CFML Language Reference for ColdFusion Express

ColdFusion Express for Windows®
NT and Windows 95/98

Allaire Corporation

Copyright Notice

Contents

Chapter 2: ColdFusion Functions 47

Welcome To ColdFusion Express

The ColdFusion Web application server provides the fastest way to integrate browser, server, and database technologies into powerful Web applications and interactive Web sites.

Up until now, the ColdFusion Web application server was available for purchase in three editions that vary based on both platform and features:

- ColdFusion Server 4.0 Professional for Windows NT
- ColdFusion Server 4.0 Enterprise for Windows NT
- ColdFusion Server 4.0 Enterprise for Solaris

When you purchase ColdFusion Server Professional or Enterprise editions, you can build everything from online stores to sophisticated business systems.

ColdFusion Express is a limited functionality version of ColdFusion Server that is distributed electronically and available for free.

Read this manual to get started developing Web applications with ColdFusion Express.

To learn about ColdFusion configurations, refer to the ColdFusion product pages.

Contents

Intended Audience

This reference is intended for web developers who have a working knowledge of ColdFusion; that is, you know how to use tags and functions, but you need a comprehensive reference that provides an in-depth description of each tag and function, and examples of how tags and functions are used in the context of an HTML/CFML page.

Developer Resources

Allaire 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	Our home page provides general information about Allaire products and services as well as regular corporate news updates.
Technical Support www.allaire.com/support	Allaire offers a wide range of professional support programs.
Training www.allaire.com/education	There are a variety of courses that you may attend at an Allaire training center, at your site, on the Internet, and even at your desktop.
Developer Community www.allaire.com/developer	The DevCenter provides the resources that you need to stay on the cutting edge of development. There are links to Support Forums, our Knowledge Base, the Developer's Exchange, technical papers and more.
Allaire Alliance www.allaire.com/partners	There is a network of solution providers, application developers, releasers, and hosting services creating solutions with ColdFusion.

About ColdFusion Express Documentation

ColdFusion Express online documentation supports all components of the ColdFusion Express Server. It is organized so that you can quickly locate the information that you need.

Online documentation

All ColdFusion Express documentation is available online in HTML in PDF formats. To view the HTML documentation, open the following URL:
http://127.0.0.1/cfdocs/dochome.htm

To view and print ColdFusion Express documentation in Acrobat format, open the following URL: http://127.0.0.1/cfdocs/AcrobatDocs/index.htm

ColdFusion Express titles

The core ColdFusion Express documentation set consists of the following titles.

Developing Web Applications with ColdFusion

Describes ColdFusion development system components, system requirements, how to verify the ColdFusion Server and Web server,ColdFusion application development and how to configure the server for performance. Task-based in nature, this guide also details the procedures for creating application pages, setting up ColdFusion data sources, and retrieving data from desktop databases.

CFML Language Reference for ColdFusion Express.

Provides the complete syntax, with example code, of all CFML tags and functions.

Documentation Conventions

When reading documentation, please be aware of these formatting cues:

- Code samples, filenames, and URLs are set in a `monospaced` font.
- URL addresses that begin with `http://127.0.0.1` direct you to pages on your Web server's local machine or the *localhost*.
- Notes and tips are identified by bold icons in the margin.
- Bulleted lists present options and features.
- Procedures are identified by bold icons in the margin.
- Our suggestions for where to go from here are identified by bold icons in the margin.
- Menu levels are separated by the greater than (>) sign.
- Text for you to type in is set in *italics*.

Printing ColdFusion Express Documentation

If you would like printed documentation to read, locate the Acrobat files by opening the following URL on the system hosting ColdFusion Express Server:

http://127.0.0.1/cfdocs/AcrobatDocs/index.htm

The Acrobat files offer excellent print output. You can print an entire manual or individual chapters.

Getting Answers

One of the best ways to solve particular programming problems is to tap into the vast expertise of the ColdFusion developer community on the Allaire Support Forums. Other ColdFusion developers on the forums can help you figure out how to do just about anything with ColdFusion. The search facility can also help you search messages going back 12 months, allowing you to learn how others have solved a problem you may be facing. The Forums are a great resource for learning ColdFusion, and they're also a great place to see the ColdFusion developer community in action.

Contacting Allaire

Corporate headquarters

Allaire Corporation
One Alewife Center
Cambridge, MA 02140

Tel: 617.761.2000
Fax: 617.761.2001

http://www.allaire.com

Sales

Toll Free: 888.939.2545

Tel: 617.761.2100
Fax: 617.761.2101

Email: sales@allaire.com

Web: http://www.allaire.com/store

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 (http://forums.allaire.com) can be made any time.

CHAPTER 1

ColdFusion Tags

This chapter describes each of the tags in the subset of the ColdFusion Markup Language (CFML) included with ColdFusion Express. The introduction contains an alphabetical summary of ColdFusion Express tags. The remainder of this chapter provides complete descriptions of each tag, listed alphabetically.

Contents

Alphabetical List of ColdFusion Express 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. ColdFusion Express supports a subset of CFML. Using CFML tags is very simple; tag syntax is much like HTML element syntax.

The following table provides brief descriptions of each CFML tag included with ColdFusion Express.

ColdFusion Express Tag Summary	
CFML Tag	**Description**
CFABORT	Stops processing of a ColdFusion page at the tag location.
CFAPPLICATION	Defines application name and activates client variables.
CFBREAK	Breaks out of a CFML looping construct.
CFCOOKIE	Defines and sets cookie variables.
CFIF/ CFELSEIF/ CFELSE	Used to create IF-THEN-ELSE constructs.
CFINCLUDE	Embeds references to ColdFusion pages.
CFINSERT	Inserts records in an ODBC data source.
CFLOCATION	Opens a ColdFusion page or HTML file.
CFLOOP	Repeats a set of instructions based on a set of conditions.
CFOUTPUT	Displays output of database query or other operation.
CFPARAM	Defines a parameter and its initial default value.
CFQUERY	Passes SQL to a database.
CFSET	Defines a variable.
CFSETTING	Define and control a variety ColdFusion settings.
CFSWITCH/ CFCASE/ CFDEFAULTCASE	Evaluates a passed expression and passes control to the CFCASE tag that matches the expression result.
CFUPDATE	Updates rows in a database data source.

ColdFusion Tags that are not in ColdFusion Express

The following tags are not in ColdFusion Express; however, both ColdFusion Server Professional and ColdFusion Server Enterprise support them.

CFML Tag Summary	
CFML Tag	**Description**
CFAPPLET	Embeds Java applets in a CFFORM.
CFASSOCIATE	Enables sub-tag data to be saved with the base tag.
CFAUTHENTICATE	Authenticates a user and sets the security context for an application.
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, optionally, the filename of a file to be downloaded by the current page.
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.
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.

CFML Tag Summary (Continued)	
CFML Tag	**Description**
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.
CFINDEX	Used to create Verity search indexes.
CFINPUT	Used in CFFORM to create input elements such as radio buttons, checkboxes, and text entry boxes.
CFLDAP	Provides access to LDAP directory servers.
CFLOCK	Synchronizes a section of CFML code.
CFMAIL	Assembles and posts an email message.
CFMODULE	Used to invoke a custom tag.
CFOBJECT	Creates and uses COM or CORBA objects.
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.
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.

CFML Tag Summary (Continued)	
CFML Tag	**Description**
CFSLIDER	Used in CFFORM to create a slider control element.
CFSTOREDPROC	Specifies database connection information and identifies the stored procedure to be executed.
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.
CFTREE	Used in CFFORM to create a tree control element.
CFTREEITEM	Used with CFTREE to populate a tree control element in a CFFORM.
CFWDDX	Serializes and de-serializes CFML data structures to the XML-based WDDX format.

You can scale your Express applications using ColdFusion Enterprise or ColdFusion Professional to deliver complex, large volume, transaction intensive solutions. Applications built with ColdFusion Express can be deployed on any other version of ColdFusion Server without modification. More detailed information on upgrade opportunities from ColdFusion Express can be found online at www.allaire.com/cfexpressupgrade

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.

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>
```

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" >
```

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

Usage

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

Example

```
<!---------------------------------------------------------------
    This example shows how to enable client variables by using the
    CFAPPLICATION tag.
---------------------------------------------------------------->
```

```
    <HTML>
    <HEAD>
        <title>Enabling Client Variables</title>
    </HEAD>

    <BASEFONT FACE="Arial, Helvetica" SIZE=2>
    <BODY  bgcolor="#FFFFD5">

    <H3>CFAPPLICATION Example</H3>

    <P>CFAPPLICATION defines scoping for a ColdFusion application and
    enables or disables the storing of client variables. This tag is
    placed in a special file called Application.cfm that is run before
    any other CF template in a directory where the Application.cfm file
    appears.

    <CFAPPLICATION NAME="ETurtle" CLIENTMANAGEMENT="yes"
    SETCLIENTCOOKIES="yes">
<!--- End of Application.cfm --->
```

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="GetProducts" DATASOURCE="CFExpress">
SELECT *
FROM Products
ORDER by Product_Name
</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 products in the
CFExpress 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 --->
<CFIF IsDefined("form.a_prod")>
    <CFLOOP QUERY="GetProducts">
        <CFIF product_ID is form.a_prod>
        <H4>Your Desired product was found:</H4>
        <PRE><CFOUTPUT>#product_name##product_description#</CFOUTPUT></
PRE>
        <CFBREAK>
        <CFELSE>
            <BR>Searching...
        </CFIF>
    </CFLOOP>
</CFIF>

<FORM action="cfbreak.cfm" method="post">

<select name="a_prod" >
        <option value="1">ColdFusion</option>
        <option value="2">HomeSite</option>
```

```
        <option value="3">JRUN</option>
</select>
<INPUT type="Submit" Value="Select an Allaire Product">

</form>

</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
shows how to schedule the expiration of the variable.
------------------------------------------------------------->
<HTML>
<HEAD>
<TITLE>
CFCOOKIE Example
</TITLE>
</HEAD>

<CFIF NOT IsDefined("bgcolor")>
    <CFCOOKIE NAME="bgcolor"
        VALUE="white"
        >
<!-------------------------------------------------------------
If expire is checked, set the cookie's expiration date to NOW.
--------------------------------------------------------------->
<CFELSEIF IsDefined("Form.Expire")>
    <CFCOOKIE NAME="bgcolor"
    VALUE="FORM.bgcolor"
    EXPIRES="NOW">
<CFELSE>
    <CFCOOKIE NAME="bgcolor"
        VALUE="FORM.bgcolor"
        EXPIRES="NEVER"
        >
</CFIF>

<CFOUTPUT>
<BODY   bgcolor="#bgcolor#">
</cfoutput>
<H3>CFCOOKIE Example</H3>

<FORM action="cfcookie.cfm" method="post">
If you do not like white, select a color from the following list:<BR>
<SELECT name="bgcolor"> <option value="white">white</option>
                                <option value="gray">grey</option>
                              <option value="green">green</option>
                              <option value="red">red</option>
                              <option value="yellow">yellow</option>
                              <option value="blue">blue</option>
</SELECT>
<BR>Check reset to make the bgcolor cookie expire.<BR>
<INPUT type="Checkbox" name="expire" value="no">
<INPUT type="submit" name="submit" Value= "My Choice">
</FORM>
</BODY>
</HTML>
```

CFIF/ CFELSEIF/ CFELSE

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
<CFELSEIF>
    HTML and CFML tags
<CFELSE 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>
```

In addition to using the decision functions in the expression, such as IsArray and IsBoolean, you can also use any of the ColdFusion operators in the expression.

CFML Operators		
CFML Operator	**Alternate**	**Checks that the right value is**
IS	EQUAL, EQ	equal to the left value.
IS NOT	NOT EQUAL, NEQ	is not equal to the left value.
CONTAINS		is contained within the left value.
DOES NOT CONTAIN		is not contained within the left value.
GREATER THAN	GT	is greater than the left value.
LESS THAN	LT	is less than the left value.
GREATER THAN OR EQUAL	GTE	is greater than or equal to the left value.
LESS THAN OR EQUAL	LTE	is less than or equal to the left value.

Note Do not substitute an equal sign (=) for the IS operator.

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 Allaire departments from the
HRAPP datasource and only display them <B>IF</B>
the location = Cambridge.

<CFQUERY name="getAllaireDepartments" datasource="HRAPP">
SELECT Department_Name, Location
From Departments
</cfquery>
<hr>
<!--- use CFIF to test a condition when outputting a query --->
<P>The following departments are in Cambridge:

<CFOUTPUT QUERY="getAllaireDepartments" >
<CFIF location is "Cambridge">
    <BR><Department:</B>#Department_Name#
</CFIF>
</CFOUTPUT>

<CFQUERY name="getAllaireEmployees" datasource="HRAPP">
SELECT Department_Name, Location
From Departments
</cfquery>

<P>If you would like more than one condition to be the case,
you can ask for a list of departments in Cambridge <B>OR</B>
San Francisco.  If the department does not meet either condition, you
can use CFELSE to show only the names and cities of the
other departments.

<CFOUTPUT QUERY="getAllaireEmployees">
<CFIF location is "San Francisco" OR location is "Cambridge">
    <BR><I>#Department_Name#, #Location#</I>
<CFELSE>
    <BR><I>#Department_Name#, #Location#</I>
</CFIF>
</CFOUTPUT>


<!--- use CFIF to specify a conditional choice for multiple
options; also note the nested CFIF --->
<hr>
<P>You can also use CFELSEIF to provide one or more conditions.</P>

<CFOUTPUT QUERY="getAllaireDepartments">
<CFIF location is "San Francisco">
    <BR><I>#Department_Name#, #Location#</I>
<CFELSEIF location is "Cambridge">
```

```
    <BR><I>#Department_Name#, #Location#</I>
<CFELSEIF location is "London">
    <BR><I>#Department_Name#, #Location#</I>
<CFELSE>
    <BR><I>#Department_Name#, #Location#</I>
</CFIF>
</CFOUTPUT>

<P>For a more complex example of the use of CFIF, see <A
HREF="listcontains.cfm">ListContains Example</A>.</P>
</BODY>
</HTML>
```

CFINCLUDE

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

Syntax

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

TEMPLATE
A logical path to an existing page.

Usage

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>
```

CFINSERT

CFINSERT inserts new records in data sources.

Syntax

```
<CFINSERT DATASOURCE="ds_name"
    DBTYPE="type"
    TABLENAME="tbl_name"
    USERNAME="username"
    PASSWORD="password"
    FORMFIELDS="formfield1, formfield2, ...">
```

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

DBTYPE
Optional. The database driver type:

- ODBC (default) — ODBC driver.

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

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.

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.

