NAME="urladdress"
  VALUE="http://www.allaire.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>

</BODY>
</HTML>
```

IIf

The function evaluates its *condition* as a Boolean. If the result is TRUE it returns the value of Evaluate(*string_expression1*); otherwise, it returns the value of Evaluate(*string_expression2*).

See also DE and Evaluate.

Syntax

IIf(*condition*, *string_expression1*, *string_expression2*)

condition
Any expression that can be evaluated as a Boolean.

string_expression1
Valid string expression to be evaluated and returned if condition is TRUE.

string_expression2
Valid string expression to be evaluated and returned 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>
```

returning *result*.

The expressions *string_expression1* and *string_expression2* must be string expressions, so that they do not get evaluated immediately as the arguments of IIf. For example:

```
IIf(y is 0, DE("Error"), x/y)
```

will generate error if y=0 because the third argument is the value of x/0 (not a valid expression).

Remember that ColdFusion evaluates *string_expression1* and *string_expression2*. To return the string itself instead of evaluate the expression, use the DE (delay evaluation) function.

Examples

```
<!--- This example shows IIf --->
<HTML>
<HEAD>
<TITLE>
IIf Example
</TITLE>
</HEAD>

<BODY bgcolor=silver>
<H3>IIf Function</H3>
```

```
<P>IIf evaluates a condition, then performs 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>The result of the expression
IIf( Hour(Now()) GT 12,
  DE("It is afternoon or evening"),
    DE("It is morning"))
is:<BR>
<CFOUTPUT>
#IIf( Hour(Now()) GT 12,
  DE("It is afternoon or evening"),
    DE("It is morning"))#
</CFOUTPUT>

</BODY>
</HTML>
```

IncrementValue

Returns integer part of *number* incremented by one.

See also DecrementValue.

Syntax

IncrementValue(*number*)

number
Number being incremented.

Examples

```
<!--- This shows the use of IncrementValue --->
<HTML>
<HEAD>
<TITLE>
IncrementValue Example
</TITLE>
</HEAD>

<BODY>
<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>

</BODY>
</HTML>
```

InputBaseN

Returns the number obtained by converting *string* using the base specified by *radix*, an integer ranging from 2 to 36.

See also FormatBaseN.

Syntax

InputBaseN(*string*, *radix*)

string
Any string representing number in base specified by radix.

radix
Base of number represented by string ranging from 2 to 36.

Examples

```
<!--- This example shows FormatBaseN and InputBaseN--->
<HTML>
<HEAD>
<TITLE>
InputBaseN Example
</TITLE>
</HEAD>

<BODY>
<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, an integer ranging
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>

</BODY>
</HTML>
```

Insert

Inserts a *substring* in a *string* after a specified character *position*. Prepends the *substring* if *position* is equal to 0.

See also RemoveChars and Len.

Syntax **Insert**(*substring*, *string*, *position*)

substring
String to be inserted.

string
String to be inserted into.

position
Integer that indicates the character position in string where the substring will be inserted.

Examples

```
<!--- This example shows the use of Insert --->
<HTML>
<HEAD>
<TITLE>
Insert Example
</TITLE>
</HEAD>

<BODY bgcolor=silver>
<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>
<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>
...
```

Int

Returns the closest integer smaller than a number.

See also Ceiling, Fix, and Round.

Syntax

Int(*number*)

number
Real number you want to round down to an integer.

Examples

```
<!--- This example shows the use of Int --->
<HTML>
<HEAD>
<TITLE>
Int Example
</TITLE>
</HEAD>

<BODY bgcolor=silver>
<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>
</BODY>
</HTML>
```

IsArray

Returns TRUE if value is an array.

See also Array Functions.

Syntax

IsArray(*value* [, *number*])

value
Variable name or array name.

number
Tests if the array has exactly the specified dimension.

Examples

```
<!--- This example shows IsArray --->
<HTML>
<HEAD>
<TITLE>IsArray Example</TITLE>
</HEAD>

<BODY>
<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>

</BODY>
</HTML>
```

IsAuthenticated

Returns TRUE if the user has been authenticated for the specified ColdFusion security context.

See also CFAUTHENTICATE, IsAuthorized.

Syntax `IsAuthenticated( )`

Example

```
<!--- This example calls the IsAuthenticated function. --->
<!--- This code is from an Application.cfm file --->
<CFIF NOT IsAuthenticated()>
  <CFTRY>
  <CFAUTHENTICATE SECURITYCONTEXT="Allaire" USERNAME=#user#
    PASSWORD=#pwd#>
  <CFCATCH TYPE="Security">
   <!--- the message to display --->
   <H3>Authentication error</H3>
    <CFOUTPUT>
    <!--- Display the message. Alternatively, you might place
      code here to define the user to the security context. --->
    <P>#CFCATCH.message#
    </CFOUTPUT>
  </CFCATCH>
  </CFTRY>
</CFIF>
<CFAPPLICATION NAME="Personnel">
</BODY>
</HTML>
```

IsAuthorized

Returns TRUE if the user is authorized to perform the specified action on the specified ColdFusion resource.

See also IsAuthenticated.

Syntax `IsAuthorized(resourcetype, resourcename [, action ])`

resourcetype
String specifying the type of resource:

- Application
- CFML
- File
- DSN
- Component
- Collection
- CustomTag
- UserObject

resourcename
String specifying the name of the resource. The value specified varies depending on the resource type:

***resourcetype* specification**	***resourcename* specification**
APPLICATION	Application name
CFML	CFML tag name
FILE	File name
DSN	Data source name
COMPONENT	Component name
COLLECTION	Verity collection name
CUSTOMTAG	Custom tag name
USEROBJECT	Object name

Resourcename is the actual resource that is protected, not to be confused with the rule name, which you specify in the ColdFusion Administrator.

action
String specifying the action for which authorization is requested. Do not specify this parameter for COMPONENT and CUSTOMTAG. For all other resource types, this parameter is required.

***resourcetype* specification**	**Possible ACTIONs**
APPLICATION	ALL USECLIENTVARIABLES
CFML	Valid actions for the tag specified by *resourcename*
FILE	READ WRITE
DSN	ALL CONNECT SELECT INSERT UPDATE DELETE SP (stored procedure)
COMPONENT	No actions for this resource type
COLLECTION	DELETE OPTIMIZE PURGE SEARCH UPDATE
CUSTOMTAG	No actions for this resource type
USEROBJECT	Action specified via the ColdFusion Administrator

Usage If you specify THROWONFAILURE=Yes in the CFAUTHENTICATE tag, you can enclose IsAuthorized in a CFTRY/CFCATCH block to handle possible exceptions programmatically.

Example

```
<!---  This example calls the IsAuthorized
       function. --->
...
<!--- Is user is authorized to select information from the
    Orders data source? --->
```

```
<CFIF IsAuthorized("Datasource", "Orders", "select")>
   <CFQUERY name="GetList" datasource="Orders">
       SELECT *    FROM Orders
   </CFQUERY>
   <CFOUTPUT QUERY="GetList">
       Authorization Succeeded. Order information follows:
       #Customer# - #BalanceDue#<BR>
    </CFOUTPUT>
</CFIF>

</BODY>
</HTML>
```

IsBoolean

Returns TRUE if *value* can be converted to a Boolean; otherwise, FALSE.

See also IsNumeric and YesNoFormat.

Syntax

IsBoolean(*value*)

value
Any number or string.

Examples

```
<!--- This example shows the use of IsBoolean --->
<HTML>
<HEAD>
<TITLE>
IsBoolean Example
</TITLE>
</HEAD>

<BODY bgcolor=silver>
<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" METHOD="POST">
<P>Enter an expression, and discover if
it can be evaluated to a Boolean value.

<INPUT TYPE="Text" NAME="TheTestValue" VALUE="1">
<INPUT TYPE="Submit" VALUE="Is it Boolean?" NAME="">
</FORM>
</BODY>
</HTML>
```

IsDate

Returns TRUE if *string* can be converted to a date/time value; otherwise, FALSE. Note that ColdFusion converts the boolean return value to its string equivalent, "Yes" and "No."

See also ParseDateTime, CreateDateTime, and IsNumericDate.

Syntax

IsDate(*string*)

string
Any string value.

Usage

Years less than 100 are interpreted as 20th century values.

Examples

```
<!--- This example shows the use of IsDate --->
<HTML>
<HEAD>
<TITLE>
IsDate Example
</TITLE>
</HEAD>

<BODY bgcolor=silver>
<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" METHOD="POST">
<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>

</BODY>
</HTML>
```

IsDebugMode

Returns TRUE if debugging mode was set via the ColdFusion Administrator and FALSE if debugging mode is disabled.

Syntax

```
IsDebugMode( )
```

Examples

```
<!--- This example shows the use of IsDebugMode --->
<HTML>
<HEAD>
<TITLE>
IsDebugMode Example
</TITLE>
</HEAD>

<BODY bgcolor=silver>
<H3>IsDebugMode Example</H3>

<CFIF IsDebugMode()>
  <H3>Debugging has been set via the ColdFusion Administrator</H3>
<CFELSE>
  <H3>Debugging is disabled</H3>
</CFIF>

</BODY>
</HTML>
```

IsDefined

Evaluates a string value to determine if the variable named in the string value exists. IsDefined returns TRUE if the specified variable is found, FALSE if not found.

IsDefined provides an alternative to the ParameterExists function, eliminating the need for cumbersome expressions used to test for the existence of a variable:

```
Evaluate("ParameterExists(#var_name#)")
```

See also Evaluate.

Syntax

IsDefined("*variable_name*")

variable_name
A string value, the name of the variable you want to test for. This value must always be enclosed in quotation marks.

Example

```
<!--- This example shows the use of IsDefined --->
<HTML>
<HEAD>
<TITLE>
IsDefined Example
</TITLE>
</HEAD>

<BODY bgcolor=silver>
<H3>IsDefined Example</H3>

<CFIF IsDefined("form.myString")>
<P>Because the variable form.myString has been defined, we
can now show its contents.  This construction allows us to place a form
and its resulting action template in the same template, while using
IsDefined to control the flow of template execution.
<P>The value of "form.myString" is <B><I><CFOUTPUT>#form.myString#
  </CFOUTPUT></I></B>
<CFELSE>
<P>During the first time through this template, the variable
"form.myString" has not yet been defined, so it is not evaluated.
</CFIF>

...

</BODY>
</HTML>
```

IsLeapYear

Returns TRUE if the *year* is a leap year; otherwise, FALSE.

See also DaysInYear.

Syntax

IsLeapYear(*year*)

year
Number representing the year.

Examples

```
<!--- This example shows the use of IsLeapYear --->
<HTML>
<HEAD>
<TITLE>
IsLeapYear Example
</TITLE>
</HEAD>

<BODY bgcolor=silver>
<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" METHOD="POST">
<P>Enter a year value, and find out if it is a valid Leap Year.

<P><INPUT TYPE="Text" NAME="TheTestValue"
  VALUE="<CFOUTPUT>#Year("#Now()#")#</CFOUTPUT>">
<INPUT TYPE="Submit" VALUE="Is it a Leap Year?" NAME="">
</FORM>

</BODY>
</HTML>
```

IsNumeric

Returns TRUE if *string* can be converted to a number; otherwise, FALSE.

See also IsBoolean.

Syntax

IsNumeric(*string*)

string
Any string value.

Examples

```
<!--- This example shows the use of IsNumeric --->
<HTML>
<HEAD>
<TITLE>
IsNumeric Example
</TITLE>
</HEAD>

<BODY bgcolor=silver>
<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" METHOD="POST">
<P>Enter a string, and discover if
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>

</BODY>
</HTML>
```

IsNumericDate

Evaluates "real value" of date/time object. Returns TRUE if the number represents "real value" of the date/time object; otherwise, FALSE.

See also IsDate and ParseDateTime.

Syntax

IsNumericDate(*number*)

number
Real number.

Examples

```
<!--- This example shows the use of IsNumericDate --->
<HTML>
<HEAD>
<TITLE>
IsNumericDate Example
</TITLE>
</HEAD>

<BODY bgcolor=silver>
<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" METHOD="POST">
<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>

</BODY>
</HTML>
```

IsQuery

Returns TRUE if value is a query.

See also QueryNew.

Syntax **IsQuery**(*value*)

value
Query variable.

Example

```
<!--- Shows an example of IsQuery and IsSimpleValue --->
<HTML>
<HEAD>
<TITLE>
IsQuery Example
</TITLE>
</HEAD>

<BODY bgcolor=silver>
<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>

</BODY>
</HTML>
```

IsSimpleValue

Returns TRUE if value is a string, number, Boolean, or date/time value.

Syntax **IsSimpleValue**(*value*)

value
Variable or expression.

Example

```
<!--- Shows an example of IsQuery and IsSimpleValue --->
<HTML>
<HEAD>
<TITLE>
IsSimpleValue Example
</TITLE>
</HEAD>

<BODY bgcolor=silver>
<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>

</BODY>
</HTML>
```

IsStruct

Returns TRUE if *variable* is a structure.

See also Struct Functions.

Syntax **IsStruct**(*variable*)

variable
Variable name.

Examples

```
<!--- This view-only example illustrates usage 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> --->
```

LCase

Returns *string* converted to lowercase.

See also UCase.

Syntax

LCase(*string*)

string
String being converted to lowercase.

Examples

```
<!--- This example shows the use of LCase --->
<HTML>
<HEAD>
<TITLE>
LCase Example
</TITLE>
</HEAD>

<BODY bgcolor=silver>
<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" METHOD="POST">
<P>Enter your sample text, and press "submit" to see
the text returned in lowercase:

<P><INPUT TYPE="Text" NAME="SampleText" VALUE="SAMPLE">

<INPUT TYPE="Submit" NAME="" VALUE="submit">
</FORM>

</BODY>
</HTML>
```

Left

Returns the count of characters from the beginning of a string argument.

See also Right, Mid, and Len.

Syntax `Left(`*`string`*`, `*`count`*`)`

string
String from which the leftmost characters are retrieved.

count
Positive integer indicating how many characters to return.

Examples

```
<!---  This example shows the use of Left --->
<HTML>
<HEAD>
<TITLE>
Left Example
</TITLE>
</HEAD>

<BODY bgcolor=silver>
<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

Returns the length of a string.

See also Left, Right, and Mid.

Syntax

Len(*string*)

string
Any string.

Examples

```
<!---  This example shows the use of Len --->
<HTML>
<HEAD>
<TITLE>
Len Example
</TITLE>
</HEAD>

<BODY bgcolor=silver>
<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" METHOD="POST">
<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>

</BODY>
</HTML>
```

ListAppend

Returns *list* with *value* appended behind its last element.

See also ListPrepend, ListInsertAt, and ListSetAt.

Syntax

ListAppend(*list*, *value* [, *delimiters*])

list
Any list.

value
Number or list being appended.

delimiters
Set of delimiters used in list.

Usage

When appending an element into a list, ColdFusion needs to insert a delimiter. If *delimiters* contains more than one delimiter, ColdFusion defaults to the first delimiter in the string, or , (comma) if *delimiters* was omitted.

If you intend to use list functions on strings that are delimited by the conjunction ", " (comma-space), as is common in HTTP header strings such as the COOKIE header, we recommend that you specify *delimiters* to include both comma and space because ColdFusion Server does not skip white space. For example,

```
ListAppend(List, "MyCookie", "," & CHR(32))
```

Examples

```
<!--- This example shows ListAppend --->
<HTML>
<HEAD>
<TITLE>ListAppend Example</TITLE>
</HEAD>

<BODY>
<H3>ListAppend Example</H3>
<!--- First, query to get some values for our list --->
<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

Returns *list* with all delimiter characters changed to *new_delimiter* string.

See also ListFirst and ListRest.

Syntax `ListChangeDelims(list, new_delimiter [, delimiters ])`

list
List of delimiters being changed.

new_delimiter
String being used as a new delimiter.

delimiters
Set of delimiters used in list.

Examples

```
<!--- This example shows ListChangeDelims --->
<HTML>
<HEAD>
<TITLE>ListChangeDelims Example</TITLE>
</HEAD>

<BODY>
<H3>ListChangeDelims Example</H3>

<P>ListChangeDelims allows you to change the delimiters
used in a list.
<!--- First, query to get some values for our list --->
<CFQUERY NAME="GetParkInfo" DATASOURCE="cfsnippets">
SELECT PARKNAME,CITY,STATE
FROMPARKS
WHEREPARKNAME LIKE 'BA%'
</CFQUERY>
<CFSET temp = #ValueList(GetParkInfo.ParkName)#>
<CFOUTPUT>
<P>The original list: #temp#
</CFOUTPUT>
<!--- now, change the delimiters in the list from "," to
"|:P|"--->
<CFSET temp2 = ListChangeDelims("#Temp#", "|:P|", ",")>
<CFOUTPUT>
<P>The appended list: #temp2#
</CFOUTPUT>

</BODY>
</HTML>
```

ListContains

Returns the index of the first element of a list that contains the specified substring within elements. The search is case-sensitive. If no element is found, returns zero (0).

See also ListContainsNoCase and ListFind.

Syntax `ListContains(list, substring [, delimiters ])`

list
List being searched.

substring
String being sought in elements of list.

delimiters
Set of delimiters used in list.

Examples

```
<!--- This example shows ListContains --->
<HTML>
<HEAD>
<TITLE>ListContains Example</TITLE>
</HEAD>

<BODY bgcolor=silver>
<H3>ListContains Example</H3>

<CFIF IsDefined("form.letter")>
    <!--- First, query to get some values for our list --->
    <CFQUERY NAME="GetParkInfo" DATASOURCE="cfsnippets">
    SELECT  PARKNAME,CITY,STATE
    FROM  PARKS
    WHERE PARKNAME LIKE '#form.letter#%'
    </CFQUERY>
    <CFSET tempList = #ValueList(GetParkInfo.City)#>
    <CFIF ListContains("#tempList#", "#form.yourCity#") is not 0 OR
      form.yourCity is "">
...
```

ListContainsNoCase

Returns the index of the first element of a list that contains the specified substring within elements. The search is case-insensitive. If no element is found, returns 0.

See also ListContains and ListFindNoCase.

Syntax

ListContainsNoCase(*list*, *substring* [, *delimiters*])

list
List being searched.

substring
String being sought in elements of list.

delimiters
Set of delimiters used in list.

Examples

```
<!--- This example shows ListContainsNoCase --->
<HTML>
<HEAD>
<TITLE>ListContainsNoCase Example</TITLE>
</HEAD>

<BODY bgcolor=silver>
<H3>ListContainsNoCase Example</H3>

<CFIF IsDefined("form.letter")>
    <!--- First, query to get some values for our list --->
    <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

Returns *list* with element deleted at the specified position.

See also ListGetAt, ListSetAt, and ListLen.

Syntax

ListDeleteAt(*list*, *position* [, *delimiters*])

list
Any list.

position
Positive integer indicating the position of the element being deleted.

delimiters
Set of delimiters used in list.

Examples

```
<!--- This example shows ListDeleteAt --->
<HTML>
<HEAD>
<TITLE>ListDeleteAt Example</TITLE>
</HEAD>

<BODY>
<H3>ListDeleteAt Example</H3>

<!--- First, query to get some values for our list --->
<CFQUERY NAME="GetParkInfo" DATASOURCE="cfsnippets">
SELECT PARKNAME,CITY,STATE
FROM  PARKS
WHERE PARKNAME LIKE 'CH%'
</CFQUERY>
<CFSET temp = #ValueList(GetParkInfo.ParkName)#>
<CFSET deleted_item = ListGetAt("#temp#", "3", ",")>
<CFOUTPUT>
<P>The original list: #temp#
</CFOUTPUT>
<!--- now, delete the third item from the list --->
<CFSET temp2 = ListDeleteAt("#Temp#", "3", ",")>
<CFOUTPUT>
<P>The changed list: #temp2#
<BR><I>Note that <B>#deleted_item#</B> is not longer present
at position three of the list.</I>
</CFOUTPUT>

</BODY>
</HTML>
```

ListFind

Returns the index of the first occurrence of a value within a list. Returns 0 if no value is found. The search is case-sensitive.

See also ListContains and ListFindNoCase.

Syntax

ListFind(*list*, *value* [, *delimiters*])

list
List being searched.

value
Number or string being sought among elements of list.

delimiters
Set of delimiters used in list.

Examples

```
<!--- This example uses ListFind and ListFindNoCase to see if a
substring exists in a list --->
...

<CFSET myList = ValueList(SearchEmpLastName.LastName)>
<!--- 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.LastName)#", "#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>
...
```

ListFindNoCase

Returns the index of the first occurrence of a value within a list. Returns 0 if no value was found. The search is case-insensitive.

See also ListContains and ListFind.

Syntax

ListFindNoCase(*list*, *value* [, *delimiters*])

list
List being searched.

value
Number or string being sought among elements of list.

delimiters
Set of delimiters used in list.

Examples

```
<!--- This example uses ListFind and ListFindNoCase to see if a
substring exists in a list --->
...
    <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.LastName)#",
          "#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>

</BODY>
</HTML>
```

ListFirst

Returns the first element of the list.

See also ListGetAt, ListLast, and ListRest.

Syntax

ListFirst(*list* [, *delimiters*])

list
List whose first element is being retrieved.

delimiters
Set of delimiters used in list.

Examples

```
<!--- This example shows ListFirst, ListLast, and ListRest --->
<HTML>
<HEAD>
<TITLE>ListFirst Example</TITLE>
</HEAD>

<BODY>
<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)>
<!--- Show the first user in the list --->
<P>The first user in the list is <CFOUTPUT>#ListFirst("#temp#")#
  </CFOUTPUT>.
<P>The rest of the users in the list is as follows:
<CFOUTPUT>#ListRest("#temp#")#</CFOUTPUT>.
<P>The last user in the list is <CFOUTPUT>#ListLast("#temp#")#</CFOUTPUT>

</BODY>
</HTML>
```

ListGetAt

Returns the element at a given position.

See also ListFirst, ListLast, ListRest, and ListSetAt.

Syntax

ListGetAt(*list*, *position* [, *delimiters*])

list
List whose first element is being retrieved.

position
Positive integer indicating the position of the element being retrieved.

delimiters
Set of delimiters.

Examples

```
<!--- This example shows ListGetAt and ListLen --->
<HTML>
<HEAD>
<TITLE>ListGetAt Example</TITLE>
</HEAD>

<BODY>
<H3>ListGetAt Example</H3>

<!--- Find a list of users who wrote messages --->
<CFQUERY NAME="GetMessageUser" DATASOURCE="cfsnippets">
SELECTUsername, 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>

</BODY>
</HTML>
```

ListInsertAt

Returns *list* with *value* inserted at the specified position.

See also ListDeleteAt, ListAppend, ListPrepend, and ListSetAt.

Syntax

ListInsertAt(*list*, *position*, *value* [, *delimiters*])

list
Any list.

position
Position where the value is being inserted.

value
Number or list being inserted.

delimiters
Set of delimiters used in list.

Usage

When inserting elements into a list, ColdFusion needs to insert a delimiter. If *delimiters* contain more than one delimiter, ColdFusion defaults to the first delimiter in the string, or , (comma) if *delimiters* was omitted.

If you intend to use list functions on strings that are delimited by the conjunction ", " (comma-space), as is common in HTTP header strings such as the COOKIE header, we recommend that you specify *delimiters* to include both comma and space because ColdFusion Server does not skip white space.

Examples

```
<!--- This example shows ListInsertAt --->
...
<!--- First, query to get some values for our list. --->
<CFQUERY NAME="GetParkInfo" DATASOURCE="cfsnippets">
SELECT PARKNAME,CITY,STATE
FROM PARKS WHERE PARKNAME LIKE 'CH%'
</CFQUERY>
<CFSET temp = #ValueList(GetParkInfo.ParkName)#>
<CFSET insert_at_this_item = ListGetAt("#temp#", "3", ",")>

<CFOUTPUT>
<P>The original list: #temp#
</CFOUTPUT>
<!--- Now, insert an item at position three. --->
<CFSET temp2 = ListInsertAt("#Temp#", "3", "my Inserted Value", ",")>
...
```

ListLast

Returns the last element of the list.

See also ListGetAt and ListFirst.

Syntax

ListLast(*list* [, *delimiters*])

list
List whose last element is being retrieved.

delimiters
Set of delimiters used in list.

Examples

```
<!--- This example shows ListFirst, ListLast, and ListRest --->
<HTML>
<HEAD>
<TITLE>ListLast Example</TITLE>
</HEAD>

<BODY>
<H3>ListLast 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)>
<!--- Show the first user in the list --->
<P>The first user in the list is <CFOUTPUT>#ListFirst("#temp#")#
  </CFOUTPUT>.
<P>The rest of the users in the list is as follows:
<CFOUTPUT>#ListRest("#temp#")#</CFOUTPUT>.
<P>The last user in the list is <CFOUTPUT>#ListLast("#temp#")#</CFOUTPUT>