Example

```
<!---------------------------------------------------------------------
This example shows how to use CFINSERT instead of CFQUERY to add a record
to the database.
---------------------------------------------------------------------->
<HTML>

<HEAD>
<TITLE>CFINSERT Example</TITLE>
</HEAD>

<CFQUERY NAME="GetMessages" DATASOURCE="CFExpress">
SELECT    Subject, Posted, Message, Message_ID, Thread_ID, Employee_ID
FROM      Messages
</CFQUERY>
```

```
<BODY  bgcolor="#FFFFD5">
<H3>CFINSERT Example</H3>
This is a read-only example. Because database access
is a sensitive area, consider the security of your database before
allowing people to insert, update, or delete data from it.

<P>First, we'll show a list of the available comments in the cfexpress
datasource.

<!--- show all the comments in the db --->
<TABLE>
    <TR>
        <TD>Subject</TD><TD>Message</TD><TD>Date Posted</TD>
    </TR>
<CFOUTPUT query="GetMessages">
    <TR>
        <TD valign=top>#Subject#</TD>
        <TD valign=top><FONT SIZE="-2">#Left(Message, 125)#</FONT></TD>
        <TD valign=top>#Posted#</TD>
        <TD valign=top>#Message_ID#</TD>
        <TD valign=top></TD>
    </TR>
<!-----------------------------------------------------------------------
Here is where the IDs are incremented so that these values will be
correct when we insert the new message.
------------------------------------------------------------------------>
<CFSET Thread_ID = Thread_ID + 1>
<CFSET Message_ID = Message_ID + 1>
<CFSET Employee_ID = Employee_ID + 1>
</CFOUTPUT>
</TABLE>

<!-----------------------------------------------------------------------
Take out the comments around this line in order to insert a message into
the messages table.
<CFINSERT datasource="CFExpress" tablename="Messages" >
------------------------------------------------------------------------>

<P>Next, we'll offer the opportunity to enter your own comment:

<!--- make a form for input --->
<FORM ACTION="cfinsert.cfm" METHOD="POST">
<!-----------------------------------------------------------------------
Dynamically determine today's date. Pass the value of posted, Thread_ID,
Employee_ID, and Message_ID to CFINSERT by declaring them as Hidden
inputs.
------------------------------------------------------------------------>
<INPUT TYPE="Hidden" NAME="posted" VALUE="<CFOUTPUT>#Now()#</CFOUTPUT>">
<INPUT TYPE="Hidden" NAME="Thread_ID_float"
VALUE="<CFOUTPUT>#Thread_ID#</CFOUTPUT>">
<INPUT TYPE="Hidden" NAME="Message_ID_float"
VALUE="<CFOUTPUT>#Message_ID#</CFOUTPUT>">
<INPUT TYPE="Hidden" NAME="Employee_ID_float"
VALUE="<CFOUTPUT>#Employee_ID#</CFOUTPUT>">
```

```
<P><B>Subject:</B><BR>
<INPUT TYPE="Text" NAME="Subject">
<P><B>Message:</B><BR>
<TEXTAREA NAME="Message" COLS="40" ROWS="6"></TEXTAREA>

Date Posted:<CFOUTPUT>#DateFormat(Now())#</CFOUTPUT>
</PRE>
<P>
<INPUT TYPE="Submit" VALUE="Insert My Comment">
</FORM>

</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:
<CFLOCATION URL="../../cfdocs/index.htm" ADDTOKEN="Yes">

</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:

- Index Loops
- Condition Loops
- Looping over a Query
- Looping over a List
- Looping over a Structure

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.

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="CFExpress">
    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="GetTemplates"
    DATASOURCE="templateData"
    MAXROWS="5">
    SELECT TemplateName
    FROM templates
</CFQUERY>

<CFLOOP QUERY="GetTemplates">
    <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. 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.

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. If you do not specify this attribute, the default delimiter is a comma.

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 Structure

The CFLOOP COLLECTION attribute allows you to loop over a structure .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.

The COLLECTION attribute is used with the ITEM attribute in a CFLOOP. In the example that follows. ITEM is assigned a variable called `person`, so that with each cycle in the CFLOOP, each item in the stucture is referenced.

Example 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>
...
```

CFOUTPUT

Displays the results of a database query or other operation. If you need to nest CFOUTPUT tags, please read the Usage section.

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.

Usage

In order to nest CFOUTPUT blocks, you must specify the GROUP and QUERY attributes at the top-most level, and the GROUP attribute for all inner blocks except for the inner-most CFOUTPUT block.

Example

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

<!--- run a sample query --->
<CFQUERY name="GetEmployeeInfo" DATASOURCE="HRApp">
SELECT StartDate, FirstName, LastName
FROM Employees
ORDER by LastName

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

<P>CFOUTPUT tells ColdFusion Server
```

```
to begin processing, and then to hand back control
of page rendering to the web server.

<P>For example, to show today's date, you could write
#DateFormat("#Now()#").  If you enclosed that expression
in CFOUTPUT, the result would be <CFOUTPUT>#DateFormat(Now())#
  </CFOUTPUT>.

<P>In addition, CFOUTPUT may be used to show the results of
a query operation, or only a partial result, as shown:

<P>There are <CFOUTPUT>#GetEmployeeInfo.RecordCount#</CFOUTPUT> total
records
in our query.  Using the MAXROWS parameter, we are limiting our
display to 4 rows.
<P>
<CFOUTPUT query="GetEmployeeInfo" MAXROWS=4>
<PRE>#FirstName##LastName##StartDate#</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 <CFOUTPUT>#DayofWeekAsString(DayofWeek(Now()))#</CFOUTPUT>

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

CFPARAM

CFPARAM is used to test for a parameter's existence, and optionally test its data type, and provide a default value if one is not assigned.

Syntax

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

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

TYPE
Optional. The type of parameter that is required. The default value is "any."

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

Usage

There are three 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 a required variable and for the type of variable — Use CFPARAM with the NAME attribute and the TYPE attribute to test that a required variable exists, and that it is of the specified type.
- 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>
```

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"
    USERNAME="username"
    PASSWORD="password"
    MAXROWS="number"
    BLOCKFACTOR="blocksize"
    TIMEOUT="milliseconds"
    DEBUG="Yes/No"
    CACHEDAFTER="date"
    CACHEDWITHIN="timespan"
>

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.

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 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.

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.

CACHEDAFTER
Optional. Specify a date value (for example,6/16/99, June 16, 1999, 6-16-99). 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.

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.

Usage

In addition to returning data from a ColdFusion data source, the CFQUERY tag also returns informations about the query. `CFQUERY.ExecutionTime` returns the time it took the query to execute in milliseconds.

CFQUERY creates a query object and provides you with information in three query variables and in the ExecutionTime variable as described in the following table.

CFQUERY Variables	
Variable Name	**Description**
query_name.RecordCount	The total number of records returned by the query.
query_name.CurrentRow	The current row of the query being processed by CFOUTPUT.
query_name.ColumnList	Returns a comma-delimited list of the query columns.
CFQUERY.ExecutionTime	Returns the time it took to execute the query.

Example

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

<BODY>
<H3>CFQUERY Example</H3>

<HTML>
<CFQUERY NAME="GetProducts" DATASOURCE="CFExpress">
    SELECT  product_name,
            product_description
    FROM    products
    ORDER BYproduct_name
</CFQUERY>

<table cellspacing="2" cellpadding="2" border="0">
<tr>
    <th align="left">Product Name</th>
    <th align="left">Product Description</th>
</tr>
<CFOUTPUT QUERY="GetProducts">
<tr>
    <td  valign="top">#product_name#</td>
    <td>#product_description#</td>
</tr>
</CFOUTPUT>
</table>

</BODY>
</HTML>
```

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">
```

Example

```
<!--- This example shows how to use CFSET --->

<CFQUERY NAME="GetEmployeeInfo" DATASOURCE="HRApp">
SELECT   *
FROM     Employees
</CFQUERY>

<HTML>

<HEAD>
<TITLE>
CFSET Example
</TITLE>
</HEAD>

<BASEFONT FACE="Arial, Helvetica" SIZE=2>
<BODY  bgcolor="#FFFFD5">

<H3>CFSET Example</H3>

<P>CFSET allows you to set and reassign values to local or
global variables within a CF template.

<CFSET NumRecords = GetEmployeeInfo.RecordCount>
<P>For example, the variable NumRecords has been declared on
this template to hold the number 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 of the user, or any
other function or expression possible in Cold Fusion.

<CFSET the_date = #DateFormat(Now())# & " " & #TimeFormat(Now())#>
<CFSET user_ip = CGI.REMOTE_ADDR>
<CFSET complex_expr = (23 MOD 12) * 3>
<CFSET str_example = "#GetEmployeeInfo.FirstName#
#GetEmployeeInfo.LastName#" >


<CFOUTPUT>
<UL>
    <LI>The date: #the_date#
    <LI>User IP Address: #user_ip#
    <LI>Complex Expression ((23 MOD 12) * 3): #complex_expr#
    <LI>String Manipulation (the first 35 characters of
    the body of the first message in our query)
    <BR><B>Normal</B> :#str_example#
    <BR><B>Reversed</B>: #Reverse("#str_example#")#
</UL>
</CFOUTPUT>
</BODY>

</HTML>

...
```

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>
```

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.

Examples

```
<!--- 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="HRApp">
SELECT   Employee_ID, FirstName, LastName, Department_ID
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_ID#>
<!--- each time the case is fulfilled, the specific
information is printed; if the case is not fulfilled,
the default case is output --->
    <CFCASE VALUE="1">
    #FirstName# #LastName# is in <B>Training</B><BR><BR>
    </CFCASE>
    <CFCASE VALUE="2">
    #FirstName# #LastName# is in <B>Marketing</B><BR><BR>
    </CFCASE>
    <CFCASE VALUE="4">
    #FirstName# #LastName# is in <B>Sales</B><BR><BR>
    </CFCASE>
    <CFDEFAULTCASE>#FirstName# #LastName# is not in Training,
    Sales, or Marketing.<BR>
    </CFDEFAULTCASE>
</CFSWITCH>
</CFOUTPUT>

</BODY>
</HTML>
```

CFUPDATE

The CFUPDATE tag updates existing records in data sources.

Syntax

```
<CFUPDATE DATASOURCE="ds_name"
    DBTYPE="type"
    TABLENAME="table_name"
    USERNAME="username"
    PASSWORD="password"
    FORMFIELDS="field_names">
```

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

DBTYPE
Optional. The database driver type:

- ODBC (default) — ODBC driver.

TABLENAME
Required. Name of the table you want to update.

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.

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 Employee_ID has been passed to this form, then
perform the update on that record in the datasource --->

<CFIF IsDefined("form.Employee_ID")>
<!--- check that course_id is numeric --->
    <CFIF Not IsNumeric(form.Employee_ID)>
        <CFABORT>
    </CFIF>
<!--- Now, do the update --->
    <CFUPDATE DATASOURCE="HRApp"
  TABLENAME="Employees"
FORMFIELDS="Employee_ID,FirstName,LastName,Department_ID, StartDate,
Salary">
```

```
</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="GetEmployeeInfo" DATASOURCE="HRApp">
SELECT     Employee_ID, FirstName, LastName, Department_ID, StartDate,
Salary
FROM       Employees
<CFIF IsDefined("url.Employee_ID")>
WHERE    Employee_ID = #Trim(url.Employee_ID)#
</CFIF>
ORDER by Employee_ID
</CFQUERY>

<HTML>
<HEAD>
<TITLE>CFUPDATE Example</TITLE>
</HEAD>

<BASEFONT FACE="Arial, Helvetica" SIZE=2>
<BODY  bgcolor="#FFFFD5">

<H3>CFUPDATE Example</H3>

<!--- If we are updating a record, don't show
the entire list. --->
<CFIF IsDefined("url.Employee_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 "Update" to use
  CFUPDATE and alter the database</H3>

  <INPUT TYPE="Hidden" NAME="Employee_ID"
VALUE="<CFOUTPUT>#Trim(GetEmployeeInfo.Employee_ID)#</CFOUTPUT>">
  <P>Employee Name <BR> <INPUT TYPE="Text" NAME="FirstName"
VALUE="<CFOUTPUT>#Trim(GetEmployeeInfo.FirstName)#</CFOUTPUT>">
                       <INPUT TYPE="Text" NAME="LastName"
VALUE="<CFOUTPUT>#Trim(GetEmployeeInfo.LastName)#</CFOUTPUT>">
  <P>Department ID<BR>
  <INPUT TYPE="Text" NAME="Department_ID"
VALUE="<CFOUTPUT>#Trim(GetEmployeeInfo.Department_ID)#</CFOUTPUT>"></P>
  <BR>
  <P>Start Date<BR>
  <INPUT TYPE="Text" NAME="StartDate"
VALUE="<CFOUTPUT>#DateFormat(GetEmployeeInfo.StartDate)#</CFOUTPUT>">
</P>
  <P>Salary<BR>
  <INPUT TYPE="Text" NAME="Salary"
VALUE="<CFOUTPUT>#DollarFormat(GetEmployeeInfo.Salary)#</CFOUTPUT>"></P>
```

```
  <P><INPUT TYPE="Submit" VALUE="Click to Update"></P>
  </FORM>
<CFELSE>
  <!--- Show the entire record set in CFTABLE form --->
  <table cellspacing="2" cellpadding="2" border="0">
    <tr>
        <th>Edit</th>
        <th>First Name</th>
        <th>Last Name</th>
        <th>Department ID</th>
        <th>Start Date</th>
        <th>Salary</th>
    </tr>
    <CFOUTPUT query="GetEmployeeInfo">
    <tr>
        <td><a href='cfupdate.cfm?Employee_ID=#Employee_ID#'>Edit</a></
td>
        <td>#FirstName#</td>
        <td>#LastName#</td>
        <td>#Department_ID#</td>
        <td>#DateFormat(StartDate)#</td>
        <td>#DollarFormat(Salary)#</td>
    </tr>
    </CFOUTPUT>
</table>
</CFIF>