</BODY>
</HTML>
```

ListLen

Returns the number of elements in the list.

See also ListAppend, ListDeleteAt, ListInsertAt, and ListPrepend.

Syntax

ListLen(*list* [, *delimiters*])

list
Any list.

delimiters
Set of delimiters used in list.

Examples

```
<!--- This example shows ListGetAt and ListLen --->
<HTML>
<HEAD>
<TITLE>ListLen Example</TITLE>
</HEAD>

<BODY>
<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>

</BODY>
</HTML>
```

ListPrepend

Returns *list* with *value* inserted at the first position, shifting all other elements one to the right.

See also ListAppend, ListInsertAt, and ListSetAt.

Syntax

ListPrepend(*list*, *value* [, *delimiters*])

list
Any list.

value
Number or list being prepended.

delimiters
Set of delimiters used in list.

Usage

When prepending an element to a list, ColdFusion needs to insert a delimiter. If *delimiters* contain more than one delimiter, ColdFusion defaults to the first delimiter in the string, or , (comma) if *delimiters* was omitted.

If you intend to use list functions on strings that are delimited by the conjunction ", " (comma-space), as is common in HTTP header strings such as the COOKIE header, we recommend that you specify *delimiters* to include both comma and space because ColdFusion Server does not skip white space.

Examples

```
<!--- This example shows ListPrepend --->
...
<!--- First, query to get some values for our list --->
<CFQUERY NAME="GetParkInfo" DATASOURCE="cfsnippets">
SELECT  PARKNAME,CITY,STATE
FROM  PARKS
WHERE PARKNAME LIKE 'DE%'
</CFQUERY>
<CFSET temp = #ValueList(GetParkInfo.ParkName)#>
<CFSET first_item = ListFirst("#temp#")>

<CFOUTPUT>
<P>The original list: #temp#
</CFOUTPUT>
<!--- now, insert an item at position 1--->
<CFSET temp2 = ListPrepend("#Temp#", "my Inserted Value", ",")>
...
```

ListRest

Returns *list* without its first element. Returns an empty list (empty string) if *list* has only one element.

See also ListFirst, ListGetAt, and ListLast.

Syntax

ListRest(*list* [, *delimiters*])

list
List whose elements are being retrieved.

delimiters
Set of delimiters used in list.

Examples

```
<!--- This example shows ListFirst, ListLast, and ListRest --->
<HTML>
<HEAD>
<TITLE>ListRest Example</TITLE>
</HEAD>

<BODY>
<H3>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)>
<!--- Show the first user in the list --->
<P>The first user in the list is <CFOUTPUT>#ListFirst("#temp#")#
  </CFOUTPUT>.
<P>The rest of the users in the list is as follows:
<CFOUTPUT>#ListRest("#temp#")#</CFOUTPUT>.
<P>The last user in the list is <CFOUTPUT>#ListLast("#temp#")#</CFOUTPUT>

</BODY>
</HTML>
```

ListSetAt

Returns *list* with *value* assigned to its element at specified position.

See also ListDeleteAt, ListGetAt, and ListInsertAt.

Syntax

ListSetAt(*list*, *position*, *value* [, *delimiters*])

list
Any list.

position
Any position.

value
Any value.

delimiters
Set of delimiters.

Usage

When assigning an element to a list, ColdFusion needs to insert a delimiter. If *delimiters* contain more than one delimiter, ColdFusion defaults to the first delimiter in the string, or , (comma) if *delimiters* was omitted.

If you intend to use list functions on strings that are delimited by the conjunction ", " (comma-space), as is common in HTTP header strings such as the COOKIE header, we recommend that you specify *delimiters* to include both comma and space because ColdFusion Server does not skip white space.

Examples

```
<!--- This example shows ListSetAt --->
<HTML>
<HEAD>
<TITLE>ListSetAt Example</TITLE>
</HEAD>

<BODY>
<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 ChangedItem = 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 item 2, "<CFOUTPUT>#changedItem#</CFOUTPUT>", has
been altered to "I changed this subject" using ListSetAt.

</BODY>
</HTML>
```

ListToArray

Converts the specified list, delimited with the character you specify, to an array.

See also ArrayToList.

Syntax **ListToArray**(*list* [, *delimiter*])

array
Name of the list variable containing elements you want to use to build an array. You can define a list variable with a CFSET statement.

delimiter
Specify the character(s) used to delimit elements in the list. Default is comma (,).

Example

```
<!--- This example shows ListSetAt --->
<HTML>
<HEAD>
<TITLE>ListSetAt Example</TITLE>
</HEAD>

<BODY>
<H3>ListSetAt Example</H3>

<!--- Find a list of users who wrote messages --->
<CFQUERY NAME="GetMessageUser" DATASOURCE="cfsnippets">
SELECTUsername, 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.

</BODY>
</HTML>
```

LJustify

Returns left-justified *string* of the specified field length.

See also CJustify and RJustify.

Syntax **LJustify**(*string*, *length*)

string
String to be left-justified.

length
Length of field.

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 bgcolor=silver>
<H3>LJustify Function</H3>

<P>Enter a string, and it will be left justified within
the sample field

<FORM ACTION="ljustify.cfm" METHOD="POST">
<P><INPUT TYPE="Text" VALUE="<CFOUTPUT>#jString#</CFOUTPUT>"
  SIZE=35 NAME="justifyString">

<P><INPUT TYPE="Submit" NAME=""> <INPUT TYPE="RESET">
</FORM>

</BODY>
</HTML>
```

Log

Returns the natural logarithm of a number. Natural logarithms are based on the constant e (2.71828182845904).

See also Exp and Log10.

Syntax

Log(*number*)

number
Positive real number for which you want the natural logarithm.

Examples

```
<!--- This example shows how to use Log --->
<HTML>
<HEAD>
<TITLE>
Log Example
</TITLE>
</HEAD>

<BODY bgcolor=silver>
<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>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="log.cfm" METHOD="POST">
Enter a number to see its value raised to the E power,
the natural logarithm of that number, 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>

</BODY>
</HTML>
```

Log10

Returns the logarithm of *number* to base 10.

See also Exp and Log.

Syntax

Log10(*number*)

number
Positive real number for which you want the logarithm.

Examples

```
<!--- This example shows how to use Log10 --->
<HTML>
<HEAD>
<TITLE>
Log10 Example
</TITLE>
</HEAD>

<BODY bgcolor=silver>
<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" METHOD="POST">
Enter a number to see its value raised to the E power,
the natural logarithm of that number, 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>

</BODY>
</HTML>
```

LSCurrencyFormat

Returns a currency value using the locale convention. Default value is "local."

Syntax `LSCurrencyFormat(number [, type ])`

number
The currency value.

type
Currency type. Valid arguments are:

- `none` — (For example, 10.00)
- `local` — (Default. For example, $10.00)
- `international` — (For example, USD10.00)

Currency output

The following table shows sample currency output for some of the locales supported by ColdFusion in each of the format types: `local`, `international`, and `none`.

Locale	Format Type Output

Dutch (Belgian)	`Local: 100.000,00 BF`
	`International: BEF100.000,00`
	`None: 100.000,00`

Dutch (Standard)	`Local: fl 100.000,00`
	`International: NLG100.000,00`
	`None: 100.000,00`

English (Australian)	`Local: $100,000.00`
	`International: AUD100,000.00`
	`None: 100,000.00`

English (Canadian)	`Local: $100,000.00`
	`International: CAD100,000.00`
	`None: 100,000.00`

English (New Zealand)	`Local: $100,000.00`
	`International: NZD100,000.00`
	`None: 100,000.00`

English (UK)	`Local: £100,000.00`
	`International: GBP100,000.00`
	`None: 100,000.00`

English (US)	`Local: $100,000.00`
	`International: USD100,000.00`
	`None: 100,000.00`

Locale	Format Type Output (Continued)
French (Belgian)	Local: 100.000,00 FB International: BEF100.000,00 None: 100.000,00
French (Canadian)	Local: 100 000,00 $ International: CAD100 000,00 None: 100 000,00
French (Standard)	Local: 100 000,00 F International: FRF100 000,00 None: 100 000,00
French (Swiss)	Local: SFr. 100’000.00 International: CHF100’000.00 None: 100’000.00
German (Austrian)	Local: öS 100.000,00 International: ATS100.000,00 None: 100.000,00
German (Standard)	Local: 100.000,00 DM International: DEM100.000,00 None: 100.000,00
German (Swiss)	Local: SFr. 100'000.00 International: CHF100'000.00 None: 100'000.00
Italian (Standard)	Local: L. 10.000.000 International: ITL10.000.000 None: 10.000.000
Italian (Swiss)	Local: SFr. 100'000.00 International: CHF100'000.00 None: 100'000.00
Norwegian (Bokmal)	Local: kr 100 000,00 International: NOK100 000,00 None: 100 000,00
Norwegian (Nynorsk)	Local: kr 100 000,00 International: NOK100 000,00 None: 100 000,00
Portuguese (Brazilian)	Local: R$100.000,00 International: BRC100.000,00 None: 100.000,00
Portuguese (Standard)	Local: R$100.000,00 International: BRC100.000,00 None: 100.000,00
Spanish (Mexican)	Local: $100,000.00 International: MXN100,000.00 None: 100,000.00

Locale	Format Type Output (Continued)
Spanish (Modern)	Local: 10.000.000 Pts International: ESP10.000.000 None: 10.000.000
Spanish (Standard)	Local: 10.000.000 Pts International: ESP10.000.000 None: 10.000.000
Swedish	Local: 100.000,00 kr International: SEK100.000,00 None: 100.000,00

Example

```
<!--- This shows LSCurrencyFormat --->
<HTML>
<HEAD>
<TITLE>LSCurrencyFormat Example</TITLE>
</HEAD>

<BODY>
<H3>LSCurrencyFormat Example</H3>

<P>LSCurrencyFormat returns a currency value using
the locale convention.  Default value is "local."

<!--- loop through a list of possible locales and
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>

</BODY>
</HTML>
```

LSDateFormat

Formats the date portion of a date/time value using the locale convention. Like DateFormat LSDateFormat returns a formatted date/time value. If no mask is specified, LSDateFormat returns a date value using the locale-specific format.

Syntax `LSDateFormat(date [, mask ])`

date
Date/time object in the period from 100 AD to 9999 AD.

mask
Set of characters that are used to show how ColdFusion should display the date:

- `d` — Day of the month as digits with no leading zero for single-digit days.
- `dd` — Day of the month as digits with a 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 with no leading zero for single-digit months.
- `mm` — Month as digits with a 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 with no leading zero for years less than 10.
- `yy` — Year as last two digits with a leading zero for years less than 10.
- `yyyy` — Year represented by four digits.
- `gg` — Period/era string. Currently ignored, but reserved for future use

Usage When passing date/time value as a string, make sure it is enclosed in quotes. Otherwise, it is interpreted as a number representation of a date/time object returning undesired results.

Examples

```
<!--- This shows LSDateFormat --->
<HTML>
<HEAD>
<TITLE>LSDateFormat Example</TITLE>
</HEAD>

<BODY>
<H3>LSDateFormat Example</H3>

<P>LSDateFormat formats the date portion of a date/time
value using the locale convention.
```

```
<!--- loop through a list of possible 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>
        #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>

</BODY>
</HTML>
```

LSIsCurrency

Checks whether a string is a locale-specific currency string. Returns TRUE if *string* is a currency string, FALSE otherwise.

Syntax **LSIsCurrency**(*string*)

string
The locale-specific currency string.

Example

```
<!--- This example shows LSIsCurrency --->
<HTML>
<HEAD>
<TITLE>LSIsCurrency Example</TITLE>
</HEAD>

<BODY bgcolor=silver>
<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" METHOD="POST">
<P>Select a locale for which you would like to check
a currency value:
<!--- check the current locale for server --->
<CFSET serverLocale = GetLocale()>
...
```

LSIsDate

Like the IsDate function, LSIsDate returns TRUE if *string* can be converted to a date/time value in the current locale, FALSE otherwise.

Syntax **LSIsDate**(*string*)

string
Any string value.

Usage Years less than 100 are interpreted as 20th century values.

Examples

```
<!--- This example shows LSIsDate --->
<HTML>
<HEAD>
<TITLE>LSIsDate Example</TITLE>
</HEAD>

<BODY bgcolor=silver>
<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" METHOD="POST">
<P>Select a locale for which you would like to check
a date value:
<!--- check the current locale for server --->
<CFSET serverLocale = GetLocale()>
...
```

LSIsNumeric

Like the IsNumeric function, LSIsNumeric returns TRUE if *string* can be converted to a number in the current locale; otherwise, FALSE.

Syntax **LSIsNumeric**(*string*)

string
Any string value.

Examples

```
<!--- This example shows LSIsNumeric --->
<HTML>
<HEAD>
<TITLE>LSIsNumeric Example</TITLE>
</HEAD>

<BODY bgcolor=silver>
<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" METHOD="POST">

<P>Select a locale for which you would like to check
a numeric value:
...
```

LSNumberFormat

Formats a number using the locale convention. If mask is omitted, the number is formatted as an integer.

Syntax **LSNumberFormat**(*number* [, *mask*])

number
The number you want to format.

mask
All LSNumberFormat mask characters apply except that ($) dollar, (,) comma, and (.) dot are mapped to their locale-specific counterparts.

LSNumberFormat Mask Characters	
Character	**Meaning**
_ (underscore)	Optional digit placeholder.
9	Optional digit placeholder. Same as _, but shows decimal places more clearly.
.	Specifies the location of a mandatory decimal point.
0	Located to the left or right of a mandatory decimal point, to force padding with zeros.
( )	Places parentheses around the mask if the number is less than 0.
+	Places + in front of positive numbers, - (minus sign) in front of negative numbers.
–	Place " " (space) in front of positive, - (minus sign) in front of negative numbers.
,	Separates thousands with commas.
L,C	Specifies left-justify or center-justify a number within the width of the mask column. L or C must appear as the first character of the mask. By default, numbers are right-justified.
$	Places a dollar sign in front of the formatted number. $ must appear as the first character of the mask.
^	Separates left from right formatting.

Note If you do not specify a sign for the mask, positive and negative numbers will not align in columns. As a result, if you expect to display both positive and negative numbers in your application, use either the space or use - (hyphen) to force a space in front of positive numbers and a minus sign in front of negative numbers.

Usage The position of codes in format masks determines where those codes will have effect. For example, if you place a dollar sign character at the far left of a format mask, ColdFusion displays a dollar sign at the very 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.

In all examples below, the numbers under the masks and the formatted output are used to clearly show the positions of characters.

Number	Mask	Result
4.37	$____.__	"$ 4.37"
4.37	_$___.__	" $4.37"
	12345678	12345678

This positioning idea can also be used to show where to place the - (minus sign) for negative numbers:

Number	Mask	Result
-4.37	-____.__	"- 4.37"
-4.37	_-___.__	" -4.37"
	12345678	12345678

There are four possible positions for any code character: far left, near left, near right, and far right. The left and right positions are determined by the side of the decimal point the code character is shown on. 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.

Whether the code is placed in the far or near position is determined by the use of _ (underscore). Most code characters will have their effect determined by which of these of fields they are located in. The following example shows how to use the field to determine exactly where to place 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) "
	12345678	12345678

Examples

```
<!--- This shows LSNumberFormat --->
<HTML>
<HEAD>
<TITLE>LSNumberFormat Example</TITLE>
</HEAD>

<BODY>
<H3>LSNumberFormat Example</H3>

<P>LSNumberFormat returns a number value using
the locale convention.

<!--- loop through a list of possible 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>
        <hr noshade>
    </CFOUTPUT>
</CFLOOP>

</BODY>
</HTML>
```

LSParseCurrency

Converts a locale-specific currency string to a number. Attempts conversion through each of the three default currency formats (none, local, international). Returns the number matching the value of *string*.

Syntax **LSParseCurrency**(*string*)

string
The locale-specific string you want to convert to a number.

Currency output

The following table shows sample currency output for some of the locales supported by ColdFusion in each of the format types: `local`, `international`, and `none`.

Currency Output by Locale	
Locale	**Format Type Output**

Dutch (Belgian)	`Local: 100.000,00 BF`
	`International: BEF100.000,00`
	`None: 100.000,00`

Dutch (Standard)	`Local: fl 100.000,00`
	`International: NLG100.000,00`
	`None: 100.000,00`

English (Australian)	`Local: $100,000.00`
	`International: AUD100,000.00`
	`None: 100,000.00`

English (Canadian)	`Local: $100,000.00`
	`International: CAD100,000.00`
	`None: 100,000.00`

English (New Zealand)	`Local: $100,000.00`
	`International: NZD100,000.00`
	`None: 100,000.00`

English (UK)	`Local: £100,000.00`
	`International: GBP100,000.00`
	`None: 100,000.00`

English (US)	`Local: $100,000.00`
	`International: USD100,000.00`
	`None: 100,000.00`

Currency Output by Locale (Continued)	
Locale	**Format Type Output**
French (Belgian)	Local: 100.000,00 FB International: BEF100.000,00 None: 100.000,00
French (Canadian)	Local: 100 000,00 $ International: CAD100 000,00 None: 100 000,00
French (Standard)	Local: 100 000,00 F International: FRF100 000,00 None: 100 000,00
French (Swiss)	Local: SFr. 100’000.00 International: CHF100’000.00 None: 100’000.00
German (Austrian)	Local: öS 100.000,00 International: ATS100.000,00 None: 100.000,00
German (Standard)	Local: 100.000,00 DM International: DEM100.000,00 None: 100.000,00
German (Swiss)	Local: SFr. 100'000.00 International: CHF100'000.00 None: 100'000.00
Italian (Standard)	Local: L. 10.000.000 International: ITL10.000.000 None: 10.000.000
Italian (Swiss)	Local: SFr. 100'000.00 International: CHF100'000.00 None: 100'000.00
Norwegian (Bokmal)	Local: kr 100 000,00 International: NOK100 000,00 None: 100 000,00
Norwegian (Nynorsk)	Local: kr 100 000,00 International: NOK100 000,00 None: 100 000,00
Portuguese (Brazilian)	Local: R$100.000,00 International: BRC100.000,00 None: 100.000,00
Portuguese (Standard)	Local: R$100.000,00 International: BRC100.000,00 None: 100.000,00

Currency Output by Locale (Continued)	
Locale	**Format Type Output**
Spanish (Mexican)	Local: $100,000.00 International: MXN100,000.00 None: 100,000.00
Spanish (Modern)	Local: 10.000.000 Pts International: ESP10.000.000 None: 10.000.000
Spanish (Standard)	Local: 10.000.000 Pts International: ESP10.000.000 None: 10.000.000
Swedish	Local: 100.000,00 kr International: SEK100.000,00 None: 100.000,00

Example

```
<!--- This example shows LSParseCurrency --->
<HTML>
<HEAD>
<TITLE>LSParseCurrency Example</TITLE>
</HEAD>

<BODY>
<H3>LSParseCurrency Example</H3>

<P>LSParseCurrency coverts a local-specific currency
string to a number.  Attempts conversion through each of
the three default currency formats.

<!--- loop through a list of possible locales and
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 Number:
#LSParseCurrency(LSCurrencyFormat(123456,"local"))#<BR>
        International: #LSCurrencyFormat(123456, "international")#<BR>
        None: #LSCurrencyFormat(123456, "none")#<BR>
        <hr noshade>
    </CFOUTPUT>
</CFLOOP>

</BODY>
</HTML>
```

LSParseDateTime

A locale-specific version of the ParseDateTime function, except that there is no option for POP date/time object parsing. Returns a date/time object.

Syntax **LSParseDateTime**(*string*)

string
String being converted to date/time object.

Examples

```
<!--- This shows LSParseDateTime --->
<HTML>
<HEAD>
<TITLE>LSParseDateTime Example</TITLE>
</HEAD>

<BODY>
<H3>LSParseDateTime Example</H3>

<P>LSParseDateTime returns a locale-specific date/time
object

<!--- loop through a list of possible 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: <BR>#LSParseDateTime("#LSDateFormat
          ("#Now()#")# #LSTimeFormat("#Now()#")#")#<BR>
        <hr noshade>
    </CFOUTPUT>
</CFLOOP>
</BODY>
</HTML>
```

LSParseNumber

Converts a locale-specific string to a number. Returns the number matching the value of *string*.

Syntax **LSParseNumber**(*string*)

string
String being converted to a number.

Examples

```
<!--- This shows LSParseNumber --->
<HTML>
<HEAD>
<TITLE>LSParseNumber Example</TITLE>
</HEAD>

<BODY>
<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 possible 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>

</BODY>
</HTML>
```

LSTimeFormat

Returns a custom-formatted time value using the locale convention.

See also LSParseDateTime.

Syntax `LSTimeFormat(time [, mask ])`

string
Any date/time value or string convertible to a time value.

mask
A set of masking characters determining the format:

- `h` — Hours with no leading zero for single-digit hours. (Uses a 12-hour clock.)
- `hh` — Hours with a leading zero for single-digit hours. (Uses a 12-hour clock.)
- `H` — Hours with no leading zero for single-digit hours. (Uses a 24-hour clock.)
- `HH` — Hours with a leading zero for single-digit hours. (Uses a 24-hour clock.)
- `m` — Minutes with no leading zero for single-digit minutes
- `mm` — Minutes with a leading zero for single-digit minutes
- `s` — Seconds with no leading zero for single-digit seconds
- `ss` — Seconds with a leading zero for single-digit seconds
- `t` — Single-character time marker string, such as A or P. Ignored by some locales.
- `tt` — Multiple-character time marker string, such as AM or PM

Usage When passing date/time value as a string, make sure it is enclosed in quotes. Otherwise, it is interpreted as a number representation of a date/time object returning undesired results.

Examples

```
<!--- This shows LSTimeFormat --->
<HTML>
<HEAD>
<TITLE>LSTimeFormat Example</TITLE>
</HEAD>

<BODY>
<H3>LSTimeFormat Example</H3>

<P>LSTimeFormat returns a time value using
the locale convention.

<!--- loop through a list of possible 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>

</BODY>
</HTML>
```

LTrim

Returns *string* with leading spaces removed.

See also RTrim and Trim.

Syntax

LTrim(*string*)

string
String being left-trimmed.

Examples

```
<!--- This example shows the use of LTrim --->
<HTML>
<HEAD>
<TITLE>
LTrim Example
</TITLE>
</HEAD>

<BODY bgcolor=silver>
<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" METHOD="POST">
<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>

</BODY>
</HTML>
```

Max

Returns the maximum, or higher, value of two numbers.

See also Min.

Syntax **Max**(*number1, number2*)

number1, number2
Any numbers.

Examples

```
<!--- This example shows the Max and Min
of two numbers --->
<HTML>
<HEAD>
<TITLE>
Max Example
</TITLE>
</HEAD>

<BODY bgcolor=silver>
<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" METHOD="POST">
<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>

</BODY>
</HTML>
```

Mid

Returns *count* characters from *string* beginning at *start* position.

See also Left, Len, and Right.

Syntax `Mid(string, start, count)`

string
Any string.

start
Starting position for count.

count
Number of characters being returned.