</BODY>
</HTML>
```

CHAPTER 2

ColdFusion Functions

This chapter describes each of the functions in the subset of the ColdFusion Markup Language (CFML) included with ColdFusion Express. The introduction contains an alphabetical summary of ColdFusion functions 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
ACos
ArrayAppend
ArrayAvg
ArrayClear
ArrayDeleteAt
ArrayInsertAt
ArrayIsEmpty
ArrayLen
ArrayMax
ArrayMin
ArrayNew
ArrayPrepend
ArrayResize
ArraySet
ArraySort
ArraySum
ArraySwap
ArrayToList
Asc
ASin
Atn
BitAnd
BitMaskClear
BitMaskRead
BitMaskSet
DayOfYear
DaysInMonth
DaysInYear
DE
DecimalFormat
DecrementValue
Decrypt
DeleteClientVariable
DirectoryExists
DollarFormat
Encrypt
Evaluate
Exp
ExpandPath
FileExists
Find
FindNoCase
FindOneOf
FirstDayOfMonth
Fix
FormatBaseN
GetBaseTemplatePath
GetClientVariablesList
GetCurrentTemplatePath
GetDirectoryFromPath
GetCurrentTemplatePath
IsStruct
LCase
Left
Len
ListAppend
ListChangeDelims
ListContains
ListContainsNoCase
ListDeleteAt
ListFind
ListFindNoCase
ListFirst
ListGetAt
ListInsertAt
ListLast
ListLen
ListPrepend
ListQualify
ListRest
ListSetAt
ListSort
ListToArray
ListValueCount
ListValueCountNoCase
LJustify
Log
QueryAddRow
QueryNew
QuerySetCell
QuotedValueList
Rand
Randomize
RandRange
REFind
REFindNoCase
RemoveChars
RepeatString
Replace
ReplaceList
ReplaceNoCase
REReplace
REReplaceNoCase
Reverse
Right
RJustify
Round
RTrim
Second
SetLocale
SetProfileString
Sgn
Sin

BitNot
BitOr
BitSHLN
BitSHRN
BitXor
Ceiling
Chr
CJustify
Compare
CompareNoCase
Cos
CreateDate
CreateDateTime
CreateODBCDate
CreateODBCDateTime
CreateODBCTime
CreateTime
CreateTimeSpan
CreateUUID
DateAdd
DateCompare
DateConvert
DateDiff
DateFormat
DatePart
Day
DayOfWeek
DayOfWeekAsString
GetDirectoryFromPath
GetFileFromPath
GetLocale
GetProfileString
GetTempDirectory
GetTempFile
GetTemplatePath
GetTickCount
GetTimeZoneInfo
GetToken
Hour
HTMLCodeFormat
HTMLEditFormat
IIf
IncrementValue
InputBaseN
Insert
Int
Boolean
IsBoolean
IsDate
IsDebugMode
IsDefined
IsLeapYear
IsNumeric
IsNumericDate
IsQuery
IsSimpleValue
Log10
LSCurrencyFormat
LSDateFormat
LSEuroCurrencyFormat
LSIsCurrency
LSIsDate
LSIsNumeric
LSNumberFormat
LSParseCurrency
LSParseDateTime
LSParseEuroCurrency
LSParseNumber
LSTimeFormat
LTrim
Max
Mid
Min
Minute
Month
MonthAsString
Now
NumberFormat
ParagraphFormat
ParseDateTime
Pi
PreserveSingleQuotes
Quarter
QueryAddColumn
SpanExcluding
SpanIncluding
Sqr
StripCR
StructClear
StructCopy
StructCount
StructDelete
StructFind
StructInsert
StructIsEmpty
StructKeyArray
StructKeyExists
StructKeyList
StructNew
StructUpdate
Tan
TimeFormat
Trim
UCase
URLEncodedFormat
Val
ValueList
Week
WriteOutput
Year
YesNoFormat

Array Functions

ArrayAppend	ArrayMax	ArraySum
ArrayAvg	ArrayMin	ArraySwap
ArrayClear	ArrayNew	ArrayToList
ArrayDeleteAt	ArrayPrepend	Boolean
ArrayInsertAt	ArrayResize	ListToArray
ArrayIsEmpty	ArraySet	
ArrayLen	ArraySort	

Date and Time Functions

CreateDate	DatePart	IsNumericDate
CreateDateTime	Day	Minute
CreateODBCDate	DayOfWeek	Month
CreateODBCDateTime	DayOfWeekAsString	MonthAsString
CreateODBCTime	DayOfYear	Now
CreateTime	DaysInMonth	ParseDateTime
CreateTimeSpan	DaysInYear	Quarter
DateAdd	FirstDayOfMonth	Second
DateCompare	GetTimeZoneInfo	Week
DateConvert	Hour	Year
DateDiff	IsDate	
DateFormat	IsLeapYear	

Decision Functions

Boolean
IsDate
IsDebugMode
IsDefined
IsLeapYear
IsNumeric
IsNumericDate
IsQuery
IsSimpleValue
IsStruct
LSEuroCurrencyFormat
LSIsDate
LSIsNumeric
ParseDateTime

Display and Formatting Functions

DateFormat
DecimalFormat
DollarFormat
FormatBaseN
HTMLCodeFormat
HTMLEditFormat
LSCurrencyFormat
LSDateFormat
LSEuroCurrencyFormat
LSNumberFormat
LSTimeFormat
NumberFormat
ParagraphFormat
TimeFormat
YesNoFormat

Dynamic Evaluation Functions

DE
Evaluate
IIf

International Functions

DateConvert
GetLocale
GetTimeZoneInfo
LSCurrencyFormat
LSDateFormat
LSEuroCurrencyFormat
LSIsCurrency
LSIsDate
LSIsNumeric
LSNumberFormat
LSParseCurrency
LSParseDateTime
LSParseEuroCurrency
LSParseNumber
LSTimeFormat
SetLocale

List Functions

ArrayToList
ListAppend
ListChangeDelims
ListContains
ListContainsNoCase
ListDeleteAt
ListFind
ListFindNoCase
ListFirst
ListGetAt
ListInsertAt
ListLast
ListLen
ListPrepend
ListQualify
ListRest
ListSetAt
ListSort
ListToArray
ListValueCount
ListValueCountNoCase

Mathematical Functions

Abs	BitXor	Max
ACos	Ceiling	Min
ASin	Cos	Pi
Atn	DecrementValue	Rand
BitAnd	Exp	Randomize
BitMaskClear	Fix	RandRange
BitMaskRead	IncrementValue	Round
BitMaskSet	InputBaseN	Sgn
BitNot	Int	Sin
BitOr	Log	Sqr
BitSHLN	Log10	Tan
BitSHRN		

Query Functions

IsQuery	QuerySetCell
QueryAddColumn	QuotedValueList
QueryAddRow	ValueList
QueryNew	

String Functions

Asc
Chr
CJustify
Compare
CompareNoCase
DayOfWeekAsString
FormatBaseN
Find
FindNoCase
FindOneOf
GetToken
Insert
LCase
Left
Len
LJustify
ListValueCount
ListValueCountNoCase
LSParseCurrency
LSParseDateTime
LSParseEuroCurrency
LSParseNumber
LTrim
Mid
MonthAsString
ParseDateTime
REFind
REFindNoCase
RemoveChars
RepeatString
Replace
ReplaceList
ReplaceNoCase
REReplace
REReplaceNoCase
Reverse
Right
RJustify
RTrim
SpanExcluding
SpanIncluding
Trim
UCase
Val

Structure Functions

IsStruct
StructClear
StructCopy
StructCount
StructDelete
StructFind
StructInsert
StructIsEmpty
StructKeyArray
StructKeyExists
StructKeyList
StructNew
StructUpdate

System Functions

DirectoryExists	GetFileFromPath
ExpandPath	GetProfileString
FileExists	GetTempFile
GetClientVariablesList	GetTemplatePath
GetDirectoryFromPath	SetProfileString

Other Functions

CreateUUID	PreserveSingleQuotes
Decrypt	QuotedValueList
DeleteClientVariable	StripCR
Encrypt	URLEncodedFormat
GetBaseTemplatePath	ValueList
GetClientVariablesList	WriteOutput
GetTickCount	

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>
```

ACos

Returns the arccosine of a number in radians. The arccosine is the angle whose cosine is *number*.

See also Cos, Sin, ASin, Tan, and Pi.

Syntax

ACos(*number*)

number
Cosine of the angle that is to be calculated. This value must be between -1 and 1, inclusive.

Usage

The range of the result is 0 to π.

To convert degrees to radians, multiply degrees by π/180. To convert radians to degrees, multiply radians by 180/π.

Examples

```
<!--- This example shows how to use ACos --->

<HTML>

<HEAD>
<TITLE>ACos Example</TITLE>
</HEAD>

<BODY bgcolor=silver>

<H3>ACos Example</H3>

<!--- output its arccosine value --->
<CFIF IsDefined("FORM.CosNum")>
    <CFIF IsNumeric(FORM.CosNum)>
        <CFIF FORM.CosNum LESS THAN OR EQUAL TO 1>
            <CFIF FORM.CosNum GREATER THAN OR EQUAL TO -1>
                ACos(<CFOUTPUT>#FORM.CosNum#</CFOUTPUT>)=
<CFOUTPUT>#ACos(FORM.cosNum)# Radians</CFOUTPUT>
            <CFELSE>
<!--- if it is empty, output an error message --->
            <H4>Please enter a number between -1 and 1</H4>
            </CFIF>
        <CFELSE>
<!--- if it is empty, output an error message --->
        <H4>Please enter a number between -1 and 1</H4>
        </CFIF>
    </CFIF>
</CFIF>

<FORM ACTION="acos.cfm" METHOD="POST">
<P>Type in a number to get its arccosine in Radians.
<BR><INPUT TYPE="Text" NAME="cosNum" SIZE="25">
```

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

<!-------------------------------------------------------------------
This following six lines of code keep track of the form fields that can
be dynamically generated on the screen. It uses the Fieldnames variable
with the ListLen function to determine the number of fields on the form.
-------------------------------------------------------------------->

<CFSET FormElem=2>
    <CFIF Isdefined("Form.Submit")>
        <CFIF Form.Submit is "More">
            <CFSET FormElem=ListLen(Form.Fieldnames)>
        </CFIF>
    </CFIF>

<HTML>
<HEAD>
<TITLE>ArrayAvg Example</TITLE>
</HEAD>

<BODY>
<H3>ArrayAvg Example</H3>
<P>
This example uses ArrayAvg to find the average of the numbers that you
enter into an array.<br>
If you would like to enter more than two numbers press the <b>more</b>
button.
</P>

<FORM action="arrayavg.cfm" method="post">

<!-------------------------------------------------------------------
The following code initially creates two fields and then adds fields
if the user presses the MORE button. Note that FormElem is initialized to
two at the beginning of this code to indicate that the form has two
fields.
-------------------------------------------------------------------->

<INPUT type="submit" name="submit" value="more">

<table cellspacing="2" cellpadding="2" border="0">
```

```
<CFLOOP index="LoopItem" from="1" to="#FormElem#">
    <tr>
        <CFOUTPUT>
            <th align="left">Number #LoopItem#</th>
            <td><INPUT type="text" name="number#LoopItem#"></td>
        </CFOUTPUT>
    </tr>
</CFLOOP>
</table>

<INPUT type="submit" name="submit" value="get the average">
</FORM>


<!--- create an array --->

<CFIF IsDefined("FORM.submit")>
    <CFSET myNumberArray=ArrayNew(1)>
    <CFSET Count=1>
    <CFLOOP index="ListItem" list="#Form.Fieldnames#">
        <CFIF Left(ListItem,3) is "Num">
            <CFSET myNumberArray[Count]=Val(Evaluate("number#Count#"))>
            <CFSET count=IncrementValue(Count)>
        </CFIF>
    </CFLOOP>

    <CFIF Form.Submit is "get the average">
        <!--- use ArrayAvg to get the average of the two numbers --->
        <P>The average of the numbers that you entered is
        <CFOUTPUT>#ArrayAvg(myNumberArray)#.</CFOUTPUT>
    <CFELSE>
        <CFOUTPUT>Try again. You must enter at least two numeric
values.</CFOUTPUT>
    </CFIF>
</CFIF>
</BODY>
</HTML>
```

ArrayClear

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 form  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="HRApp">
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 --->
<HTML>
<HEAD>
<TITLE>ArrayMax Example</TITLE>
</HEAD>

<BODY>
<H3>ArrayMax Example</H3>
<P>
This example uses ArrayMax to find the largest number that you have
entered into an array.<br>
</P>
<!---------------------------------------------------------------------
After checking to see if the form has been submitted, the following
code creates an array and assigns the form fields to the first two
elements in the array.
---------------------------------------------------------------------->

<CFIF IsDefined("FORM.submit")>
    <CFSET myNumberArray=ArrayNew(1)>
    <CFSET myNumberArray[1]=number1>
    <CFSET myNumberArray[2]=number2>

    <CFIF Form.Submit is "Maximum Value">
        <!--- use ArrayMax to find the largest number in the array --->
        <P>The largest number that you entered is
        <CFOUTPUT>#ArrayMax(myNumberArray)#.</CFOUTPUT>
    </CFIF>
</CFIF>


<!---------------------------------------------------------------------
The following form provides two numeric fields that are compared when
the form is submitted.
---------------------------------------------------------------------
>
<FORM action="arraymax.cfm" method="post">

<INPUT type="hidden" name="number1_Float">
<INPUT type="hidden" name="number2_Float">
<INPUT type="text" name="number1">
```

```
<br>
<INPUT type="text" name="number2">
<br>
<INPUT type="submit" name="submit" value="Maximum Value">
</FORM>
</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 ArrayMin --->
<HTML>
<HEAD>
<TITLE>ArrayMin Example</TITLE>
</HEAD>

<BODY>
<H3>ArrayMin Example</H3>
<P>
This example uses ArrayMin to find the smallest number that you have
entered into an array.<br>
</P>

<!---------------------------------------------------------------------
After checking to see if the form has been submitted, the following
code creates an array and assigns the form fields to the first two
elements in the array.
---------------------------------------------------------------------->

<CFIF IsDefined("FORM.submit")>
    <CFSET myNumberArray=ArrayNew(1)>
    <CFSET myNumberArray[1]=FORM.number1>
    <CFSET myNumberArray[2]=FORM.number2>

    <CFIF Form.Submit is "Minimum Value">
        <!--- use ArrayMin to find the smallest number in the array --->
        <P>The smallest number that you entered is
        <CFOUTPUT>#ArrayMin(myNumberArray)#.</CFOUTPUT>
    </CFIF>
</CFIF>

<!---------------------------------------------------------------------
The following form provides two numeric fields that are compared when
the form is submitted.
---------------------------------------------------------------------
>
<FORM action="arraymin.cfm" method="post">

<INPUT type="hidden" name="number1_Float">
<INPUT type="hidden" name="number2_Float">
<INPUT type="text" name="number1">
```

```
<br>
<INPUT type="text" name="number2">
<br>
<INPUT type="submit" name="submit" value="Minimum Value">
</FORM>

</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="HRApp">
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 perform a query to get the list of course --->
<CFQUERY name="GetEmployeeInfo" DATASOURCE="HRApp">
SELECT * FROM Employees
</CFQUERY>
<!--- make a new array --->
<CFSET MyArray=ArrayNew(1)>
<!--- resize that array to the number of records
in the query --->
<CFSET temp=ArrayResize(MyArray, GetEmployeeInfo.RecordCount)>
<CFOUTPUT>
The array is now #ArrayLen(MyArray)# elements, to match
the query of #GetEmployeeInfo.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 one-dimensional array with elements numerically or alphanumerically sorted.

Syntax `ArraySort(array, sort_type [, sort_order ])`

array
Name of the one-dimensional array that 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="HRApp">
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>

<BASEFONT FACE="Arial, Helvetica" SIZE=2>
<BODY  bgcolor="#FFFFD5">

<H3>ArraySum Example</H3>
<P>
This example uses ArraySum to add two numbers together.<br>
</P>
<!---------------------------------------------------------------------
After checking to see if the form has been submitted, the following
code creates an array and assigns the form fields to the first two
elements in the array.
---------------------------------------------------------------------->

<CFIF IsDefined("FORM.submit")>
    <CFSET myNumberArray=ArrayNew(1)>
    <CFSET myNumberArray[1]=number1>
    <CFSET myNumberArray[2]=number2>

    <CFIF Form.Submit is "Add">
        <!--- use ArraySum to add the number in the array --->
        <P>The sum of the numbers is
        <CFOUTPUT>#ArraySum(myNumberArray)#.</CFOUTPUT>
    </CFIF>
</CFIF>

<!---------------------------------------------------------------------
The following form provides two numeric fields that are added when
the form is submitted.
---------------------------------------------------------------------
>
<FORM action="arraysum.cfm" method="post">

<INPUT type="hidden" name="number1_Float">
<INPUT type="hidden" name="number2_Float">
<INPUT type="text" name="number1">
```

```
<br>
<INPUT type="text" name="number2">
<br>
<INPUT type="submit" name="submit" value="Add">
</FORM>
</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="HRApp">
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>
```

ASin

Returns the arcsine of a number in radians. The arcsine is the angle whose sine is *number*.

See also Sin, Cos, Pi, and Tan.

Syntax

ASin(*number*)

number
Sine of the angle that is to be calculated. This value must be between 1 and -1.

Usage

The range of the result is -π/2 to π/2 radians. To convert degrees to radians, multiply degrees by π/180. To convert radians to degrees, multiply radians by 180/π.

Examples

```
<!--- This example shows how to use ASin --->

<HTML>
<HEAD>
<TITLE>ASin Example</TITLE>
</HEAD>

<BODY bgcolor=silver>
<H3>ASin Example</H3>
<!--- output its arcsine value --->
<CFIF IsDefined("FORM.SinNum")>
    <CFIF IsNumeric(FORM.SinNum)>
        <CFIF FORM.SinNum LESS THAN OR EQUAL TO 1>
            <CFIF FORM.SinNum GREATER THAN OR EQUAL TO -1>
                ASin(<CFOUTPUT>#FORM.SinNum#</CFOUTPUT>) =
                  <CFOUTPUT>#Evaluate(ASin(FORM.sinNum))# Radians</
CFOUTPUT>
            <CFELSE>
<!--- if it is less than negative one, output an error message --->
            <H4>Please enter the sine of the angle that is to be
calculated in degrees and radians. This value must be between 1 and -1,
inclusive.</H4>
            </CFIF>
        <CFELSE>
<!--- if it is greater than one, output an error message --->
        <H4>Please enter the sine of the angle that is to be calculated in
degrees and radians. This value must be between 1 and -1, inclusive.</H4>
        </CFIF>
    <CFELSE>
<!--- if it is empty, output an error message --->
        <H4>Please enter the sine of the angle that is to be calculated in
radians. This value must be between 1 and -1, inclusive.</H4>
    </CFIF>
</CFIF>
```

```
<FORM ACTION="asin.cfm" METHOD="POST">
<P>Type in a number to get its arcsine in Radians and Degrees.
<BR><INPUT TYPE="Text" NAME="sinNum" SIZE="25">
<P><INPUT TYPE="Submit" NAME=""> <INPUT TYPE="RESET">
</FORM>

</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 -π/2 to π/2 radians. To convert degrees to radians, multiply degrees by π/180. To convert radians to degrees, multiply radians by 180/π.

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
    <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.Submit")>
        <CFOUTPUT>#FORM.charVals#=#CHR(FORM.charVals)#</CFOUTPUT>
</CFIF>

<FORM ACTION="chr.cfm" METHOD="POST">
<P>Type in a number between 1 and 256 to see the ASCII character
representation.
<INPUT TYPE="hidden" Name="CharVals_range" Value="Min=1 Max=256">
<INPUT TYPE="hidden" Name="CharVals_required" Value="Please enter an
integer from 1 to 256">

<BR><INPUT TYPE="Text"
       NAME="CharVals">
<P><INPUT TYPE="Submit" NAME="Submit"> <INPUT TYPE="RESET">
</FORM>

</BODY>
</HTML>
```

CJustify

Centers a string in the specified field length by adding padding on either side of the string.