Examples

```
<!---  This example shows the use of Mid --->
<HTML>
<HEAD>
<TITLE>
Mid Example
</TITLE>
</HEAD>

<BODY bgcolor=silver>
<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

Returns the minimum, or smaller, value of two numbers.

See also Max.

Syntax **Min**(*number1, number2*)

number1, number2
Any numbers.

Examples

```
<!--- This example shows the Max and Min
of two numbers --->
<HTML>
<HEAD>
<TITLE>
Min Example
</TITLE>
</HEAD>

<BODY bgcolor=silver>
<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" METHOD="POST">
<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>

</BODY>
</HTML>
```

Minute

Returns the ordinal for the minute, ranging from 0 to 59.

See also DatePart, Hour, and Second.

Syntax

Minute(*date*)

date
Any date.

Usage

Years less than 100 are interpreted as 20th century values.

When passing a date/time value as a string, make sure it is enclosed in quotes. Otherwise, it is interpreted as a number representation of a date/time object, returning undesired results.

Examples

```
<!--- This example shows the use of Hour, Minute, and Second --->
<HTML>
<HEAD>
<TITLE>
Minute Example
</TITLE>
</HEAD>

<BODY bgcolor=silver>
<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>

</BODY>
</HTML>
```

Month

Returns the ordinal for the month, ranging from 1 (January) to 12 (December).

See also DatePart, MonthAsString, and Quarter.

Syntax

Month(*Date*)

date
Any date.

Usage

Years less than 100 are interpreted as 20th century values.

When passing a date/time value as a string, make sure it is enclosed in quotes. Otherwise, it is interpreted as a number representation of a date/time object, returning undesired results.

Examples

```
<!--- shows the value of the Month function --->
<HTML>
<HEAD>
<TITLE>
Month Example
</TITLE>
</HEAD>

<BODY bgcolor=silver>
<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("#yourDate#")#")>This is a leap
  year<CFELSE>This is not a leap year</CFIF>
</CFOUTPUT>
</CFIF>
...
```

MonthAsString

Returns the name of the month corresponding to *month_number*.

See also DatePart, Month, and Quarter.

Syntax **MonthAsString**(*month_number*)

month_number
An integer ranging from 1 to 12.

Examples

```
<!--- shows the value of the MonthAsString function --->
<HTML>
<HEAD>
<TITLE>
MonthAsString Example
</TITLE>
</HEAD>

<BODY bgcolor=silver>
<H3>MonthAsString 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>
...
```

Now

Returns the current date and time as a valid date time object.

See also CreateDateTime and DatePart.

Syntax **Now**()

Examples

```
<!--- This example shows Now() --->
<HTML>
<HEAD>
<TITLE>Now Example</TITLE>
</HEAD>

<BODY>
<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>

</BODY>
</HTML>
```

NumberFormat

Creates a custom-formatted number value. If no mask is specified, returns the value as an integer with a thousands separator.

See also DecimalFormat, DollarFormat, and IsNumeric.

Syntax `NumberFormat(number [, mask ])`

number
The number you want to format.

mask
Set of characters that are used to show how ColdFusion should display the number.

Mask characters	
Character	**Meaning**
_ (underscore)	Optional digit placeholder.
9	Optional digit placeholder. Same as _, but shows decimal places more clearly.
.	Specifies the location of a mandatory decimal point.
0	Located to the left or right of a mandatory decimal point, to force padding with zeros.
( )	Places parentheses around the mask if the number is less than 0.
+	Places + in front of positive numbers, - (minus sign) in front of negative numbers.
-	Place " " (space) in front of positive, - (minus sign) in front of negative numbers.
,	Separates thousands with commas.
L,C	Specifies left-justify or center-justify a number within the width of the mask column. L or C must appear as the first character of the mask. By default, numbers are right-justified.
$	Places a dollar sign in front of the formatted number. $ must appear as the first character of the mask.
^	Separates left from right formatting.

Note If you do not specify a sign for the mask, positive and negative numbers will not align in columns. As a result, if you expect to display both positive and negative numbers in your application, use either the space or use - (minus sign) to force a space in front of positive numbers and a minus sign in front of negative numbers.

Usage The position of codes in format masks determines where those codes will have effect. For example, if you place a dollar sign character at the far left of a format mask, ColdFusion displays a dollar sign at the very 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.

In all examples below, the numbers under the masks and the formatted output are used to clearly show the positions of characters.

Number	Mask	Result
4.37	$____.__	"$ 4.37"
4.37	_$___.__	" $4.37"
	12345678	12345678

This positioning idea can also be used to show where to place the - (minus sign) for negative numbers:

Number	Mask	Result
-4.37	-____.__	"- 4.37"
-4.37	_-___.__	" -4.37"
	12345678	12345678

There are four possible positions for any code character: far left, near left, near right, and far right. The left and right positions are determined by the side of the decimal point the code character is shown on. 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.

Whether the code is placed in the far or near position is determined by the use of _ (underscore). Most code characters will have their effect determined by which of these of fields they are located in. The following example shows how to use the field to determine exactly where to place 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) "
	12345678	12345678

Examples

```
<!--- This example shows the use of NumberFormat --->
<HTML>
<HEAD>
<TITLE>
NumberFormat Example
</TITLE>
</HEAD>

<BODY bgcolor=silver>
<H3>NumberFormat Example</H3>

<CFLOOP FROM=1000 TO=1020 INDEX="counter">
<CFSET CounterRoot2 = #Evaluate("#counter# * #sqr(2)#")#>

<!--- Show the result in default format, adding the comma
for the thousands place, and also 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>

</BODY>
</HTML>
```

ParagraphFormat

Returns *string* with converted single newline characters (CR/LF sequences) into spaces and double newline characters into HTML paragraph markers (<P>).

See also StripCR.

Syntax

ParagraphFormat(*string*)

string
String being converted to the HTML paragraph format.

Usage

ParagraphFormat is useful for displaying data entered into TEXTAREA fields.

Example

```
<!--- This shows ParagraphFormat --->
<HTML>
<HEAD>
<TITLE>
ParagraphFormat Example
</TITLE>
</HEAD>

<BODY bgcolor=silver>
<H3>ParagraphFormat Example</H3>

<P>Enter some 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" METHOD="POST">
<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>

</BODY>
</HTML>
```

ParameterExists

Returns True if the specified parameter has been passed to the current template or has been already created during execution of the current template. Otherwise returns NO.

This function is provided for backward compatibility with previous versions of ColdFusion. You should use the function IsDefined instead.

See also GetClientVariablesList and IsDefined.

Syntax

ParameterExists(*parameter*)

parameter
Any syntactically valid parameter name.

Example

```
<!--- This example shows ParameterExists --->
<HTML>
<HEAD>
<TITLE>
ParameterExists Example
</TITLE>
</HEAD>

<BODY bgcolor=silver>
<H3>ParameterExists Example</H3>

<CFIF ParameterExists(form.myString)>
<P>Using ParameterExists, we have shown that the form
field "myString" now exists.  However, this function is
provided for backward compatibility and
the function IsDefined should be used instead, as below.
</CFIF>

<CFIF IsDefined("form.myString")>
<P>Because the variable form.myString has been defined, we
can now show its contents.  This construction allows us
to place a form and its resulting action template in the same
template, while using IsDefined to control the
flow of template execution.
<P>The value of "form.myString" is
  <B><I><CFOUTPUT>#form.myString#</CFOUTPUT></I></B>
<CFELSE>
<P>During the first time through this template, the variable
"form.myString" has not yet been defined, so we
are not attempting to evaluate it.
</CFIF>
...
```

ParseDateTime

Returns a date/time object from a string.

See also IsDate and IsNumericDate.

Syntax

ParseDateTime(*string1* [, *string2*])

string1
String being converted to date/time object.

string2
POP or STANDARD. POP takes the date/time object passed from a POP mail server and converts it to GMT (Greenwich Mean Time). Enter STANDARD or leave empty for no conversion.

Usage

ParseDateTime is similar to CreateDateTime except that it takes a string instead of specifically enumerated date/time values.

Both ParseDateTime and CreateDateTime are provided primarily to increase the readability of code in compound expressions.

Years from 0 to 29 are interpreted as 21st century values. Years 30 to 99 are interpreted as 20th century values.

When passing a date/time value as a string, make sure it is enclosed in quotes. Otherwise, it is interpreted as a number representation of a date/time object, returning undesired results.

Examples

```
<!--- This example shows the use of ParseDateTime--->
<HTML>
<HEAD>
<TITLE>
ParseDateTime Example
</TITLE>
</HEAD>

<BODY bgcolor=silver>
<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

Returns the number 3.14159265358979, the mathematical constant π, accurate to 15 digits.

See also Atn, Cos, Sin, and Tan.

Syntax

```
Pi()
```

Examples

```
<!--- This shows the use of Pi --->
<HTML>
<HEAD>
<TITLE>
Pi Example
</TITLE>
</HEAD>

<BODY>
<H3>Pi Example</H3>

Returns the number
<CFOUTPUT>
#NumberFormat("#Pi()#", '_.________________')#
</CFOUTPUT>, the
mathematical constant pi, accurate to 15 digits.

</BODY>
</HTML>
```

PreserveSingleQuotes

Prevents ColdFusion from automatically "escaping" single-quotes contained in *variable*.

Syntax **PreserveSingleQuotes**(*variable*)

variable
Variable containing the string for which single quotes are preserved.

Usage PreserveSingleQuotes is useful in SQL statements.

Example

```
<!--- This example shows the use of PreserveSingleQuotes --->
<HTML>
<HEAD>
<TITLE>
PreserveSingleQuotes Example
</TITLE>
</HEAD>

<BODY>
<H3>PreserveSingleQuotes Example</H3>

<P>This is a useful function for creating lists of
information to return from a query.  In the following
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>

</BODY>
</HTML>
```

Quarter

Returns the number of the quarter, an integer ranging from 1 to 4.

See also DatePart and Month.

Syntax

Quarter(*date*)

date
Any date.

Usage

Years less than 100 are interpreted as 20th century values.

When passing a date/time value as a string, make sure it is enclosed in quotes. Otherwise, it is interpreted as a number representation of a date/time object, returning undesired results.

Examples

```
<!--- This example shows the use of Quarter --->
<HTML>
<HEAD>
<TITLE>
Quarter Example
</TITLE>
</HEAD>

<BODY bgcolor=silver>
<H3>Quarter Example</H3>

Today, <CFOUTPUT>#DateFormat("#Now()#")#</CFOUTPUT>,
is in Quarter <CFOUTPUT>#Quarter("#Now()#")#</CFOUTPUT>.

</BODY>
</HTML>
```

QueryAddRow

Adds a specified number of empty rows to the specified query. Returns the total number of rows in the query that you are adding rows to.

See also QueryNew and QuerySetCell.

Syntax

QueryAddRow(*query* [, *number*])

query
Name of the query already executed.

number
Number of rows to add to the query. Default is 1.

Example

```
<!--- This example shows the use of QueryAddRow and QuerySetCell --->
<HTML>
<HEAD>
<TITLE>
QueryAddRow Example
</TITLE>
</HEAD>

<BODY>
<H3>QueryAddRow Example</H3>

<!--- start by making a query --->
<CFQUERY NAME="GetCourses" DATASOURCE="cfsnippets">
SELECT Course_ID, Number, 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, "Course_ID", CountVar)>
 <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>
...
```

QueryNew

Returns an empty query with a set of columns.

See also QueryAddRow and QuerySetCell.

Syntax

QueryNew(*columnlist*)

columnlist
Comma-separated list of columns you want to add to the new query.

Example

```
<!--- This example shows the use of QueryNew --->
<HTML>
<HEAD>
<TITLE>
QueryNew Example
</TITLE>
</HEAD>

<BODY>
<H3>QueryNew Example</H3>

<P>We will construct a new query with two rows:
<!--- make a new query --->
<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", "588-1444", 2)>

<!--- output the query --->
<CFOUTPUT query="myQuery">
<PRE>#name##address##phone#</PRE>
</CFOUTPUT>
To get any item in the query, we can output it individually
<CFOUTPUT>
<P>Jane's phone number: #MyQuery.phone[2]#
</CFOUTPUT>
</BODY>
</HTML>
```

QuerySetCell

Sets the cell in a specified column to a specified value. If no row number is specified, the cell on the last row will be set. Returns TRUE.

See also QueryAddRow and QueryNew.

Syntax `QuerySetCell(query, column_name, value [, row_number ])`

query
Name of the query already executed.

column_name
Name of the column in the query.

value
Value to set in the specified cell.

row_number
Number of the row. Defaults to last row.

Example

```
<!--- This example shows the use of QueryAddRow and QuerySetCell --->
<HTML>
<HEAD>
<TITLE>
QuerySetCell Example
</TITLE>
</HEAD>

<BODY>
<H3>QuerySetCell Example</H3>

<!--- start by making a query --->
<CFQUERY NAME="GetCourses" DATASOURCE="cfsnippets">
SELECT  Course_ID, Number, 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, "Course_ID", CountVar)>
 <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##Number##Descript#</PRE>
</CFOUTPUT>

</BODY>
</HTML>
```

QuotedValueList

Returns a comma-separated list of the values of each record returned from a previously executed query. Each value in the list is enclosed in single quotes.

See also ValueList.

Syntax

QuotedValueList(*query.column* [, *delimiter*])

query.column
Name of an already executed query and column. Separate query name and column name with a period (.).

delimiter
A string delimiter to separate column data.

Example

```
<!--- This example shows the use of QuotedValueList --->
<HTML>
<HEAD>
<TITLE>
QuotedValueList Example
</TITLE>
</HEAD>

<BODY>
<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>
</BODY>
</HTML>
```

Rand

Returns a random decimal number in the range 0 to 1.

See also Randomize and RandRange.

Syntax **Rand**()

Usage To ensure even greater randomness, call Randomize before calling Rand.

Example

```
<!--- This example shows the use of Rand() --->
<HTML>
<HEAD>
<TITLE>
Rand Example
</TITLE>
</HEAD>

<BODY bgcolor=silver>
<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>

</BODY>
</HTML>
```

Randomize

Seeds the random number generator in ColdFusion with the integer part of a *number*. By seeding the random number generator with a variable value, you help to ensure that the Rand function generates highly random numbers.

See also Rand and RandRange.

Syntax

Randomize(*number*)

number
Any number.

Usage

Call this function before calling Rand. Although this function returns a decimal number in the range 0 to 1, it is not a random number and you should not use it.

Examples

```
<!--- This example shows the use of Randomize --->
<HTML>
<HEAD>
<TITLE>
Randomize Example
</TITLE>
</HEAD>

<BODY bgcolor=silver>
<H3>Randomize Example</H3>

<P>Call Randomize to seed the random number generator.
This helps to ensure the randomness of numbers generated
by the Rand function.
<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" METHOD="POST">
<P>Enter a number to seed the randomizer:
<INPUT TYPE="Text" NAME="MyRandomInt">
<P><INPUT TYPE="Submit" NAME="">
</FORM>

</BODY>
</HTML>
```

RandRange

Returns a random integer between two specified numbers. Note that requests for random integers greater than 100,000,000 will result in non-random behavior. This restriction prevents overflow during internal computations.

See also Rand and Randomize.

Syntax

RandRange(*number1, number2*)

number1, number2
Integer numbers less than 100,000,000.

Examples

```
<!--- This example shows the use of RandRange --->
<HTML>
<HEAD>
<TITLE>
RandRange Example
</TITLE>
</HEAD>

<BODY bgcolor=silver>
<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" METHOD="POST">
<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>

</BODY>
</HTML>
```

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. This search is case-sensitive.

See also Find and REReplace.

Syntax

REFind(*reg_expression*, *string*, *start*)

reg_expression
Regular expression used for search. This regular expression can include POSIX-specified character classes (for example, [:alpha:], [:digit:], [:upper:], and [:lower:]).

string
String being searched.

start
Starting position for the search.

Examples

```
<!--- This example shows the use of REFind --->
<HTML>
<HEAD>
<TITLE>
REFind Example
</TITLE>
</HEAD>

<BODY>
<H3>REFind Example</H3>

<P>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>REFind("a+c+", "abcaaccdd"):<CFOUTPUT>#REFind("a+c+", "abcaaccdd")#
</CFOUTPUT>
<P>REFind("a+c*", "abcaaccdd"):<CFOUTPUT>#REFind("a+c*", "abcaaccdd")#
</CFOUTPUT>
<P>REFind("[\?&]rep=", "report.cfm?rep=1234&u=5"):
  <CFOUTPUT>#REFind("[\?&]rep=", "report.cfm?rep=1234&u=5")#</CFOUTPUT>

</BODY>
</HTML>
```

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. The search is case-insensitive.

See also Find, FindNoCase, REReplace, and REReplaceNoCase.

Syntax

```
REFindNoCase(reg_expression, string [, start ]
   [, returnsubexpressions ] )
```

reg_expression
Regular expression used for search. This regular expression can include POSIX-specified character classes (for example, [:alpha:], [:digit:], [:upper:], and [:lower:]).

string
String being searched.

start
Starting position for the search. Default is 1.

returnsubexpressions
Boolean indicating whether subexpressions are returned. If you set this parameter to TRUE, the function returns a CFML structure with two arrays containing the positions and lengths of the matched sub-expressions, if any. You can retrieve from the structure using the keys "pos" and "len". If there are no occurrences of the regular expression, the "pos" and the "len" arrays each contain 1 (value = 0).

Examples

```
<!--- This example shows the use of REFindNoCase --->
<HTML>
<HEAD>
<TITLE>
REFindNoCase Example
</TITLE>
</HEAD>
<BODY>
<H3>REFindNoCase Example</H3>
<P>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. REFindNoCase is case-insensitive.
<P>REFindNoCase("a+c+", "ABCAACCDD"):
  <CFOUTPUT>#REFindNoCase("a+c+", "ABCAACCDD")#</CFOUTPUT>
<P>REFindNoCase("a+c*", "ABCAACCDD"):
  <CFOUTPUT>#REFindNoCase("a+c*", "ABCAACCDD")#</CFOUTPUT>
<P>REFindNoCase("[\?&]rep=", "report.cfm?rep=1234&u=5"):
  <CFOUTPUT>#REFindNoCase("[\?&]rep=", "report.cfm?rep=1234&u=5")#
    </CFOUTPUT>
...
```

RemoveChars

Returns *string* with *count* characters removed from the specified starting position. Return 0 if no characters are found.

See also Insert and Len.

Syntax

RemoveChars(*string*, *start*, *count*)

string
Any string.

start
Starting position for the search.

count
Number of characters being removed.

Examples

```
<!--- This example shows the use of RemoveChars --->
<HTML>
<HEAD>
<TITLE>
RemoveChars Example
</TITLE>
</HEAD>

<BODY bgcolor=silver>
<H3>RemoveChars Example</H3>

Returns a string with <I>count</I> characters
removed from the specified starting 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

Returns a string created from *string* being repeated a specified number of times.

See also CJustify, LJustify, and RJustify.

Syntax

RepeatString(*string*, *count*)

string
String being repeated.

count
Number of repeats.

Examples

```
<!--- This example shows RepeatString --->
<HTML>
<HEAD>
<TITLE>
RepeatString Example
</TITLE>
</HEAD>

<BODY bgcolor=silver>
<H3>RepeatString Example</H3>

<P>RepeatString returns a string created from
<I>string</I> being 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>

</BODY>
</HTML>
```

Replace

Returns *string* with occurrences of *substring1* being replaced with *substring2* in the specified scope.

See also Find, ReplaceNoCase, ReplaceList, and REReplace.

Syntax

Replace(*string*, *substring1*, *substring2* [, *scope*])

string
Any string.

substring1
String to be replaced.

substring2
String that should replace occurrences of substring1.

scope
Defines how to complete the replace operation:

- ONE — Replace only the first occurrence (default).
- ALL — Replace all occurrences.

Examples

```
<!--- This example shows the use of Replace --->
<HTML>
<HEAD>
<TITLE>
Replace Example
</TITLE>
</HEAD>

<BODY bgcolor=silver>
<H3>Replace Example</H3>

<P>The Replace function returns <I>string</I> with
<I>substring1</I> being 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

Returns *string* with all occurrences of the elements from the specified comma-delimited list being replaced with their corresponding elements from another comma-delimited list. The search is case sensitive.

See also Find, Replace, and REReplace.

Syntax

ReplaceList(*string*, *list1*, *list2*)

string
Any string.

list1
Comma-delimited list of substrings to be replaced.

list2
Comma-delimited list of replace substrings.

Usage

Note that the list of substrings to be replaced is processed one after another. In this way you may experience recursive replacement if one of your *list1* elements is contained in *list2* elements. The second example listed below demonstrates such replacement.

Examples

```
<!--- This example shows the use of Replacelist --->
<HTML>
<HEAD>
<TITLE>
Replacelist Example
</TITLE>
</HEAD>

<BODY bgcolor=silver>
<H3>Replacelist Example</H3>

<P>The Replacelist function returns <I>string</I> with
<I>substringlist1</I> (e.g. "a,b") being 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>

</BODY>
</HTML>
```

ReplaceNoCase

Returns *string* with occurrences of *substring1* being replaced regardless of case matching with *substring2* in the specified scope.

See also Find, Replace, ReplaceList, and REReplace.

Syntax `ReplaceNoCase(string, substring1, substring2 [, scope ])`

string
Any string.

substring1
String to be replaced.

substring2
String that should replace occurrences of substring1.

scope
Defines how to complete the replace operation:

- `ONE` — Replace only the first occurrence (default).
- `ALL` — Replace all occurrences.

Examples

```
<!--- This example shows the use of ReplaceNoCase --->
<HTML>
<HEAD>
<TITLE>
ReplaceNoCase Example
</TITLE>
</HEAD>

<BODY bgcolor=silver>
<H3>ReplaceNoCase Example</H3>

<P>The ReplaceNoCase function returns <I>string</I> with
<I>substring1</I> being 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

Returns *string* with a regular expression being replaced with *substring* in the specified scope. This is a case-sensitive search.

See also REFind, Replace, ReplaceList, and REReplaceNoCase.

Syntax

REReplace(*string*, *reg_expression*, *substring* [, *scope*])

string
Any string.

reg_expression
Regular expression to be replaced. This regular expression can include POSIX-specified character classes (for example, [:alpha:], [:digit:], [:upper:], and [:lower:]).

substring
String replacing reg_expression.

scope
Defines how to complete the replace operation:

- ONE — Replace only the first occurrence (default).
- ALL — Replace all occurrences.

Examples

```
<!--- This example shows the use of REReplace --->
<HTML>
<HEAD>
<TITLE>
REReplace Example
</TITLE>
</HEAD>

<BODY bgcolor=silver>
<H3>REReplace Example</H3>
<P>The REReplace function returns <i>string</i> with a regular expression
being replaced with <i>substring</i> in the specified scope.
This is a 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>
</BODY>
</HTML>
```

REReplaceNoCase

Returns *string* with a regular expression being replaced with *substring* in the specified scope. The search is case-insensitive.

See also REFind, REFindNoCase, Replace, and ReplaceList.

Syntax

REReplaceNoCase(*string*, *reg_expression*, *substring* [, *scope*])

string
Any string.

reg_expression
Regular expression to be replaced. This regular expression can include POSIX-specified character classes (for example, [:alpha:], [:digit:], [:upper:], and [:lower:]).

substring
String replacing *reg_expression*.

scope
Defines how to complete the replace operation:

- ONE — Replace only the first occurrence (default).
- ALL — Replace all occurrences.

Examples

```
<!--- This example shows the use of REReplaceNoCase --->
<HTML>
<HEAD>
<TITLE>
REReplaceNoCase Example
</TITLE>
</HEAD>
<BODY bgcolor=silver>
<H3>REReplaceNoCase Example</H3>
<P>The REReplaceNoCase function returns <i>string</i> with a regular
expression being 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>
</BODY>
</HTML>
```

Reverse

Returns *string* with reversed order of characters.

See also Left, Mid, and Right.

Syntax

Reverse(*string*)

string
String being reversed.

Examples

```
<!--- This example shows the use of Reverse --->
<HTML>
<HEAD>
<TITLE>
Reverse Example
</TITLE>
</HEAD>

<BODY bgcolor=silver>
<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" METHOD="POST">
<P>Enter a string to be reversed:
<INPUT TYPE="Text" NAME="MyString">
<P><INPUT TYPE="Submit" NAME="">
</FORM>

</BODY>
</HTML>
```

Right

Returns the rightmost *count* characters of a string.

See also Left, Len, and Mid.

Syntax

Right(*string*, *count*)

string
String from which the rightmost characters are retrieved.

count
Integer indicating how many characters to return.

Examples

```
<!---  This example shows the use of Right --->
<HTML>
<HEAD>
<TITLE>
Right Example
</TITLE>
</HEAD>

<BODY bgcolor=silver>
<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

Returns right-justified *string* in the specified field length.

See also CJustify and LJustify.

Syntax **RJustify**(*string*, *length*)

string
String to be right-justified.

length
Length of field.

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 bgcolor=silver>
<H3>RJustify Function</H3>

<P>Enter a string, and it will be right justified within
the sample field

<FORM ACTION="rjustify.cfm" METHOD="POST">
<P><INPUT TYPE="Text" VALUE="<CFOUTPUT>#jString#</CFOUTPUT>" SIZE=35
NAME="justifyString">

<P><INPUT TYPE="Submit" NAME=""> <INPUT TYPE="RESET">
</FORM>

</BODY>
</HTML>
```

Round

Rounds a number to the closest integer.

See also Ceiling, Fix, and Int.

Syntax

Round(*number*)

number
Number being rounded.

Examples

```
<!--- This example shows the use of Round --->
<HTML>
<HEAD>
<TITLE>
Round Example
</TITLE>
</HEAD>

<ODY>

<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>

</BODY>
</HTML>
```

RTrim

Returns *string* with removed trailing spaces.

See also LTrim and Trim.

Syntax

RTrim(*string*)

string
String being right-trimmed.

Examples

```
<!--- This example shows the use of RTrim --->
<HTML>
<HEAD>
<TITLE>
RTrim Example
</TITLE>
</HEAD>

<BODY bgcolor=silver>
<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>Type in some text, and it will be modified
by Rtrim to remove leading spaces from the right
<P><INPUT TYPE="Text" NAME="myText" VALUE="TEST     ">

<P><INPUT TYPE="Submit" NAME="">
</FORM>

</BODY>
</HTML>
```

Second

For a date/time value, returns the ordinal for the second, an integer from 0 to 59.

See also DatePart, Hour, and Minute.

Syntax

Second(*date*)

date
Any date.

Usage

When passing a date/time value as a string, make sure it is enclosed in quotes. Otherwise, it is interpreted as a number representation of a date/time object, returning undesired results.

Examples

```
<!--- This example shows the use of Hour, Minute, and Second --->
<HTML>
<HEAD>
<TITLE>
Second Example
</TITLE>
</HEAD>

<BODY bgcolor=silver>
<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>

</BODY>
</HTML>
```

SetLocale

Sets the locale to the specified new locale for the current session.

Note: SetLocale returns the old locale in case it needs to be restored.

See also GetLocale.

Syntax `SetLocale(new_locale)`

new_locale
The name of the locale you want to set.

Locale support

ColdFusion can be expected to support the following locales in a default Windows NT installation:

Locales Supported by ColdFusion		
Dutch (Belgian)	French (Canadian)	Norwegian (Bokmal)
Dutch (Standard)	French (Standard)	Norwegian (Nynorsk)
English (Australian)	French (Swiss)	Portuguese (Brazilian)
English (Canadian)	German (Austrian)	Portuguese (Standard)
English (New Zealand)	German (Standard)	Spanish (Mexican)
English (UK)	German (Swiss)	Spanish (Modern)
English (US)	Italian (Standard)	Spanish (Standard)
French (Belgian)	Italian (Swiss)	Swedish

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. GetLocale( ) will return an entry from that list. SetLocale will fail if called with a locale name not on that list.

Example

```
<!--- This example shows SetLocale --->
<HTML>
<HEAD>
<TITLE>SetLocale Example</TITLE>
</HEAD>

<BODY>
```

```
<H3>SetLocale Example</H3>

<P>SetLocale sets the locale to the specified new locale
for the current session.