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 --->
<CFIF NOT IsDefined("jstring")>
    <CFSET jstring="">
</CFIF>

<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
          </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. Years from 0 to 29 are interpreted as 21st century values. Years 30 to 99 are interpreted as 20th century values.

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.

Examples

```
<!-------------------------------------------------------
This example shows how to use CreateDate, CreateDateTime, and
createODBCdate
-------------------------------------------------------->

<HTML>

<HEAD>
<TITLE>
CreateDate Example
</TITLE>
</HEAD>

<BASEFONT FACE="Arial, Helvetica" SIZE=2>
<BODY  bgcolor="#FFFFD5">

<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>


<FORM ACTION="createdate.cfm" METHOD="POST">
<!-------------------------------------------------------------------------
The following hidden fields provide server-side validation of required
fields and ensures that the data type is numeric.
-------------------------------------------------------------------------->
<INPUT TYPE="hidden" NAME="year_Required">
<INPUT TYPE="hidden" NAME="month_Required">
<INPUT TYPE="hidden" NAME="day_Required">
<INPUT TYPE="hidden" NAME="year_Integer">
<INPUT TYPE="hidden" NAME="month_Integer">
<INPUT TYPE="hidden" NAME="day_Integer">

<P>Please enter the year, month and day in integer FORMat for
the date value you would like to view:
<PRE>
Year<INPUT TYPE="Text" NAME="year" VALUE="1999">
Month<INPUT TYPE="Text" NAME="month" VALUE="6">
Day <INPUT TYPE="Text" NAME="day" VALUE="8">
</PRE>
<P><INPUT TYPE="Submit" NAME=""> <INPUT TYPE="RESET">

</FORM>

</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. Years from 0 to 29 are interpreted as 21st century values. Years 30 to 99 are interpreted as 20th century values.

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.

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:
```

```
        #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>

<FORM ACTION="createdatetime.cfm" METHOD="POST">
<!-----------------------------------------------------------------------
The following hidden fields provide server-side validation of required
fields and ensures that the data type is numeric.
------------------------------------------------------------------------>
<INPUT TYPE="hidden" NAME="year_Required">
<INPUT TYPE="hidden" NAME="month_Required">
<INPUT TYPE="hidden" NAME="day_Required">
<INPUT TYPE="hidden" NAME="hour_Required">
<INPUT TYPE="hidden" NAME="minute_Required">
<INPUT TYPE="hidden" NAME="second_Required">
<INPUT TYPE="hidden" NAME="year_Integer">
<INPUT TYPE="hidden" NAME="month_Range" Value="Min=1 Max=12">
<INPUT TYPE="hidden" NAME="day_Range" Value="Min=1 Max=31">
<INPUT TYPE="hidden" NAME="hour_Range" Value="Min=1 Max=24">
<INPUT TYPE="hidden" NAME="minute_Range" Value="Min=1 Max=60">
<INPUT TYPE="hidden" NAME="second_Range" Value="Min=0 Max=60">

<P>Please enter the year, month and day in integer FORMat for
the date value you would like to view:

<PRE>
Year<INPUT TYPE="Text" NAME="year" VALUE="1999">
Month<INPUT TYPE="Text" NAME="month" VALUE="6">
Day <INPUT TYPE="Text" NAME="day" VALUE="8">
Hour<INPUT TYPE="Text" NAME="hour" VALUE="16">
Minute<INPUT TYPE="Text" NAME="minute" VALUE="12">
Second<INPUT TYPE="Text" NAME="second" VALUE="0">
</PRE>
<P><INPUT TYPE="Submit" NAME=""> <INPUT TYPE="RESET">
</FORM>
```

```
</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, and
createODBCdate
------------------------------------------------------------>

<HTML>

<HEAD>
<TITLE>
CreateODBCDate Example
</TITLE>
</HEAD>

<BASEFONT FACE="Arial, Helvetica" SIZE=2>
<BODY  bgcolor="#FFFFD5">

<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>

<FORM ACTION="createodbcdate.cfm" METHOD="POST">
<!-------------------------------------------------------------------
The following hidden fields provide server-side validation of required
fields and ensures that the data type is numeric.
-------------------------------------------------------------------->
<INPUT TYPE="hidden" NAME="year_Required">
<INPUT TYPE="hidden" NAME="month_Required">
<INPUT TYPE="hidden" NAME="day_Required">
<INPUT TYPE="hidden" NAME="hour_Required">
<INPUT TYPE="hidden" NAME="minute_Required">
<INPUT TYPE="hidden" NAME="second_Required">
<INPUT TYPE="hidden" NAME="month_Range" Value="Min=1 Max=12">
<INPUT TYPE="hidden" NAME="day_Range" Value="Min=1 Max=31">
<INPUT TYPE="hidden" NAME="hour_Range" Value="Min=1 Max=24">
<INPUT TYPE="hidden" NAME="minute_Range" Value="Min=1 Max=60">
<INPUT TYPE="hidden" NAME="second_Range" Value="Min=0 Max=60">

<P>Please enter the year, month and day in integer FORMat for
the date value you would like to view:
<PRE>
Year<INPUT TYPE="Text" NAME="year" VALUE="1999">
Month<INPUT TYPE="Text" NAME="month" VALUE="6">
Day <INPUT TYPE="Text" NAME="day" VALUE="8">
Hour<INPUT TYPE="Text" NAME="hour" VALUE="16">
Minute<INPUT TYPE="Text" NAME="minute" VALUE="12">
Second<INPUT TYPE="Text" NAME="second" VALUE="0">
</PRE>
<P><INPUT TYPE="Submit" NAME=""> <INPUT TYPE="RESET">

</FORM>
</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, 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>

<FORM ACTION="createodbcdatetime.cfm" METHOD="POST">
<!-----------------------------------------------------------------------
The following hidden fields provide server-side validation of required
fields and ensures that the data type is numeric.
------------------------------------------------------------------------>
<INPUT TYPE="hidden" NAME="year_Required">
<INPUT TYPE="hidden" NAME="month_Required">
<INPUT TYPE="hidden" NAME="day_Required">
<INPUT TYPE="hidden" NAME="hour_Required">
<INPUT TYPE="hidden" NAME="minute_Required">
<INPUT TYPE="hidden" NAME="second_Required">
<INPUT TYPE="hidden" NAME="month_Range" Value="Min=1 Max=12">
<INPUT TYPE="hidden" NAME="day_Range" Value="Min=1 Max=31">
<INPUT TYPE="hidden" NAME="hour_Range" Value="Min=1 Max=24">
<INPUT TYPE="hidden" NAME="minute_Range" Value="Min=1 Max=60">
<INPUT TYPE="hidden" NAME="second_Range" Value="Min=0 Max=60">

<P>Please enter the year, month and day in integer FORMat for
the date value you would like to view:
<PRE>
Year<INPUT TYPE="Text" NAME="year" VALUE="1999">
Month<INPUT TYPE="Text" NAME="month" VALUE="6">
Day <INPUT TYPE="Text" NAME="day" VALUE="8">
Hour<INPUT TYPE="Text" NAME="hour" VALUE="16">
Minute<INPUT TYPE="Text" NAME="minute" VALUE="12">
Second<INPUT TYPE="Text" NAME="second" VALUE="0">
</PRE>
<P><INPUT TYPE="Submit" NAME=""> <INPUT TYPE="RESET">

</FORM>
</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.

Years from 0 to 29 are interpreted as 21st century values. Years 30 to 99 are interpreted as 20th century values.

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>

<FORM action="createodbctime.cfm" METHOD="post">
<INPUT TYPE="hidden" NAME="hour_Required">
```

```
<INPUT TYPE="hidden" NAME="minute_Required">
<INPUT TYPE="hidden" NAME="second_Required">
<INPUT TYPE="hidden" NAME="hour_Range" Value="Min=1 Max=24">
<INPUT TYPE="hidden" NAME="minute_Range" Value="Min=1 Max=60">
<INPUT TYPE="hidden" NAME="second_Range" Value="Min=0 Max=60">
<PRE>
Hour<INPUT TYPE="Text" NAME="hour" VALUE="16">
Minute<INPUT TYPE="Text" NAME="minute" VALUE="12">
Second<INPUT TYPE="Text" NAME="second" VALUE="0">
</PRE>
<P><INPUT TYPE="Submit" NAME=""> <INPUT TYPE="RESET">
</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>

<BASEFONT FACE="Arial, Helvetica" SIZE=2>
<BODY  bgcolor="#FFFFD5">
<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>

<FORM action="createtime.cfm" METHOD="post">
<INPUT TYPE="hidden" NAME="hour_Required">
<INPUT TYPE="hidden" NAME="minute_Required">
<INPUT TYPE="hidden" NAME="second_Required">
<INPUT TYPE="hidden" NAME="hour_Range" Value="Min=1 Max=24">
<INPUT TYPE="hidden" NAME="minute_Range" Value="Min=1 Max=60">
<INPUT TYPE="hidden" NAME="second_Range" Value="Min=0 Max=60">
<PRE>
Hour<INPUT TYPE="Text" NAME="hour" VALUE="16">
Minute<INPUT TYPE="Text" NAME="minute" VALUE="12">
Second<INPUT TYPE="Text" NAME="second" VALUE="0">
</PRE>
<P><INPUT TYPE="Submit" NAME=""> <INPUT TYPE="RESET">
</FORM>
</BODY>
</HTML>
```

CreateTimeSpan

Creates a date/time object for adding and subtracting other date/time objects.

See also CreateDateTime, DateAdd, and DateConvert.

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 CFPARAM 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>
...
```

CreateUUID

Returns a Universally Unique Identifier (UUID) formatted as 'XXXXXXXX-XXXX-XXXX-XXXXXXXXXXXXXXXX' where 'X' stands for a hexadecimal digit (0-9 or A-F).

Syntax

CreateUUID()

Usage

Each UUID returned by the CreateUUID function is a 35-character-string representation of a unique 128-bit integer. Use the CreateUUID function when you need a unique string that you will use as a persistent identifier in a distributed environment. To a very high degree of certainty, this function returns a unique value; no other invocation on the same or any other system should return the same value.

UUIDs are used by distributed computing frameworks, such as DCE/RPC, COM+, and CORBA. With ColdFusion, you can use UUIDs as primary table keys for applications where data is stored on a number of shared databases. In such cases, using numeric keys may cause primary key constraint violations during table merges. By using UUIDs, you can eliminate these violations because each UUID is unique.

Examples

```
<!--- This example shows how to use CreateUUID --->

<HTML>
<HEAD>
<TITLE>CreateUUID Example</TITLE>
</HEAD>

<BODY>
<H3>CreateUUID Example</H3>

<P>
This example uses CreateUUID to generate a UUID when
you submit the form. <CFFORMYou can submit the form as many times as you
wish.
</P>

<!--- This code checks to see if the form was submitted, then creates a
    UUID if it was. --->

<CFIF IsDefined("Form.CreateUUID") Is True>
    <HR>
    <P>Your new UUID is: <CFOUTPUT>#CreateUUID()#</CFOUTPUT></P>
</CFIF>

<FORM action="createuuid.cfm" METHOD="post">

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

</BODY>
</HTML>
```

DateAdd

Returns a date to which a specified time interval has been added.

See also DateConvert, 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="GetStartDate" DATASOURCE="HRApp">
SELECT  StartDate, LastName, FirstName
FROM Employees
</CFQUERY>

<P>This example uses DateAdd to determine the due date
of the first review for each employee in the database.
```

```
</P>

<TABLE>
<TR>
    <TD>Name</TD>
    <TD>Start Date</TD>
    <TD>Years</TD>
</TR>
<CFOUTPUT query="GetStartDate">
<TR>
    <TD>#FirstName# #LastName#</TD>
    <TD>#DateFormat(StartDate)#</TD>
    <CFSET ReviewDate = DateAdd("m", 6, StartDate)>
    <TD>#DateFormat(ReviewDate)#</TD>
</TR>
</CFOUTPUT>
</TABLE>

</BODY>
</HTML>
```

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 the description of *datePart* for information on specifying the precision of the comparison.

See also CreateDateTime.

Syntax **DateCompare**(*date1*, *date2* [, *datePart*])

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.

datePart
Optional. The precision of the comparison. This parameter can have any of the following values:

- s - precise to the second.
- n- precise to the minute.
- h- precise to the hour.
- d- precise to the day.
- m- precise to the month.
- yyyy- precise to the year.

By default, precision is to the second.

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 --->
<HTML>

<HEAD>
<TITLE>
DateCompare Example
</TITLE>
</HEAD>

<BASEFONT FACE="Arial, Helvetica" SIZE=2>

<H3>DateCompare Example</H3>

<P>The datecompare function compares two date/time values.
```

```
<CFIF IsDefined("FORM.date1")>

    <CFIF IsDate(FORM.date1) and IsDate(FORM.date2)>

        <CFSET comparison=DateCompare(FORM.date1, FORM.date2,
FORM.precision)>

<!--- switch on the variable to give various responses --->
        <CFSWITCH EXPRESSION="#comparison#">
            <CFCASE value="-1">
                <H3><CFOUTPUT>#DateFormat(FORM.date1)#
                #TimeFormat(FORM.date1)#</CFOUTPUT> (Date 1) is
                earlier than <CFOUTPUT>#DateFormat(FORM.date2)#
                #TimeFormat(FORM.date2)#</CFOUTPUT> (Date 2)</H3>
                <I>The dates are not equal</I>
            </CFCASE>
            <CFCASE value="0">
                <H3><CFOUTPUT>#DateFormat(FORM.date1)#
                #TimeFormat(FORM.date1)#</CFOUTPUT> (Date 1) is equal
                to <CFOUTPUT>#DateFormat(FORM.date2)#
                #TimeFormat(FORM.date2)#</CFOUTPUT> (Date 2)</H3>
                <I>The dates are equal!</I>
            </CFCASE>
            <CFCASE value="1">
                <H3><CFOUTPUT>#DateFormat(FORM.date1)#
                #TimeFormat(FORM.date1)#</CFOUTPUT> (Date 1) is later
                than <CFOUTPUT>#DateFormat(FORM.date2)#
                #TimeFormat(FORM.date2)#</CFOUTPUT> (Date 2)</H3>
                <I>The dates are not equal</I>
            </CFCASE>
            <CFDEFAULTCASE>
                <H3>This is the default case</H3>
            </CFDEFAULTCASE>
        </CFSWITCH>

    <CFELSE>
        <H3>Please enter two valid date values</H3>

    </CFIF>

</CFIF>

<FORM ACTION="datecompare.cfm" METHOD="POST">
<HR size="2" color="#0000A0">
<P>Date 1
<BR><INPUT TYPE="Text" NAME="date1" VALUE="<CFOUTPUT>#DateFormat(Now())#
#TimeFormat(Now())#
</CFOUTPUT>">

<P>Date 2
<BR><INPUT TYPE="Text" NAME="date2" VALUE="<CFOUTPUT>#DateFormat(Now())#
#TimeFormat(Now())#
</CFOUTPUT>">
```

```
<P>Specify precision to the:
<BR><SELECT NAME="precision">
            <OPTION VALUE="s">
                Second
            </OPTION>
            <OPTION VALUE="n">
                Minute
            </OPTION>
            <OPTION VALUE="h">
                Hour
            </OPTION>
            <OPTION VALUE="d">
                Day
            </OPTION>
            <OPTION VALUE="m">
                Month
            </OPTION>
            <OPTION VALUE="yyyy">
                Year
            </OPTION>
    </SELECT>

<P><INPUT TYPE="Submit" VALUE="Compare these dates" NAME="">
<INPUT TYPE="RESET">

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

DateConvert

Converts local time to Universal Coordinated Time (UTC) or UTC to local time based on the specified parameters. This function uses the daylight savings settings in the executing machine to compute daylight savings time, if required.