<P>A locale is an encapsulation of the set of attributes
that govern the display and formatting of international
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>

</BODY>
</HTML>
```

SetVariable

The function sets the variable specified by *name* to *value* and returns the new value of the variable.

See also DeleteClientVariable and GetClientVariablesList.

Syntax

SetVariable(*name*, *value*)

name
Valid variable name.

value
String or number assigned to the variable.

Usage

When setting client variables, it is required that the client variable exists prior to the using of this function and the ClientManagement attribute of CFAPPLICATION tag has been set to "Yes" for this template.

Examples

```
<!--- This example shows SetVariable --->
<HTML>
<HEAD>
<TITLE>
SetVariable Example
</TITLE>
</HEAD>

<BODY bgcolor=silver>
<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

Determines the sign of a number. Returns 1 if *number* is positive; 0 if *number* is 0; and -1 if *number* is negative.

See also Abs.

Syntax

Sgn(*number*)

number
Any number.

Examples

```
<!--- This example shows the use of Sgn --->
<HTML>
<HEAD>
<TITLE>Sgn Example</TITLE>
</HEAD>

<BODY>
<H3>Sgn Example</H3>

<P>Sgn determines the sign of a number.  Returns
1 if number is positive; 0 if number is 0; and -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>

</BODY>
</HTML>
```

Sin

Returns the sine of the given angle.

See also Atn, Cos, Pi, and Tan.

Syntax

Sin(*number*)

number
Angle in radians for which you want the sine. If the angle is in degrees, multiply it by PI()/180 to convert it to radians.

Examples

```
<!--- This snippet shows how to use Sin --->
<HTML>
<HEAD>
<TITLE>
Sin Example
</TITLE>
</HEAD>

<BODY bgcolor=silver>
<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 Sin value</H4>
    </CFIF>
</CFIF>

<FORM ACTION="sin.cfm" METHOD="POST">
<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>

</BODY>
</HTML>
```

SpanExcluding

Returns all characters from *string* from its beginning until it reaches a character from the *set* of characters. The search is case-sensitive.

See also GetToken and SpanIncluding.

Syntax

SpanExcluding(*string*, *set*)

string
Any string.

set
String containing one or more characters being sought.

Examples

```
<!--- Displays SpanExcluding --->
<HTML>
<HEAD>
<TITLE>
SpanExcluding Example
</TITLE>
</HEAD>

<BODY bgcolor=silver>
<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 until it reaches a character from the set
of characters.  The search is case-sensitive.

<FORM ACTION="spanexcluding.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">
<BR><INPUT TYPE="Submit" NAME="">
</FORM>

</BODY>
</HTML>
```

SpanIncluding

Returns all characters from *string* from its beginning until it reaches a character that is not included in the specified *set* of characters. The search is case-sensitive.

See also GetToken and SpanExcluding.

Syntax

SpanIncluding(*string*, *set*)

string
Any string.

set
String containing one or more characters being sought.

Examples

```
<!--- Displays SpanIncluding --->
<HTML>
<HEAD>
<TITLE>
SpanIncluding Example
</TITLE>
</HEAD>

<BODY bgcolor=silver>
<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 up until all of the characters in the set have been
  found is:
<CFOUTPUT>#SpanIncluding("#form.myString#", "#form.mySet#")#</CFOUTPUT>
</CFIF>

<P>Returns all characters from string from
beginning until it all characters from the set
of characters have been found.  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>

</BODY>
</HTML>
```

Sqr

Returns a positive square root.

See also Abs.

Syntax **Sqr**(*number*)

number
Number for which you want the square root.

Usage *Number* must be greater than or equal to 0.

Examples

```
<!--- This example shows Sqr --->
<HTML>
<HEAD>
<TITLE>
Sqr Example
</TITLE>
</HEAD>

<BODY>
<H3>Sqr Example</H3>

<P>Returns a positive square root for a 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>

</BODY>
</HTML>
```

StripCR

Returns *string* with all carriage return characters removed.

See also ParagraphFormat.

Syntax **StripCR**(*string*)

string
String being formatted.

Usage Function StripCR is useful for preformatted HTML display of data (PRE) entered into TEXTAREA fields

Example

```
<!--- This example shows StripCR --->
<HTML>
<HEAD>
<TITLE>
StripCR Example
</TITLE>
</HEAD>

<BODY bgcolor=silver>
<H3>StripCR Example</H3>

<P>Function StripCR is useful for preformatted HTML
display of data (PRE) entered into TEXTAREA fields.

<CFIF IsDefined("form.myTextArea")>

<PRE>
<CFOUTPUT>#StripCR(form.myTextArea)#</CFOUTPUT>
</PRE>
</CFIF>
<!--- use #Chr(10)##Chr(13)# to simulate a line feed/carriage
return combination; i.e, a return --->
<FORM ACTION="stripcr.cfm" METHOD="POST">
<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>

</BODY>
</HTML>
```

StructClear

Removes all data from the specified structure. Always returns Yes.

See also StructDelete, StructFind, StructInsert, StructIsEmpty, StructKeyExists, StructCount, StructNew, and StructUpdate.

Syntax **StructClear**(*structure*)

structure
Structure to be cleared.

Example

```
<!--- This example shows how to use the StructClear
      function. It calls the CF_ADDEMPLOYEE custom tag,
      which uses the addemployee.cfm file. --->
<html>
<head>
<title>Add New Employees</title>
</head>
<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

Returns a new structure with the all the keys and values of the specified structure.

See also StructClear, StructDelete, StructFind, StructInsert, StructIsEmpty, StructKeyExists, StructNew, and StructUpdate.

Syntax **StructCopy**(*structure*)

structure
Structure to be copied.

Usage This function throws an exception if *structure* does not exist.

Example

```
<!--- This view-only example illustrates usage
      of StructCopy. --->
<P>This file is similar to addemployee.cfm, which is called
by StructNew, StructClear, and StructDelete.
<!---
<cfswitch expression="#ThisTag.ExecutionMode#">
   <cfcase value="start">
      <CFIF StructIsEmpty(attributes.EMPINFO)>
        <CFOUTPUT>Error. No employee data was passed.</cfoutput>
          <CFEXIT METHOD="ExitTag">
     <cfelse>
        tempStruct=StructCopy(EMPINFO)
        <CFQUERY NAME="AddEmployee" DATASOURCE="cfsnippets">
          INSERT INTO Employees
              (FirstName, LastName, Email, Phone, Department)
            VALUES
              <CFOUTPUT>
                (
                  '#StructFind(attributes.tempStruct, "firstname")#' ,
                  '#StructFind(attributes.tempStruct, "lastname")#' ,
                  '#StructFind(attributes.tempStruct, "email")#' ,
                  '#StructFind(attributes.tempStruct, "phone")#' ,
                  '#StructFind(attributes.tempStruct, "department")#'
                 )
              </cfoutput>
          </cfquery>
        </cfif>
      <CFOUTPUT><HR>Employee Add Complete
         <P>#StructCount(attributes.tempStruct)# columns added.
          </cfoutput>
   </cfcase>
</cfswitch> --->
```

StructCount

Returns the number of keys in the specified structure.

See also StructClear, StructDelete, StructFind, StructInsert, StructIsEmpty, StructKeyExists, StructNew, and StructUpdate.

Syntax

StructCount(*structure*)

structure
Structure to be accessed.

Usage

This function throws an exception if *structure* does not exist.

Example

```
<!--- This view-only example illustrates usage
      of StructCount. --->
<P>This file is similar to addemployee.cfm, which is called
by StructNew, StructClear, and StructDelete. To test this file,
copy the StructCount function to the 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

Removes the specified item from the specified structure.

See also StructClear, StructFind, StructInsert, StructIsEmpty, StructKeyExists, StructCount, StructNew, and StructUpdate.

Syntax **StructDelete**(*structure*, *key* [, *indicatenotexisting*])

structure
Structure containing the item to be removed.

key
Item to be removed.

indicatenotexisting
Indicates whether the function returns FALSE if *key* does not exist. The default is FALSE, which means that the function returns Yes regardless of whether *key* exists. If you specify TRUE for this parameter, the function returns Yes if *key* exists and No if it does not.

Example

```
<!--- This example shows how to use the StructDelete
      function. It calls the CF_ADDEMPLOYEE custom tag,
      which uses the addemployee.cfm file. --->
<html>
<head>
<title>Add New Employees</title>
</head>
<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>StructDelete(employee);</CFSCRIPT>
 </cfif>
<hr>
<FORM action="structdelete.cfm" method="post">
<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>

</body>
</html>
```

StructFind

Returns the value associated with the specified key in the specified structure.

See also StructClear, StructDelete, StructInsert, StructIsEmpty, StructKeyExists, StructCount, StructNew, and StructUpdate.

Syntax **StructFind**(*structure*, *key*)

structure
Structure containing the value to be returned.

key
Key whose value is returned.

Usage This function throws an exception if *structure* does not exist.

Example

```
<!--- This view-only example illustrates usage of StructFind. --->
<P>This file is identical 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 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> --->
```

StructInsert

Inserts the specified key-value pair into the specified structure. Returns Yes if the insert was successful and No if an error occurs.

See also StructClear, StructDelete, StructFind, StructIsEmpty, StructKeyExists, StructCount, StructNew, and StructUpdate.

Syntax `StructInsert(structure, key, value [, allowoverwrite ])`

structure
Structure to contain the new key-value pair.

key
Key that contains the inserted value.

value
Value to be added.

allowoverwrite
Optionally indicates whether to allow overwriting an existing key. The default is FALSE.

Usage This function throws an exception if *structure* does not exist or if *key* exists and allowoverwrite is set to FALSE.

Example

```
<!--- This example shows how to use the StructInsert
      function. It calls the CF_ADDEMPLOYEE custom tag,
      which uses the addemployee.cfm file. --->
<html>
<head>
<title>Add New Employees</title>
</head>

<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>

  <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" method="post">
<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>

</body>
</html>
```

StructIsEmpty

Indicates whether the specified structure contains data. Returns TRUE if *structure* is empty and FALSE if it contains data.

See also StructClear, StructDelete, StructFind, StructInsert, StructKeyExists, StructCount, StructNew, and StructUpdate.

Syntax

StructIsEmpty(*structure*)

structure
Structure to be tested.

Usage

This function throws an exception if *structure* does not exist.

Example

```
<!--- This view-only example illustrates usage of StructIsEmpty. --->
<P>This file is identical 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 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> --->
```

StructKeyExists

Returns TRUE if the specified key is in the specified structure and FALSE if it is not.

See also StructClear, StructDelete, StructFind, StructInsert, StructIsEmpty, StructCount, StructNew, and StructUpdate.

Syntax

StructKeyExists(*structure*, *key*)

structure
Structure to be tested.

key
Key to be tested.

Usage

This function throws an exception if *structure* does not exist.

Example

```
<!--- This view-only example illustrates usage of StructKeyExists. --->
<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 NOT StructKeyExists(attributes.EMPINFO, "department")>
        <cfscript>StructUpdate(attributes.EMPINFO, "department",
            "Unassigned")</cfscript>
        <CFEXIT METHOD="ExitTag">
      <cfelse>
          ...
```

StructNew

Returns a new structure.

See also StructClear, StructDelete, StructFind, StructInsert, StructIsEmpty, StructKeyExists, StructCount, and StructUpdate.

Syntax `StructNew()`

Example

```
<!--- This example shows how to use the StructNew function. It calls
the CF_ADDEMPLOYEE custom tag, which uses the addemployee.cfm file. --->
<html>
<head>
<title>Add New Employees</title>
</head>

<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>

  <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>
...
```

StructUpdate

Updates the specified key with the specified value. Returns Yes if the function is successful and throws an exception if an error occurs.

See also StructClear, StructDelete, StructFind, StructInsert, StructIsEmpty, StructKeyExists, StructCount, and StructNew.

Syntax

StructUpdate(*structure*, *key*, *value*)

structure
Structure to be updated.

key
Key whose value is updated.

value
New value.

Usage

This function throws an exception if *structure* does not exist.

Example

```
<!--- This view-only example illustrates usage
      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

Returns the tangent of a given angle.

See also Atn, Cos, Sin, and Pi.

Syntax

Tan(*number*)

number
Angle in radians for which you want the tangent. If the angle is in degrees, multiply it by PI()/180 to convert it to radians.

Examples

```
<!--- This example shows Tan --->
<HTML>
<HEAD>
<TITLE>
Tan Example
</TITLE>
</HEAD>

<BODY>
<H3>Tan Example</H3>

<P>Returns the tangent of a given angle.

<P>Tan(1): <CFOUTPUT>#Tan(1)#</CFOUTPUT>
<P>Tan(Pi()/4): <CFOUTPUT>#Tan(Pi()/4)#</CFOUTPUT>

</BODY>
</HTML>
```

TimeFormat

Returns a custom-formatted time value. If no mask is specified, the TimeFormat function returns time value using the *hh:mm tt* format.

See also CreateTime, Now, and ParseDateTime.

Syntax `TimeFormat(time [, mask ])`

time
Any date/time value or string convertible to a time value.

mask
A set of masking characters determining the format:

- `h` — Hours with no leading zero for single-digit hours. (Uses a 12-hour clock.)
- `hh` — Hours with a leading zero for single-digit hours. (Uses a 12-hour clock.)
- `H` — Hours with no leading zero for single-digit hours. (Uses a 24-hour clock.)
- `HH` — Hours with a leading zero for single-digit hours. (Uses a 24-hour clock.)
- `m` — Minutes with no leading zero for single-digit minutes
- `mm` — Minutes with a leading zero for single-digit minutes
- `s` — Seconds with no leading zero for single-digit seconds
- `ss` — Seconds with a leading zero for single-digit seconds
- `t` — Single-character time marker string, such as A or P
- `tt` — Multiple-character time marker string, such as AM or PM

Usage When passing a date/time value as a string, make sure it is enclosed in quotes. Otherwise, it is interpreted as a number representation of a date/time object, returning undesired results.

Examples

```
<!--- This example shows the various types of output
possible with TimeFormat --->
<HTML>
<HEAD>
<TITLE>
TimeFormat Example
</TITLE>
</HEAD>
<CFSET todayDate = #Now()#>

<BODY>
<H3>TimeFormat Example</H3>

<P>Today's date is <CFOUTPUT>#todayDate#</CFOUTPUT>.
```

```
<P>Using Timeformat, we can display that date/time value
in a number of 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>
</HTML>
```

Trim

Returns *string* with both leading and trailing spaces removed.

See also LTrim and RTrim.

Syntax

Trim(*string*)

string
String being trimmed.

Examples

```
<!--- This example shows the use of Trim --->
<HTML>
<HEAD>
<TITLE>
Trim Example
</TITLE>
</HEAD>

<BODY bgcolor=silver>
<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" METHOD="POST">
<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>

</BODY>
</HTML>
```

UCase

Returns *string* converted to uppercase.

See also LCase.

Syntax

UCase(*string*)

string
String being converted to uppercase.

Examples

```
<!--- This example shows the use of UCase --->
<HTML>
<HEAD>
<TITLE>
UCase Example
</TITLE>
</HEAD>

<BODY bgcolor=silver>
<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" METHOD="POST">
<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>

</BODY>
</HTML>
```

URLEncodedFormat

Returns URL encoded *string*. Spaces are replaced with + and all non-alphanumeric characters with equivalent hexadecimal escape sequences. This function enables you to pass arbitrary strings within a URL, because ColdFusion automatically decodes all URL parameters that are passed to the template.

Syntax

URLEncodedFormat(*string*)

string
String being URL encoded.

Examples

```
<!--- This example shows URLEncodedFormat --->
<HTML>
<HEAD>
<TITLE>
URLEncodedFormat Example
</TITLE>
</HEAD>

<BODY bgcolor=silver>
<H3>URLEncodedFormat Example</H3>

<CFIF IsDefined("url.myExample")>
<P>The url variable url.myExample has been passed from the
previous link ... its value is:
<BR>"<CFOUTPUT>#url.myExample#</CFOUTPUT>"
</CFIF>

<P>This function returns a URL encoded string, making it
safe to pass strings through a URL.
<CFSET s =
   "My url-encoded string has special characters & other stuff">

<P>
<a href=
   "urlencodedformat.cfm?myExample=<CFOUTPUT>#URLEncodedFormat("#s#")#
</CFOUTPUT>">Click me</A>

</BODY>
</HTML>
```

Val

Returns a number that the beginning of a string can be converted to. Returns 0 if conversion is not possible.

See also IsNumeric.

Syntax

Val(*string*)

string
Any string.

Examples

```
<!--- This example shows Val --->
<HTML>

<HEAD>
<TITLE>
Val Example
</TITLE>
</HEAD>

<BODY bgcolor=silver>
<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" METHOD="POST">
<P>Enter a string, and discover if
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>

</BODY>
</HTML>
```

ValueList

Returns a comma-separated list of the values of each record returned from a previously executed query.

See also QuotedValueList.

Syntax `ValueList(query.column [, delimiter ])`

query.column
Name of an already executed query and column. Separate query name and column name with a period (.).

delimiter
A string delimiter to separate column data.

Example

```
<!--- This example shows the use of ValueList --->
<HTML>
<HEAD>
<TITLE>ValueList Example</TITLE>
</HEAD>

<BODY>
<H3>ValueList Example</H3>

<!--- use the contents of one 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>

</BODY>
</HTML>
```

Week

Returns the ordinal for the week number in a year; an integer ranging from 1 to 53.

See also DatePart.

Syntax

Week(*date*)

date
Any date/time value or string convertible to date.

Usage

Years less than 100 are interpreted as 20th century values.

When passing date as a string, make sure it is enclosed in quotes. Otherwise, it is interpreted as a number representation of a date returning undesired results.

Examples

```
<!--- shows the value of the Week function --->
<HTML>
<HEAD>
<TITLE>
Week Example
</TITLE>
</HEAD>

<BODY bgcolor=silver>
<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

Appends text to the page output stream. Although you can call this function anywhere within a page, it is most useful inside a CFSCRIPT block.

Note: When within the CFQUERY and CFMAIL tags, the WriteOutput function does not output to the current page, but instead writes to the current SQL statement or mail text. Do not use WriteOutput within CFQUERY and CFMAIL.

This function writes to the page output stream regardless of conditions established by the CFSETTING tag.

Syntax

WriteOutput(*string*)

string
Text to be appended to the page output stream.

Examples

```
...
<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>
...
```

Year

Returns the year corresponding to *date*.

See also DatePart and IsLeapYear.

Syntax

Year(*date*)

date
Any date/time value or string convertible to date.

Usage

Years less than 100 are interpreted as 20th century values.

When passing a date as a string, make sure it is enclosed in quotes. Otherwise, it is interpreted as a number representation of a date returning, undesired results.

Examples

```
<!--- shows the value of the Year function --->
<HTML>
<HEAD>
<TITLE>
Year Example
</TITLE>
</HEAD>

<BODY bgcolor=silver>
<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

Returns Boolean data as YES or NO.

See also IsBoolean and IsNumeric.

Syntax

YesNoFormat(*value*)

value
Any number or Boolean value.

Usage

The YesNoFormat function returns all non-zero values as YES and zero values as NO.

Examples

```
<!--- This example shows the YesNoFormat --->
<HTML>
<HEAD>
<TITLE>YesNoFormat Example</TITLE>
</HEAD>

<BODY>
<H3>YesNoFormat Example</H3>

<P>The YesNoFormat function returns all non-zero values
as "YES" and 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>

</BODY>
</HTML>
```

CHAPTER 4

WDDX JavaScript Objects

This chapter provides information about JavaScript objects and functions used when implementing WDDX in a ColdFusion application.

Contents

WddxSerializer Object

The WddxSerializer object includes functions that serialize any JavaScript data structure.

The only function that developers typically call is serialize.

serialize

Creates a WDDX packet for a passed WddxRecordset instance.

Syntax *object*.**serialize**(*rootobj*)

object
Instance name of the WddxSerializer object.

rootobj
JavaScript data structure to be serialized.

Return value String. Returns the serialized WDDX packet if the function succeeds and a null value if an error occurs.

Usage Call this function to serialize the data in a WddxRecordset instance.

Example This example illustrates a JavaScript function that you can call to serialize a WddxRecordset instance. The function 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

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.

Syntax *object*.**serializeVariable**(*name*, *obj*)

object
Instance name of the WddxSerializer object.

name
Property to be serialized.

obj
Instance name of the value to be serialized.

Return value Boolean. Returns True if serialization was successful and False if an error occurs.

Usage Internal. You do not typically call this function.

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

Recursively serializes all eligible data in a passed instance. Data that can be serialized includes:

- String
- Number
- Boolean
- Date
- Array
- Recordset
- Any JavaScript object

This function serialize null values as empty strings.

Syntax *object*.**serializeValue**(*obj*)

object
Instance name of the WddxSerializer object.

obj
Instance name of the WddxRecordset object to be serialized.

Return value Boolean. Returns True if *obj* was serialized successfully and False if an error occurs.

Usage Internal. You do not typically call this function.

Example This example is from the WddxSerializer serialize function:

```
...
this.wddxPacket = "";
this.write("<wddxPacket version='0.9'><header/><data>");
bSuccess = this.serializeValue(rootObj);
this.write("</data></wddxPacket>");

if (bSuccess)
{
    return this.wddxPacket;
}
else
{
    return null;
}
...
```

write

Appends data to the serialized data stream.

Syntax *object*.**write**(*str*)

object
Instance name of the WddxSerializer object.

str
String to be copied to the serialized data stream.

Return value String. Returns the updated serialized data stream.

Usage Internal. You do not typically call this function.

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

The WddxRecordset object includes functions that you call as needed when constructing a WDDX recordset.

addColumn

Adds the specified column to all rows in the WddxRecordset instance.

Syntax *object*.**addColumn**(*name*)

object
Instance name of the WddxRecordset object.

name
Name of the column to add.

Return value None.

Usage This function adds the specified 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 recordset
rs = new WddxRecordset();

// add a new column
rs.addColumn("NewColumn");

// extend the recordset 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

Adds the specified number of rows to all columns in the WddxRecordset instance.

Syntax *object*.**addRows**(*n*)

object
Instance name of the WddxRecordset object.

n
Integer specifying the number of rows to add.

Return value None.

Usage This function adds the specified number of rows to every column of the WDDX record set. Initially, the row/column values are set to NULL.

Example This example calls the addRows function:

```
// create a new recordset
rs = new WddxRecordset();

// add a new column
rs.addColumn("NewColumn");

// extend the recordset 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

Returns the element in the specified row/column position.

Syntax *object*.**getField**(*row*, *col*)

object
Instance name of the WddxRecordset object.

row
Integer specifying the zero-based row number of the value to be returned.

col
Integer or string specifying the 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

Indicates the number of rows in the WddxRecordset instance.

Syntax *object*.**getRowCount**( )

object
Instance name of the 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 the 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

Sets the element in the specified row/column position.

Syntax *object*.**setField**(*row*, *col*, *value*)

object
Instance name of the WddxRecordset object.

row
Integer specifying the row containing the element to be set.

col
Integer or string specifying the column containing the element to be set.

value
Value to be 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 recordset 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

Serializes a record set.

Syntax *object*.**wddxSerialize**(*serializer*)

object
Instance name of the WddxRecordset object.

serializer

WddxSerializer instance.

Return value Boolean. Returns True if serialization was successful and False if an error occurs.

Usage Internal. You do not typically call this function.

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);
  }
...
```