See also GetTimeZoneInfo, CreateDateTime, and DatePart.

Syntax

DateConvert(*conversion-type*, *date*)

conversion-type
There are two conversion types: "local2Utc" and "utc2Local." The former converts local time to UTC time. The later converts UTC time to local time.

date
Any ColdFusion date and time string. In order to create a ColdFusion date and time, use CreateDateTime.

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 DateConvert --->

<HTML>
<HEAD>
<TITLE>DateConvert Example</TITLE>
</HEAD>

<BASEFONT FACE="Arial, Helvetica" SIZE=2>

<BODY  bgcolor="#FFFFD5">

<H3>DateConvert Example</H3>

<!--- This part of the example shows the conversion of the current date
       and time to UTC time and back again. --->

<CFSET curDate=Now()>
<P><CFOUTPUT>The current date and time: #curDate#. </CFOUTPUT></P>
<CFSET utcDate=DateConvert("local2utc", curDate)>
<CFOUTPUT>
    <P>The current date and time converted to UTC time: #utcDate#.</P>
</CFOUTPUT>

<!--- This code checks to see if the form was submitted.
      If it was submitted the code generates the CFML date with the
CreateDateTime function. --->

<CFIF IsDefined("FORM.submit")>
```

```
    <CFSET yourDate=CreateDateTime(FORM.year, FORM.month, FORM.day,
FORM.hour,FORM.minute, FORM.second)>
    <p><CFOUTPUT>Your date value, presented as a ColdFusion date-time
string:#yourdate#.</CFOUTPUT></p>
    <CFSET yourUTC=DateConvert("local2utc", yourDate)>
    <P><CFOUTPUT>Your date and time value, converted into Universal
Coordinated Time (UTC): #yourUTC#.</CFOUTPUT></P>
    <P><CFOUTPUT>Your UTC date and time, converted back to local date and
time: #DateConvert("utc2local", yourUTC)#.</CFOUTPUT></P>
<CFELSE>
    Type the date and time, and press Enter to see the conversion.
</CFIF>


<Hr size="2" color="#0000A0">

<FORM ACTION="dateconvert.cfm" METHOD="POST">
<P>Please enter the year, month and day in integer format for
the date value you would like to view:


<table cellspacing="2" cellpadding="2" border="0">
<tr>
    <td>Year</td>
    <td><INPUT TYPE="Text" NAME="year" VALUE="1998" VALIDATE="integer"
REQUIRED="Yes"></td>
</tr>
<tr>
    <td>Month</td>
    <td><INPUT TYPE="Text" NAME="month" VALUE="6" RANGE="1,12"
MESSAGE="Please enter a month (1-12)" VALIDATE="integer"
REQUIRED="Yes"></td>
</tr>
<tr>
    <td>Day</td>
    <td><INPUT TYPE="Text" NAME="day" VALUE="8" RANGE="1,31"
MESSAGE="Please enter a day of the month (1-31)" VALIDATE="integer"
REQUIRED="Yes"></td>
</tr>
<tr>
    <td>Hour</td>
    <td><INPUT TYPE="Text" NAME="hour" VALUE="16" RANGE="0,23"
MESSAGE="You must enter an hour (0-23)" VALIDATE="integer"
REQUIRED="Yes"></td>
</tr>
<tr>
    <td>Minute</td>
    <td><INPUT TYPE="Text" NAME="minute" VALUE="12" RANGE="0,59"
MESSAGE="You must enter a minute value (0-59)" VALIDATE="integer"
REQUIRED="Yes"></td>
</tr>
<tr>
    <td>Second</td>
```

```
    <td><INPUT TYPE="Text" NAME="second" VALUE="0" RANGE="0,59"
MESSAGE="You must enter a value for seconds (0-59)" VALIDATE="integer"
REQUIRED="Yes"></td>
</tr>
<tr>
    <td><INPUT TYPE="Submit" NAME="submit" VALUE="Submit"></td>
    <td><INPUT TYPE="RESET"></td>
</tr>
</table>

</BODY>
</HTML>
```

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 DateConvert.

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 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 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 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 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 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 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 from 0 to 29 are interpreted as 21st century values. Years 30 to 99 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 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

```
<!--- 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 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

```
<!--- 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 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

```
<!--- 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 .

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>

<!--- 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>
</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(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.

Note To check for the existence of a directory, ColdFusion must have access rights to its drive.

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, 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=counter + (1/4)>
<CFSET half=counter + (1/2)>
<CFSET threefourth=counter + (3/4)>
<CFOUTPUT>
<PRE>
bill#DollarFormat(full)##DollarFormat(quarter)#
  #DollarFormat(half)#  #DollarFormat(threefourth)#
18% tip#DollarFormat(full * (18/100))#
  #DollarFormat(quarter * (18/100))#
  #DollarFormat(half * (18/100))#
  #DollarFormat(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>

<!--- 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.Submit")>
<CFOUTPUT>
<P>Your number, #FORM.number#
<BR>#FORM.number# raised to the E power: #exp(FORM.number)#
<CFIF FORM.number LTE 0><BR>You must enter a positive real number to
see 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>
<FORM 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.
<INPUT TYPE=”Hidden” NAME="number_Required">
<INPUT TYPE=”Hidden” NAME="number_Float" Value="You must enter a number">
<INPUT TYPE="Submit" NAME="Submit">
</FORM>
</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, GetClientVariablesList, 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.

Note To check for the existence of a file, ColdFusion must have access rights to its drive.

See also DirectoryExists, ExpandPath, and GetClientVariablesList.

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, and Replace.

Syntax `Find(substring, string [, start ])`

substring
String being sought.

string
String being searched.

start
Starting position for the search.

Examples

```
<!--- This example uses find, findnocase, and findoneof
 to see if a substring exists in a web page --->
<HTML>
<HEAD>
<TITLE>
Find Example
</TITLE>
</HEAD>

<BODY  bgcolor="#FFFFD5">

<CFSET thisPath=ExpandPath("*.*")>
<CFSET thisDirectory=GetDirectoryFromPath(thisPath)>
<CFIF IsDefined("form.yourFile")>
    <CFSET yourFile=form.yourFile>
    <CFIF FileExists(ExpandPath(yourfile))>
        <P>Your file exists in this directory.  You entered the correct
        file name, <CFOUTPUT>#GetFileFromPath("#thisPath#/#yourfile#")#
        </CFOUTPUT>.
    <CFELSE>
        <P>Your file was not found in this directory:
        <CFOUTPUT>#thisDirectory#</CFOUTPUT>.
    </CFIF>
</CFIF>

<!--- wait to have a string for searching defined --->
<CFIF IsDefined("form.myString") and IsDefined("form.type")>
    <!--- Now, test to see if the term was found --->
    A   <CFIF form.type is "find">case-sensitive
            <CFSET tag="find">
        <CFELSEIF form.type is "findNoCase">case-insensitive
            <CFSET tag="findNoCase">
```

```
        <CFELSEIF form.type is "findOneOf">find the first instance of any
                                        character
            <CFSET tag="findOneOf">
        </CFIF>
    test for your string "<CFOUTPUT>#form.MyString# </CFOUTPUT>" in
    "<CFOUTPUT>#Form.yourFile#</CFOUTPUT>"
</CFIF>

<!--- depending upon the action desired, perform different function --->
<CFIF IsDefined("tag")>
<CFSWITCH EXPRESSION="#tag#">
    <CFCASE value="find">
        <CFSET TheAction=Find(form.MyString, FORM.yourFile , 1)>
    </CFCASE>
    <CFCASE value="findNoCase">
        <CFSET TheAction=FindNoCase(form.MyString, FORM.yourFile, 1)>
    </CFCASE>
    <CFCASE value="findOneof">
        <CFSET TheAction=FindOneOf(form.MyString, FORM.yourFile, 1)>
    </CFCASE>
    <CFDEFAULTCASE>
        <CFSET TheAction=Find(form.MyString, FORM.YourFile, 1)>
    </CFDEFAULTCASE>
</CFSWITCH>
    <CFIF TheAction is not 0>
        found an instance of your string at index <CFOUTPUT>#theAction#
        </CFOUTPUT>
    <CFELSE>
        <P>No instance of your string was found.
    </CFIF>
</CFIF>

<H3>Find Example</H3>

<P>This example uses find, findnocase and findoneof to see
if a substring exists in a particular file in the directory
<CFOUTPUT>#GetDirectoryFromPath(thisPath)#</CFOUTPUT>.

<FORM ACTION="find.cfm" METHOD="POST">
<P>Enter the name of a file in this directory <I><FONT SIZE="-1"></P>
<INPUT TYPE="Text" NAME="yourFile" VALUE="daysinyear.cfm">

<P>And a substring to search for:
<BR><INPUT TYPE="Text" size=25 NAME="myString" VALUE="daysinyear">
<P>Pick a search type:
    <SELECT NAME="type">
        <OPTION VALUE="find" SELECTED>Case-Sensitive
        <OPTION VALUE="findNoCase">Case-Insensitive
        <OPTION VALUE="findOneOf">Find first instance
    </SELECT>
<INPUT TYPE="Submit" NAME="" VALUE="start search">
</FORM>
</BODY>
</HTML>
```

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, and Replace functions.

Syntax `FindNoCase(substring, string [, start ])`

substring
String being sought.

string
String being searched.

start
Starting position for the search.

Examples

```
<!--- This example uses find, findnocase, and findoneof
 to see if a substring exists in a web page --->
<HTML>
<HEAD>
<TITLE>
FindNoCase Example
</TITLE>
</HEAD>

<BODY  bgcolor="#FFFFD5">

<CFSET thisPath=ExpandPath("*.*")>
<CFSET thisDirectory=GetDirectoryFromPath(thisPath)>
<CFIF IsDefined("form.yourFile")>
    <CFSET yourFile=form.yourFile>
    <CFIF FileExists(ExpandPath(yourfile))>
        <P>Your file exists in this directory.  You entered the correct
        file name, <CFOUTPUT>#GetFileFromPath("#thisPath#/#yourfile#")#
        </CFOUTPUT>.
    <CFELSE>
        <P>Your file was not found in this directory:
        <CFOUTPUT>#thisDirectory#</CFOUTPUT>.
    </CFIF>
</CFIF>

<!--- wait to have a string for searching defined --->
<CFIF IsDefined("form.myString") and IsDefined("form.type")>
    <!--- Now, test to see if the term was found --->
    A   <CFIF form.type is "find">case-sensitive
            <CFSET tag="find">
        <CFELSEIF form.type is "findNoCase">case-insensitive
            <CFSET tag="findNoCase">
```

```
        <CFELSEIF form.type is "findOneOf">find the first instance of any
                                        character
            <CFSET tag="findOneOf">
        </CFIF>
    test for your string "<CFOUTPUT>#form.MyString# </CFOUTPUT>" in
    "<CFOUTPUT>#Form.yourFile#</CFOUTPUT>"
</CFIF>

<!--- depending upon the action desired, perform different function --->
<CFIF IsDefined("tag")>
<CFSWITCH EXPRESSION="#tag#">
    <CFCASE value="find">
        <CFSET TheAction=Find(form.MyString, FORM.yourFile , 1)>
    </CFCASE>
    <CFCASE value="findNoCase">
        <CFSET TheAction=FindNoCase(form.MyString, FORM.yourFile, 1)>
    </CFCASE>
    <CFCASE value="findOneof">
        <CFSET TheAction=FindOneOf(form.MyString, FORM.yourFile, 1)>
    </CFCASE>
    <CFDEFAULTCASE>
        <CFSET TheAction=Find(form.MyString, FORM.YourFile, 1)>
    </CFDEFAULTCASE>
</CFSWITCH>
    <CFIF TheAction is not 0>
        found an instance of your string at index <CFOUTPUT>#theAction#
        </CFOUTPUT>
    <CFELSE>
        <P>No instance of your string was found.
    </CFIF>
</CFIF>

<H3>FindNoCase Example</H3>

<P>This example uses find, findnocase and findoneof to see
if a substring exists in a particular file in the directory
<CFOUTPUT>#GetDirectoryFromPath(thisPath)#</CFOUTPUT>.

<FORM ACTION="findnocase.cfm" METHOD="POST">
<P>Enter the name of a file in this directory <I><FONT SIZE="-1"></P>
<INPUT TYPE="Text" NAME="yourFile" VALUE="daysinyear.cfm">

<P>And a substring to search for:
<BR><INPUT TYPE="Text" size=25 NAME="myString" VALUE="daysinyear">
<P>Pick a search type:
    <SELECT NAME="type">
        <OPTION VALUE="find" SELECTED>Case-Sensitive
        <OPTION VALUE="findNoCase">Case-Insensitive
        <OPTION VALUE="findOneOf">Find first instance
    </SELECT>
<INPUT TYPE="Submit" NAME="" VALUE="start search">
</FORM>
</BODY>
</HTML>
```

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 and Compare 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

```
<!--- This example uses find, findnocase, and findoneof
 to see if a substring exists in a web page --->
<HTML>
<HEAD>
<TITLE>
FindOneOf Example
</TITLE>
</HEAD>

<BODY  bgcolor="#FFFFD5">

<CFSET thisPath=ExpandPath("*.*")>
<CFSET thisDirectory=GetDirectoryFromPath(thisPath)>
<CFIF IsDefined("form.yourFile")>
    <CFSET yourFile=form.yourFile>
    <CFIF FileExists(ExpandPath(yourfile))>
        <P>Your file exists in this directory.  You entered the correct
        file name, <CFOUTPUT>#GetFileFromPath("#thisPath#/#yourfile#")#
        </CFOUTPUT>.
    <CFELSE>
        <P>Your file was not found in this directory:
        <CFOUTPUT>#thisDirectory#</CFOUTPUT>.
    </CFIF>
</CFIF>

<!--- wait to have a string for searching defined --->
<CFIF IsDefined("form.myString") and IsDefined("form.type")>
    <!--- Now, test to see if the term was found --->
    A   <CFIF form.type is "find">case-sensitive
            <CFSET tag="find">
        <CFELSEIF form.type is "findNoCase">case-insensitive
            <CFSET tag="findNoCase">
```

```
        <CFELSEIF form.type is "findOneOf">find the first instance of any
                                        character
            <CFSET tag="findOneOf">
        </CFIF>
    test for your string "<CFOUTPUT>#form.MyString# </CFOUTPUT>" in
    "<CFOUTPUT>#Form.yourFile#</CFOUTPUT>"
</CFIF>

<!--- depending upon the action desired, perform different function --->
<CFIF IsDefined("tag")>
<CFSWITCH EXPRESSION="#tag#">
    <CFCASE value="find">
        <CFSET TheAction=Find(form.MyString, FORM.yourFile , 1)>
    </CFCASE>
    <CFCASE value="findNoCase">
        <CFSET TheAction=FindNoCase(form.MyString, FORM.yourFile, 1)>
    </CFCASE>
    <CFCASE value="findOneof">
        <CFSET TheAction=FindOneOf(form.MyString, FORM.yourFile, 1)>
    </CFCASE>
    <CFDEFAULTCASE>
        <CFSET TheAction=Find(form.MyString, FORM.YourFile, 1)>
    </CFDEFAULTCASE>
</CFSWITCH>
    <CFIF TheAction is not 0>
        found an instance of your string at index <CFOUTPUT>#theAction#
        </CFOUTPUT>
    <CFELSE>
        <P>No instance of your string was found.
    </CFIF>
</CFIF>

<H3>FindOneOf Example</H3>

<P>This example uses find, findnocase and findoneof to see
if a substring exists in a particular file in the directory
<CFOUTPUT>#GetDirectoryFromPath(thisPath)#</CFOUTPUT>.

<FORM ACTION="findoneof.cfm" METHOD="POST">
<P>Enter the name of a file in this directory <I><FONT SIZE="-1"></P>
<INPUT TYPE="Text" NAME="yourFile" VALUE="daysinyear.cfm">

<P>And a substring to search for:
<BR><INPUT TYPE="Text" size=25 NAME="myString" VALUE="daysinyear">
<P>Pick a search type:
    <SELECT NAME="type">
        <OPTION VALUE="find" SELECTED>Case-Sensitive
        <OPTION VALUE="findNoCase">Case-Insensitive
        <OPTION VALUE="findOneOf">Find first instance
    </SELECT>
<INPUT TYPE="Submit" NAME="" VALUE="start search">
</FORM>
</BODY>
</HTML>
```

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 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 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>
```

GetBaseTemplatePath

Returns the fully specified path of the base template.

See also GetClientVariablesList, FileExists and ExpandPath.

Syntax **GetBaseTemplatePath**()

Example

```
<!--- This example uses GetBaseTemplatePath to show
the template path of the current page --->
<HTML>
<HEAD>
<TITLE>
GetBaseTemplatePath Example
</TITLE>
</HEAD>

<BODY>
<H3>GetBaseTemplatePath Example</H3>

<P>The template path of the current page is:
<CFOUTPUT>#GetBaseTemplatePath()#</CFOUTPUT>

</BODY>
</HTML>
```

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>
```

GetCurrentTemplatePath

Returns the fully specified path of the template containing the call to this function.

See also GetBaseTemplatePath, FileExists and ExpandPath.

Syntax `GetCurrentTemplatePath()`

Usage This function differs from GetBaseTemplatePath in that it will return the template path of an included template if the call is made from a template included with a CFINCLUDE tag; whereas GetBaseTemplatePath returns the template path of the top-level template even when the call to GetBaseTemplatePath is actually made from an included template.

Example

```
<!--- This example uses GetCurrentTemplatePath to show
the template path of the current page --->
<HTML>
<HEAD>
<TITLE>
GetCurrentTemplatePath Example
</TITLE>
</HEAD>

<BODY>
<H3>GetCurrentTemplatePath Example</H3>

<P>The template path of the current page is:
<CFOUTPUT>#GetCurrentTemplatePath()#</CFOUTPUT>

</BODY>
</HTML>
```

GetDirectoryFromPath

Extracts the directory (with a \ (back slash)) 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)#
</CFOUTPUT>
```

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>

<BASEFONT FACE="Arial, Helvetica" SIZE=2>
<BODY  bgcolor="#FFFFD5">

<H3>GetFileFromPath Example</H3>

<p>This is a read-only example. Because file access is a sensitive area,
consider the security of your directory hierarchy prior to allowing
access to it.</p>

<!---<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:
            </CFIF>
    </CFIF>
<CFELSE>
<H3>Please enter a file name</H3>
</CFIF>
</CFOUTPUT>

<FORM action="getfilefrompathe.cfm" METHOD="post">
<H3>Enter the name of a file in this directory <I><FONT SIZE="-1">(try
expandpath.cfm)</FONT></I></H3>
```

```
<INPUT TYPE="Text" NAME="yourFile">
<INPUT TYPE="Submit" NAME="">
</FORM> --->

</BODY>
</HTML>
```

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>
```

GetProfileString

Returns the value of an entry in an initialization file or an empty string if the value does not exist. An *initialization file* assigns values to configuration variables, also known as entries, that need to be set when the system boots, the operating system comes up, or an application starts. An initialization file is distinguished from other files by its .ini suffix, for example, boot.ini, Win32.ini, and setup.ini.

See also SetProfileString.

Syntax

GetProfileString(*IniPath*, *Section, Entry*)

IniPath
Fully qualified path (drive, directory, filename, and extension) of the initialization file, for example, `C:\boot.ini`.

Section
The section of the initialization file from which you would like to extract information.

Entry
The name of the value that you would like to see.

Example

```
<!---This example uses GetProfileString to set the
     timeout value in an initialization file. --->

<HTML>
<HEAD>
<TITLE>GetProfileString Example</TITLE>
</HEAD>

<BODY  bgcolor="#FFFFD5">

<H3>GetProfileString Example</H3>

This example uses GetProfileString to get the value of timeout in an
initialization file. Enter the full path of your initialization file, and
submit the form.

<!---  This section of code checks to see if the form was submitted.
       If the form was submitted, this section gets the initialization
       path and timeout value of the path and timeout value specified
       in the form                                                    --->
<CFIF IsDefined("Form.Submit")>

    <CFSET IniPath=FORM.iniPath>
    <CFSET Section="boot loader">
    <CFSET timeout=GetProfileString(IniPath, Section, "timeout")>

    <H4>Boot Loader</H4>
```

```
    <!--- If you do not have an entry in an initialization file, nothing
          will be displayed --->

    <P>Timeout is set to: <CFOUTPUT>#timeout#</CFOUTPUT>.</P>

</CFIF>

<FORM ACTION="getprofilestring.cfm" METHOD="POST">
<HR size="2" color="#0000A0">
<table cellspacing="2" cellpadding="2" border="0">
<tr>
    <td>Full Path of Init File</td>
    <td><INPUT type="Text" name="IniPath" value="C:\myboot.ini"></td>
</tr>
<tr>
    <td><INPUT type="Submit" name="Submit" value="Submit"></td>
    <td></td>
</tr>
</table>

</FORM>
<HR size="2" color="#0000A0">

</BODY>
</HTML>
```

GetTempDirectory

Returns the full path name of a directory, including the trailing slash. The directory that is returned depends on the account under which ColdFusion is running as well as a variety of other factors. Before using this function in an application, test to see the directory it returns under your account.

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.

Note: For backward compatibility, GetTemplatePath is still supported. However, GetBaseTemplatePath supersedes this function, and should be used in place of it in all code written after the release of ColdFusion 4.0.

See also GetBaseTemplatePath, 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>
```

GetTimeZoneInfo

Returns a structure containing time zone information for the machine on which this function is executed. The structure contains four elements with the following keys.

- utcTotalOffset — total offset of the local time in minutes from UTC (Universal Coordinated Time). A plus sign (+) indicates that a time zone is west of UTC, such as all of the time zones in North and South America. A minus sign (-) indicates that a time zone is east of UTC, such as the time zones in Germany.
- utcHourOffset — offset in hours of local time from UTC.
- utcMinuteOffset — offset in minutes after the hours offset is taken into account. For North America, this will always be zero. However, for some countries that do not land exactly on the hour offset, the number will be between 0 and 60. For example, standard time in Adelaide, Australia has an offset of 9 hours and 30 minutes from UTC.
- isDSTOn — True if Daylight Savings Time (DST) is on in the host machine; False if DST is off.

See also DateConvert, CreateDateTime and DatePart.

Syntax **GetTimeZoneInfo()**

Examples

```
<HTML>
<HEAD>
<TITLE>GetTimeZoneInfo Example</TITLE>
</HEAD>

<BODY bgcolor=silver>

<H3>GetTimeZoneInfo Example</H3>
<!--- This example shows the use of GetTimeZoneInfo --->

<CFOUTPUT>
The local date and time are #now()#.
</CFOUTPUT>

<CFSET info=GetTimeZoneInfo()>
<CFOUTPUT>
<P>Total offset in minutes is #info.utcTotalOffset#.</P>
<P>Offset in hours is #info.utcHourOffset#.</P>
<P>Offset in minutes minus the offset in hours is
#info.utcMinuteOffset#.</P>
<P>Is Daylight Savings Time in effect? #info.isDSTOn#.</P>
</CFOUTPUT>

</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 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 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

```
<HTML>
<HEAD>
    <TITLE>HTMLCodeFormat</TITLE>
</HEAD

<H3>HTMLCodeFormat</H3>

<BODY>
<P>Often HTMLCodeFormat and HTMLEditFormat are used so that you can store
textbox information in the database in HTML form. Here is an example that
uses HTMLEditFormat to edit descriptive information found within the
CFExpress database. Because you may not want to change the database, the
CFQUERY statement that inserts the values into the database is commented
out.
</P>

<CFIF IsDefined("Form.gogo")>
<!-----------------------------------------------------------------------
    <CFQUERY NAME="PostMessage" DATASOURCE="CFExpress">
     INSERT INTO Messages(employee_id,thread_id,subject,posted,message)
     VALUES('#form.employee_id_float#','#form.thread_id_float#',
'#HTMLCodeFormat(Form.Subject)#',Now(),'#HTMLCodeFormat(Form.Message)#')
    </CFQUERY>
----------------------------------------------------------------------->
    <CFQUERY NAME="GetMessages" DATASOURCE="CFExpress">
        SELECT subject, posted, employee_id, thread_id
        From Messages
    </CFQUERY>
    <H2>New Messages</H2>
    <table cellspacing="2" cellpadding="2" border="0">
```

```
        <tr>
            <th>Employee ID</th>
            <th>Thread ID</th>
            <th>Subject</th>
            <th>Posted</th>
        </tr>
        <CFOUTPUT query="GetMessages">
        <tr>
            <td>#employee_id#</td>
            <td>#thread_id#</td>
            <td>#subject#</td>
            <td>#posted#</td>
        </tr>
        </CFOUTPUT>
    </table>

</CFIF>

<CFOUTPUT>
<FORM ACTION="htmlcodeformat.cfm" METHOD="POST">
<!-----------------------------------------------------------------------
To simplify this example, set employee_id to represent a new employee to
join the group, set thread_id to indicate a new thread, and pass these
values as hidden form values.
------------------------------------------------------------------------>
<CFQUERY NAME="GetIDs" DATASOURCE="CFExpress">
    SELECT employee_id, thread_id
    From Messages
</CFQUERY>
<CFLOOP QUERY="GetIDs">
    <CFSET employee_id = employee_id +1>
    <CFSET thread_id = thread_id +1>
</CFLOOP>
<INPUT TYPE="hidden" NAME="employee_id_float"  VALUE="#employee_id#">
<INPUT TYPE="hidden" NAME="thread_id_float"  VALUE="#thread_id#">
<TABLE>
<TR>
    <TD><B>Subject:</B></FONT></TD>
    <TD><INPUT TYPE="text" NAME="Subject" SIZE=35 VALUE=""></FONT></TD>
</TR>
<TR>
    <TD COLSPAN=2><B>Message</B><BR>
    <TEXTAREA NAME="Message" COLS=50 ROWS=10 WRAP="VIRTUAL"></TEXTAREA>
    </TD>
</TR>
</TABLE>

<INPUT TYPE="submit" NAME="gogo" VALUE="Post Message">
</FORM>
</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

```
<HTML>
...
<BODY>

<H3>HTMLEditFormat</H3>

<P>Often HTMLCodeFormat and HTMLEditFormat are used so that you can edit
information within a text box. Here is an example that uses
HTMLEditFormat to edit descriptive information found within the CFExpress
database. Because you may not want to change the database, the CFQUERY
statement that inserts the values into the database is commented out.
</P>

<CFIF IsDefined("Form.gogo")>
<!-----------------------------------------------------------------------
    <CFQUERY NAME="PostMessage" DATASOURCE="CFExpress">
     INSERT INTO Messages(employee_id,thread_id,subject,posted,message)
     VALUES('#form.employee_id_float#','#form.thread_id_float#',
'#HTMLEditFormat(Form.Subject)#',Now(),'#HTMLEditFormat(Form.Message)#')
    </CFQUERY>
------------------------------------------------------------------------>
    <CFQUERY NAME="GetMessages" DATASOURCE="CFExpress">
        SELECT subject, posted, employee_id, thread_id
        From Messages
    </CFQUERY>
```

```
    <H2>New Messages</H2>
    <table cellspacing="2" cellpadding="2" border="0">
        <tr>
            <th>Employee ID</th>
            <th>Thread ID</th>
            <th>Subject</th>
            <th>Posted</th>
        </tr>
        <CFOUTPUT query="GetMessages">
        <tr>
            <td>#employee_id#</td>
            <td>#thread_id#</td>
            <td>#subject#</td>
            <td>#posted#</td>
        </tr>
        </CFOUTPUT>
    </table>

</CFIF>

<CFOUTPUT>
<FORM ACTION="htmleditformat.cfm" METHOD="POST">
<!---------------------------------------------------------------------
To simplify this example, set employee_id to represent a new employee to
join the group, set thread_id to indicate a new thread, and pass these
values as hidden form values.
---------------------------------------------------------------------->
<CFQUERY NAME="GetIDs" DATASOURCE="CFExpress">
    SELECT employee_id, thread_id
    From Messages
</CFQUERY>
<CFLOOP QUERY="GetIDs">
    <CFSET employee_id = employee_id +1>
    <CFSET thread_id = thread_id +1>
</CFLOOP>
<INPUT TYPE="hidden" NAME="employee_id_float"  VALUE="#employee_id#">
<INPUT TYPE="hidden" NAME="thread_id_float"  VALUE="#thread_id#">
<TABLE>
<TR>
    <TD><B>Subject:</B></FONT></TD>
    <TD><INPUT TYPE="text" NAME="Subject" SIZE=35 VALUE=""></FONT></TD>
</TR>
<TR>
    <TD COLSPAN=2><B>Message</B><BR>
    <TEXTAREA NAME="Message" COLS=50 ROWS=10 WRAP="VIRTUAL"></TEXTAREA>
    </TD>
</TR>
</TABLE>

<INPUT TYPE="submit" NAME="gogo" VALUE="Post Message">
</FORM>
</cfoutput>

</BODY>
```

```
</HTML>
```

IIf

The function evaluates its *condition* as a Boolean. If the result is TRUE, it returns the value of *expression1*; otherwise, it returns the value of *expression2*.

Prior to using IIf, please read the Usage section carefully.

Syntax

IIf(*condition*, *string_expression1*, *string_expression2*)

condition
Any expression that can be evaluated as a Boolean.

expression1
Valid expression to be evaluated and returned if condition is TRUE.

expression2
Valid 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=expression1>
<CFELSE>
    <CFSET result=expression2>
</CFIF>
```

returning *result*.

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 evaluates on
expression 1 or expression 2 depending on the
Boolean outcome <I>(TRUE=run expression 1; FALSE=run
expression 2)</I>.
</P>

<CFSET number1=-10>
<CFSET number2=2>
<CFSET number3=number1 * number2>
<P>The result of the expression
IIf(number3 GT number1, number1 * number2, Abs(number1) * Abs(number2))
is:<BR>
```

```
<CFOUTPUT>
#IIf(number3 GT number1, number1 * number2, Abs(number1) * Abs(number2))#
</CFOUTPUT>
</P>

</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 REFind 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>
```

Boolean

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>
```

IsBoolean

Returns True if *value* can be converted to a Boolean; otherwise, False. ColdFusion considers the terms "True," "False," "Yes," and "No" and all integer and floating point values to be Boolean values.

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>#FORM.theTestValue#</CFOUTPUT>
      is Boolean</H3>
    <CFELSE>
    <H3>The expression <CFOUTPUT>#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 21st 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>#FORM.theTestValue#</CFOUTPUT>
      is a valid date</H3>
    <CFELSE>
    <H3>The string <CFOUTPUT>#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.

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>#FORM.theTestValue#</CFOUTPUT>
      is a Leap Year</H3>
    <CFELSE>
    <H3>The year value <CFOUTPUT>#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>#FORM.theTestValue#</CFOUTPUT>
      can be converted to a number</H3>
    <CFELSE>
    <H3>The string <CFOUTPUT>#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>#FORM.theTestValue#</CFOUTPUT>
          is a valid numeric date</H3>
        <CFELSE>
        <H3>The string <CFOUTPUT>#FORM.theTestValue#</CFOUTPUT>
          is not a valid numeric date</H3>
        </CFIF>
    <CFELSE>
    <H3>The string <CFOUTPUT>#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 QueryAddRow.

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" --->
<CFIF NOT IsDefined("getEmployees")>
    <CFSET getEmployees="#Now()#">
</CFIF>

<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="HRApp">
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>#Employee_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" --->
<CFIF NOT IsDefined("getEmployees")>
    <CFSET getEmployees="#Now()#">
</CFIF>

<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="HRApp">
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>#Employee_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 Structure Functions.

Syntax **IsStruct**(*variable*)

variable
Variable name.

Examples

```
<!---------------------------------------------------------------
This example illustrates the use of IsStruct and StructIsEmpty.
It assumes that you have created and assigned values to the fields of a
structure named EMPINFO within an include file. Sometimes when
you have included a file that contains variable declarations, you
may not know if a variable is a structure or a scalar value.
This code not only checks to see if EMPINFO is a structure, it
also checks to see if its fields have been assigned values.
---------------------------------------------------------------->
<CFIF IsStruct(EMPINFO)>
    <CFIF StructIsEmpty(EMPINFO)>
        <P>Error. EMPINFO is empty.</P>
    <CFELSE>
        <CFOUTPUT>
          <P>EMPINFO contains #StructCount(tempStruct)# fields.
        </CFOUTPUT>
    </CFIF>
<CFELSE>
    <P>Error. EMPINFO is not a structure.</P>
</CFIF>
...
```

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

```
<!--- First, query to get some values for our list --->
<HTML>
<HEAD>
<TITLE>ListAppend Example</TITLE>
</HEAD>

<H3>ListAppend Example</H3>

<!--- Find a list of users who wrote messages --->
<CFQUERY NAME="GetMessageUser" DATASOURCE="cfexpress">
SELECTSubject, 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>
<CFLOOP FROM="1" TO="#ListLen(temp)#" INDEX="Counter">
    <CFOUTPUT><LI>(#Counter#) SUBJECT: #ListGetAt(temp, Counter)#</
CFOUTPUT>
</CFLOOP>
```

```
<CFSET TempToo = ListAppend(temp, "More Applause for Express", ",")>
<UL>
<CFLOOP FROM="1" TO="#ListLen(temptoo)#" INDEX="Counter">
    <CFOUTPUT><LI>(#Counter#) SUBJECT: #ListGetAt(temptoo, Counter)#</
CFOUTPUT>
</CFLOOP>
</UL>
<P>Note that one new item "More Applause for Express" has been appended
to the list, but has not been added to the database.

</BODY>
</HTML>
```

ListChangeDelims

Returns *list* with all delimiter characters changed to *new_delimiter* string.

See also ListFirst and ListQualify.

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 ListFirst, ListLast, and ListRest --->
<HTML>
<HEAD>
<TITLE>ListFirst Example</TITLE>
</HEAD>

<BASEFONT FACE="Arial, Helvetica" SIZE=2>
<BODY  bgcolor="#FFFFD5">
<H3>ListFirst Example</H3>

<!--- Find a list of users who wrote messages --->
<CFQUERY NAME="GetMessages" DATASOURCE="CFExpress">
SELECTSubject, Posted, Message
FROM    Messages
</CFQUERY>

<CFSET temp = ValueList(GetMessages.Message)>
<!--- Show the first user in the list --->
<P>The first message in the list is
<UL>
<LI><CFOUTPUT>#ListFirst(temp)#</CFOUTPUT>
</UL>
<P>The rest of the messages are as follows:
<UL>
<li>
<CFOUTPUT>#ListChangeDelims(ListRest(temp), "<li>")#</CFOUTPUT>.
</UL>
<P>The last message in the list is
<UL>
<LI><CFOUTPUT>#ListLast(temp)#</CFOUTPUT>
</UL>
</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>

<BASEFONT FACE="Arial, Helvetica" SIZE=2>
<BODY  bgcolor="#FFFFD5">
<H3>ListContains Example</H3>

<CFIF IsDefined("form.letter") OR IsDefined("form.yourCity")>
    <!--- First, query to get some values for our list --->
    <CFQUERY NAME="GetLocations" DATASOURCE="HRAPP">
    SELECT Location
    FROMDepartments
    WHERE   <CFIF form.yourCity is "">Location LIKE '#form.letter#%'
<CFELSE>Location='#form.yourCity#'</CFIF>
    </CFQUERY>

    <CFSET tempList = ValueList(GetLocations.location)>
    <CFIF ListContains(tempList, form.yourCity) OR form.yourCity is ""
AND GetLocations.RecordCount Is Not 0>
    <P><CFIF form.yourCity is "" >You didn't specify a city, but using
<CFOUTPUT>#form.letter#</CFOUTPUT> to determine
        a location, we found the following information.
       </CFIF>
     <CFQUERY NAME="GetInformation" DATASOURCE="HRAPP">
        SELECT Department_Name, Location
        FROM Departments
        WHERE <CFIF form.yourCity is "">Location LIKE '#form.letter#%'
<CFELSE>Location LIKE '#form.yourCity#%'</CFIF>
        </CFQUERY>
```

```
        <UL>
        <CFOUTPUT query="GetInformation">
            <LI><B>A #Department_Name# Department is in <I>#location#</
I></B>
        </CFOUTPUT>
        </UL>
    <CFELSE>
    <P>Sorry, no Allaire offices in your city; however, Allaire has a
partnership program that you may
       wish to investigate <a href="http://www.allaire.com/partners/
index.cfm">Partnership Program</a>.
    </CFIF>
</CFIF>


<FORM ACTION="listcontains.cfm" METHOD="POST">
Letter City begins with:
<SELECT name="Letter">
    <OPTION value="A">A
    <OPTION value="B">B
    <OPTION value="C" SELECTED>C
    <CFSET temp ="D">
    <CFLOOP FROM="1" TO="25" INDEX="Counter">
    <OPTION value="<CFOUTPUT>#temp#">#Temp#</CFOUTPUT>
    <CFSET temp = CHR(Evaluate(Asc(temp) + 1))>
    </CFLOOP>
</SELECT>
<P>Name of your city: <INPUT TYPE="Text" NAME="YourCity" VALUE="">
<BR>(<I>hint: try "C" or "S", "Cambridge"</I>)
<INPUT TYPE="Submit" NAME="Find an Allaire office">
</FORM>
</BODY>
</HTML>
```

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>

<BASEFONT FACE="Arial, Helvetica" SIZE=2>
<BODY  bgcolor="#FFFFD5">
<H3>ListContainsNoCase Example</H3>

<CFIF IsDefined("form.letter") OR IsDefined("form.yourCity")>
    <!--- First, query to get some values for our list --->
    <CFQUERY NAME="GetLocations" DATASOURCE="HRAPP">
    SELECT Location
    FROMDepartments
    WHERE   <CFIF form.yourCity is "">Location LIKE '#form.letter#%'
<CFELSE>Location='#form.yourCity#'</CFIF>
    </CFQUERY>

    <CFSET tempList = ValueList(GetLocations.location)>
    <CFIF ListContainsNoCase(tempList, form.yourCity) OR form.yourCity is
"" AND GetLocations.RecordCount Is Not 0>
    <P><CFIF form.yourCity is "" >You didn't specify a city, but using
<CFOUTPUT>#form.letter#</CFOUTPUT> to determine
        a location, we found the following information.
       </CFIF>
     <CFQUERY NAME="GetInformation" DATASOURCE="HRAPP">
        SELECT Department_Name, Location
        FROM Departments
        WHERE <CFIF form.yourCity is "">Location LIKE '#form.letter#%'
<CFELSE>Location LIKE '#form.yourCity#%'</CFIF>
        </CFQUERY>
```

```
        <UL>
        <CFOUTPUT query="GetInformation">
            <LI><B>A #Department_Name# Department is in <I>#location#</
I></B>
        </CFOUTPUT>
        </UL>
    <CFELSE>
    <P>Sorry, no Allaire offices in your city; however, Allaire has a
partnership program that you may
       wish to investigate <a href="http://www.allaire.com/partners/
index.cfm">Partnership Program</a>.
    </CFIF>
</CFIF>


<FORM ACTION="listcontainsnocase.cfm" METHOD="POST">
Letter City begins with:
<SELECT name="Letter">
    <OPTION value="A">A
    <OPTION value="B">B
    <OPTION value="C" SELECTED>C
    <CFSET temp ="D">
    <CFLOOP FROM="1" TO="25" INDEX="Counter">
    <OPTION value="<CFOUTPUT>#temp#">#Temp#</CFOUTPUT>
    <CFSET temp = CHR(Evaluate(Asc(temp) + 1))>
    </CFLOOP>
</SELECT>
<P>Name of your city: <INPUT TYPE="Text" NAME="YourCity" VALUE="">
<BR>(<I>hint: try "C" or "S", "Cambridge"</I>)
<INPUT TYPE="Submit" NAME="Find an Allaire office">
</FORM>
</BODY>
</HTML>
```

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. The starting position in a list is denoted by the number 1, not 0.

delimiters
Set of delimiters used in list.

Examples

```
<!--- This example shows ListDeleteAt --->
<HTML>
<HEAD>
<TITLE>ListDeleteAt Example</TITLE>
</HEAD>

<BASEFONT FACE="Arial, Helvetica" SIZE=2>
<BODY  bgcolor="#FFFFD5">
<H3>ListDeleteAt Example</H3>

<!--- First, query to get some values for our list --->
<CFQUERY NAME="GetEmployees" DATASOURCE="HRApp">
SELECTLastName, FirstName
FROM Employees
</CFQUERY>

<CFSET temp = ValueList(GetEmployees.LastName)>
<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 no 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 --->
<HTML>
<HEAD>
<TITLE>ListFind Example</TITLE>
</HEAD>

<BASEFONT FACE="Arial, Helvetica" SIZE=2>
<BODY  bgcolor="#FFFFD5">
<H3>ListFind Example</H3>
<P>This example uses ListFind or ListFindNoCase to see if an exact string
exists in a list.

<FORM ACTION="listfind.cfm" METHOD="POST">
<P>Try changing the case in Leary's last name:
<BR><INPUT TYPE="Text" size=25 NAME="myString" VALUE="Hove">
<P>Pick a search type:
    <SELECT NAME="type">
        <OPTION VALUE="ListFind" SELECTED>Case-Sensitive
        <OPTION VALUE="ListFindNoCase">Case-Insensitive
    </SELECT>
<INPUT TYPE="Submit" NAME="" VALUE="Search Employee List">
</FORM>
<!--- wait to have a string for searching defined --->
<CFIF IsDefined("form.myString") and IsDefined("form.type")>

<CFQUERY Name="SearchEmpLastName" DATASOURCE="HRAPP">
SELECT FirstName, RTrim(LastName) AS LName,  StartDate
FROM Employees
</CFQUERY>

<CFSET myList = ValueList(SearchEmpLastName.LName)>
```

```
<!--- Is this case-sensitive or case-insensitive searching --->
<CFIF form.type is "ListFind">
    <CFSET temp = ListFind(myList, form.myString)>
    <CFIF temp is 0>
        <H3>An employee with that exact last name was not found</H3>
    <CFELSE>
        <CFOUTPUT>
        <P>Employee #ListGetAt(ValueList(SearchEmpLastName.FirstName),
temp)#
        #ListGetAt(ValueList(SearchEmpLastName.LName), temp)#
        started
#DateFormat(ListGetAt(ValueList(SearchEmpLastName.StartDate), temp))#.
        <P>This was the first employee found under this case-sensitive
last name search.
        </CFOUTPUT>
    </CFIF>
<CFELSE>
    <CFSET temp = ListFindNoCase(myList, form.myString)>
    <CFIF temp is 0>
        <H3>An employee with that exact last name was not found</H3>
    <CFELSE>
        <CFOUTPUT>
        <P>Employee #ListGetAt(ValueList(SearchEmpLastName.FirstName),
temp)#
        #ListGetAt(ValueList(SearchEmpLastName.LName), temp)#,
        started on
#DateFormat(ListGetAt(ValueList(SearchEmpLastName.StartDate), temp))#.
        <P>This was the first employee found under this case-insensitive
last name search.
        </CFOUTPUT>
    </CFIF>
</CFIF>
</CFIF>

</BODY>
</HTML>
```

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 --->
<HTML>
<HEAD>
<TITLE>ListFind Example</TITLE>
</HEAD>

<BASEFONT FACE="Arial, Helvetica" SIZE=2>
<BODY  bgcolor="#FFFFD5">
<H3>ListFind Example</H3>
<P>This example uses ListFind or ListFindNoCase to see if an exact string
exists
in a list

<FORM ACTION="listfind.cfm" METHOD="POST">
<P>Try changing the case in Leary's last name:
<BR><INPUT TYPE="Text" size=25 NAME="myString" VALUE="Hove">
<P>Pick a search type:
    <SELECT NAME="type">
        <OPTION VALUE="ListFind" SELECTED>Case-Sensitive
        <OPTION VALUE="ListFindNoCase">Case-Insensitive
    </SELECT>
<INPUT TYPE="Submit" NAME="" VALUE="Search Employee List">
</FORM>
<!--- wait to have a string for searching defined --->
<CFIF IsDefined("form.myString") and IsDefined("form.type")>

<CFQUERY Name="SearchEmpLastName" DATASOURCE="HRAPP">
SELECT FirstName, RTrim(LastName) AS LName,  StartDate
FROM Employees
</CFQUERY>
```

```
<CFSET myList = ValueList(SearchEmpLastName.LName)>
<!--- Is this case-sensitive or case-insensitive searching --->
<CFIF form.type is "ListFind">
    <CFSET temp = ListFind(myList, form.myString)>
    <CFIF temp is 0>
        <H3>An employee with that exact last name was not found</H3>
    <CFELSE>
        <CFOUTPUT>
        <P>Employee #ListGetAt(ValueList(SearchEmpLastName.FirstName),
temp)#
        #ListGetAt(ValueList(SearchEmpLastName.LName), temp)#
        started
#DateFormat(ListGetAt(ValueList(SearchEmpLastName.StartDate), temp))#.
        <P>This was the first employee found under this case-sensitive
last name search.
        </CFOUTPUT>
    </CFIF>
<CFELSE>
    <CFSET temp = ListFindNoCase(myList, form.myString)>
    <CFIF temp is 0>
        <H3>An employee with that exact last name was not found</H3>
    <CFELSE>
        <CFOUTPUT>
        <P>Employee #ListGetAt(ValueList(SearchEmpLastName.FirstName),
temp)#
        #ListGetAt(ValueList(SearchEmpLastName.LName), temp)#,
        started on
#DateFormat(ListGetAt(ValueList(SearchEmpLastName.StartDate), 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 ListQualify.

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>

<BASEFONT FACE="Arial, Helvetica" SIZE=2>
<BODY  bgcolor="#FFFFD5">
<H3>ListFirst Example</H3>

<!--- Find a list of users who wrote messages --->
<CFQUERY NAME="GetMessages" DATASOURCE="CFExpress">
SELECTSubject, Posted, Message
FROM    Messages
</CFQUERY>

<CFSET temp = ValueList(GetMessages.Message)>
<!--- Show the first user in the list --->
<P>The first message in the list is
<UL>
<LI><CFOUTPUT>#ListFirst(temp)#</CFOUTPUT>
</UL>
<P>The rest of the messages are as follows:
<UL>
<li>
<CFOUTPUT>#ListChangeDelims(ListRest(temp), "<li>")#</CFOUTPUT>.
</UL>
<P>The last message in the list is
<UL>
<LI><CFOUTPUT>#ListLast(temp)#</CFOUTPUT>
</UL>
</BODY>
</HTML>


<!--- This example shows ListFirst and ListLast --->
```

```
<HTML>
<HEAD>
<TITLE>ListFirst Example</TITLE>
</HEAD>

<BASEFONT FACE="Arial, Helvetica" SIZE=2>
<BODY  bgcolor="#FFFFD5">
<H3>ListFirst Example</H3>

<!--- Find a list of users who wrote messages --->
<CFQUERY NAME="GetEmployees" DATASOURCE="HRApp">
SELECTLastName, FirstName
FROM    Employees
</CFQUERY>

<CFSET first = ValueList(GetEmployees.FirstName)>
<CFSET last = ValueList(GetEmployees.LastName)>
<!--- Show the first user in the list --->
<P>The first employee in the list is <CFOUTPUT>#ListFirst(first)#
#ListFirst(last)#</CFOUTPUT>.
<P>The last name of the last employee in the list is
<CFOUTPUT>#ListLast(first)# #ListLast(last)#</CFOUTPUT>

</BODY>
</HTML>
```

ListGetAt

Returns the element at a given position.

See also ListFirst, ListLast, ListQualify, and ListSetAt.

Syntax **ListGetAt**(*list*, *position* [, *delimiters*])

list
List whose element is being retrieved.

position
Positive integer indicating the position of the element being retrieved.

delimiters
Set of delimiters used in *list*.

Usage The first position in a list is denoted by the number 1, not 0.

Examples

```
<!--- This example shows ListGetAt and ListLen --->
<HTML>
<HEAD>
<TITLE>ListGetAt Example</TITLE>
</HEAD>

<BODY>
<H3>ListGetAt Example</H3>
<!--- Find a list of employees in HR --->
<CFQUERY NAME="GetEmployees" DATASOURCE="HRApp">
SELECTLastName, FirstName
FROM Employees
Where Department_ID = 1
</CFQUERY>

<CFSET last=ValueList(GetEmployees.LastName)>
<CFSET first=ValueList(GetEmployees.FirstName)>

<!--- loop through the list and show it with ListGetAt --->
<H3>There are <CFOUTPUT>#ListLen(last)#</CFOUTPUT> people in the Training
Department. They are
</H3>
<UL>
<CFLOOP FROM="1" TO="#ListLen(first)#" INDEX="Counter">
    <CFOUTPUT><LI>#Counter#: #ListGetAt(first, Counter)#
     #ListGetAt(last, 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. The first position in a list is denoted by the number 1, not 0.

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. --->
<HTML>
<CFQUERY NAME="GetMessages" DATASOURCE="CFExpress">
SELECT Subject
FROM  Messages
</CFQUERY>

<CFSET temp=ValueList(GetMessages.Subject)>

<CFOUTPUT>
<P>The original list: #temp#
</CFOUTPUT>
<!--- Now, insert an item at position three. --->
<CFSET temp2=ListInsertAt(Temp, "3", "<B>my Inserted Value</B>", ",")>
<CFOUTPUT>
<P>The new list: #temp2#
</CFOUTPUT>
</HTML>
```

. . .

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>

<BASEFONT FACE="Arial, Helvetica" SIZE=2>
<BODY  bgcolor="#FFFFD5">
<H3>ListLast Example</H3>

<!--- Find a list of users who wrote messages --->
<CFQUERY NAME="GetMessages" DATASOURCE="CFExpress">
SELECTSubject, Posted, Message
FROM    Messages
</CFQUERY>

<CFSET temp = ValueList(GetMessages.Message)>
<!--- Show the first user in the list --->
<P>The first message in the list is
<UL>
<LI><CFOUTPUT>#ListFirst(temp)#</CFOUTPUT>
</UL>
<P>The rest of the messages are as follows:
<UL>
<li>
<CFOUTPUT>#ListChangeDelims(ListRest(temp), "<li>")#</CFOUTPUT>.
</UL>
<P>The last message in the list is
<UL>
<LI><CFOUTPUT>#ListLast(temp)#</CFOUTPUT>
</UL>
</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 employees in HR --->
<CFQUERY NAME="GetEmployees" DATASOURCE="HRApp">
SELECTLastName, FirstName
FROM Employees
Where Department_ID = 1
</CFQUERY>

<CFSET last=ValueList(GetEmployees.LastName)>
<CFSET first=ValueList(GetEmployees.FirstName)>

<!--- loop through the list and show it with ListGetAt --->
<H3>There are <CFOUTPUT>#ListLen(last)#</CFOUTPUT> people in the Training
Department. They are
</H3>
<UL>
<CFLOOP FROM="1" TO="#ListLen(first)#" INDEX="Counter">
    <CFOUTPUT><LI>#Counter#: #ListGetAt(first, Counter)#
     #ListGetAt(last, Counter)#
    </CFOUTPUT>
</CFLOOP>
</UL>

</BODY>
```

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 --->
<HTML>
<HEAD>
<TITLE>ListPrepend Example</TITLE>
</HEAD>

<H3>ListPrepend Example</H3>

<!--- Find a list of users who wrote messages --->
<CFQUERY NAME="GetMessageUser" DATASOURCE="cfexpress">
SELECTSubject, 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>
<CFLOOP FROM="1" TO="#ListLen(temp)#" INDEX="Counter">
    <CFOUTPUT><LI>(#Counter#) SUBJECT: #ListGetAt(temp, Counter)#</
CFOUTPUT>
</CFLOOP>
```

```
<CFSET TempToo = ListPrepend(temp, "Applause for Express", ",")>
<UL>
<CFLOOP FROM="1" TO="#ListLen(temptoo)#" INDEX="Counter">
    <CFOUTPUT><LI>(#Counter#) SUBJECT: #ListGetAt(temptoo, Counter)#</
CFOUTPUT>
</CFLOOP>
</UL>
<P>Note that one new item "Applause for Express" has been prepended to
the list, but has not been added to the database.

</BODY>
</HTML>
```

ListQualify

Returns a list with a qualifying character around each item in the list, such as double or single quotes.

See other list functions.

Syntax **ListQualify**(*list*, *qualifier* [, *delimiters*] [, *elements*])

list
Any list of items or a variable that names a list.

qualifier
The character that is to be placed at the beginning and end of each item in the list.

delimiters
Set of delimiters used in *list*.

elements
Either the keyword "ALL" or "CHAR." If you specify "ALL," the function qualifies all items in the list. If you specify "CHAR," the function qualifiers only items comprised of alphabetic characters; it does not qualify numeric items.

Usage The new list may not preserve all of the delimiters in the previous list.

Examples

```
<!--- This example uses ListQualify to put quotes around each
      employees full name --->
<HTML>
<HEAD>
<TITLE>ListQualify Example</TITLE>
</HEAD>

<BODY  bgcolor="#FFFFD5">

<CFQUERY Name="GetEmployeeNames" DATASOURCE="HRApp">
SELECT FirstName, LastName
FROM Employees
</CFQUERY>

<H3>ListQualify Example</H3>
<P>This example uses ListQualify to place the full names of the employees
found
in the query within quotation marks.</P>

<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>

<!--- sort that array descending alphabetically --->
<CFSET myAlphaArray=ArraySort(myArray, "textnocase")>

<!--- show the resulting array as a list --->
<CFSET myList=ArrayToList(myArray, ",")>

<CFOUTPUT>
    <P>The contents of the unqualified list are as follows:</P>
    #myList#
</CFOUTPUT>

<!--- show the resulting alphabetized array as a qualified
      list with single quotes around each full name.        --->
<CFSET qualifiedList1=ListQualify(myList,"'",",","CHAR")>

<!--- output the array as a list --->
<CFOUTPUT>
    <P>The contents of the qualified list are as follows:</P>
    <P>#qualifiedList1#</P>
</CFOUTPUT>

<!--- show the resulting alphabetized array as a qualified
      list with quotation marks around each full name. Note that
      we use &quot; to denote quotation marks because the
      quotation mark character is a control character.         --->
<CFSET qualifiedList2=ListQualify(myList,"&quot;",",","CHAR")>

<!--- output the array as a list --->
<CFOUTPUT>
    <P>The contents of the second qualified list are as follows:</P>
    <P>#qualifiedList2#</P>
</CFOUTPUT>
</BODY>
</HTML>
```

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>

<BASEFONT FACE="Arial, Helvetica" SIZE=2>
<BODY  bgcolor="#FFFFD5">
<H3>ListRest Example</H3>

<!--- Find a list of users who wrote messages --->
<CFQUERY NAME="GetMessages" DATASOURCE="CFExpress">
SELECTSubject, Posted, Message
FROM    Messages
</CFQUERY>

<CFSET temp = ValueList(GetMessages.Message)>
<!--- Show the first user in the list --->
<P>The first message in the list is
<UL>
<LI><CFOUTPUT>#ListFirst(temp)#</CFOUTPUT>
</UL>
<P>The rest of the messages are as follows:
<UL>
<li>
<CFOUTPUT>#ListChangeDelims(ListRest(temp), "<li>")#</CFOUTPUT>.
</UL>
<P>The last message in the list is
<UL>
<LI><CFOUTPUT>#ListLast(temp)#</CFOUTPUT>
</UL>
</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. The first position in a list is denoted by the number 1, not 0.

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>

<BASEFONT FACE="Arial, Helvetica" SIZE=2>
<BODY  bgcolor="#FFFFD5">
<H3>ListSetAt Example</H3>

<!--- Find a list of users who wrote messages --->
<CFQUERY NAME="GetMessageUser" DATASOURCE="cfexpress">
SELECTSubject, 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>
```

ListSort

Sorts and delimits the items in a list according to the specified sort type and sort order.

Syntax `ListSort(list, sort_type [, sort_order] [, delimiter ])`

list
List to be sorted. The items in the list must be separated by commas or otherwise delimited.

sort_type
The type of sort to be executed. You can specify any of the following sort types:

- Numeric - sorts numbers.
- Text - sorts text alphabetically.
- Textnocase - sorts text alphabetically. The case is ignored.

sort_order
The order to be followed. You can specify any of the following:

- Asc - (Default) Ascending sort order.
- Desc - Descending sort order.

delimiter
Specify the character(s) used to delimit elements in the list. Default is comma (,).

Example

```
<!--- This example shows ListSort--->
<HTML>
<HEAD>
<TITLE>ListSort Example</TITLE>
</HEAD>

<BASEFONT FACE="Arial, Helvetica" SIZE=2>
<BODY  bgcolor="#FFFFD5">

<BASEFONT FACE="Arial, Helvetica" SIZE=2>
<BODY  bgcolor="#FFFFD5">
<H3>ListSort Example</H3>

<!--- Find a list of users who wrote messages --->
<CFQUERY NAME="GetDepartments" DATASOURCE="HRApp">
SELECTDepartment_Name
FROM    Departments
</CFQUERY>

<CFSET myList = ValueList(GetDepartments.Department_Name)>
<P>Here is the unsorted list. </P>
```

```
<CFOUTPUT>
#myList#
</CFOUTPUT>
<P>Here is the list sorted alphabetically:</P>
<CFSET sortedList = ListSort(myList, "Text")>
<CFOUTPUT>
#sortedList#
</CFOUTPUT>

<P>Here is a numeric list that is to be sorted in descending order.</P>
<CFSET sortedNums = ListSort("12,23,107,19,1,65","Numeric", "Desc")>
<CFOUTPUT>
#sortedNums#
</CFOUTPUT>

<P>Here is a list that must be sorted numerically, since it contains both
negative and positive numbers, as well as decimal numbers. </P>

<CFSET sortedNums2 = ListSort("23.75;-34,471:100,-9745","Numeric",
"ASC", ";,:")>

<CFOUTPUT>
#sortedNums2#
</CFOUTPUT>

<P>Here is a list that is to be sorted alphabetically without
consideration of case.</P>

<CFSET sortedMix = ListSort("hello;123,HELLO,hello:jeans,-
345,887;ColdFusion:coldfusion", "TextNoCase", "ASC", ";,:")>

<CFOUTPUT>
#sortedMix#
</CFOUTPUT>

</BODY>
</HTML>
```

ListToArray

Converts the specified list into an array.

See also ArrayToList.

Syntax `ListToArray(`*list* [, *delimiter*]`)`

list
Name of the list variable that contains the elements to be used to build an array. You can define a list variable with a CFSET statement. The items in the list must be separated by commas or otherwise delimited.

delimiter
Specify the character(s) used to delimit elements in the list. Default is comma (,).

Example

```
<!--- This example shows ListToArray --->
<HTML>
<HEAD>
<TITLE>ListToArray Example</TITLE>
</HEAD>

<BODY>
<H3>ListToArray Example</H3>

<!--- Find the deparments in the database --->
<CFQUERY NAME="GetDepartments" DATASOURCE="HRApp">
SELECT Department_Name
FROM Departments
</CFQUERY>

<CFSET myList=ValueList(GetDepartments.Department_Name)>
<P>My list has <CFOUTPUT>#ListLen(myList)#</CFOUTPUT>
  items.
<CFSET myArrayList=ListToArray(myList)>
<P>My array has <CFOUTPUT>#ArrayLen(myArrayList)#
  </CFOUTPUT> elements.

</BODY>
</HTML>
```

ListValueCount

Returns the number of instances of a specified value in a list. The underlying search that finds the instances is case-sensitive.

See also ListValueCountNoCase.

Syntax `ListValueCount(list, value [, delimiters ])`

list
A list or the name of a list that is to be searched.

value
The string or number that the function is to find and count.

delimiter
Optional. Specify the character(s) used to delimit elements in the list. The default is a comma (,).

Example

```
<!--- This example uses ListValueCount to see how many
      employees are in a department --->
<HTML>
<HEAD>
<TITLE>ListValueCount Example</TITLE>
</HEAD>

<BASEFONT FACE="Arial, Helvetica" SIZE=2>

<BODY  bgcolor="#FFFFD5">

<CFQUERY Name="SearchByDepartment" DATASOURCE="HRApp">
SELECT Department_ID
FROM Employees
</CFQUERY>

<H3>ListValueCount Example</H3>
<P>This example uses ListValueCount to see how many employees
are in a department.

<FORM ACTION="listvaluecount.cfm" METHOD="POST">
<P>Select a department:</P>
    <SELECT NAME="departmentName">
            <OPTION VALUE="1">
                Training
            </OPTION>
            <OPTION VALUE="2">
                Marketing
            </OPTION>
            <OPTION VALUE="3">
                HR
```

```
            </OPTION>
            <OPTION VALUE="4">
                Sales
            </OPTION>
    </SELECT>
<INPUT TYPE="Submit" NAME="Submit" VALUE="Search Employee List">
</FORM>

<!--- wait to have a string for searching defined --->
<CFIF IsDefined("form.Submit") and IsDefined("form.departmentName")>
    <CFSET myList = ValueList(SearchByDepartment.Department_ID)>
    <CFSET numberInDepartment = ListValueCount(myList,
form.departmentName)>
    <CFOUTPUT>
    <P>There are #numberInDepartment# people in
    </CFOUTPUT>
    <CFIF FORM.DepartmentName Is "1">
    Training.
    <CFELSEIF FORM.DepartmentName Is "2">
    Marketing.
    <CFELSEIF FORM.DepartmentName Is "3">
    HR.
    <CFELSEIF FORM.DepartmentName Is "4">
    Sales.
    <CFELSE>
    Try again.
    </CFIF>
</CFIF>

</BODY>
</HTML>
```

ListValueCountNoCase

Returns the number of instances of a specified value in a list. The underlying search that finds the instances is not case-sensitive.

See also ListValueCount.

Syntax `ListValueCountNoCase(list, value [, delimiters ])`

list
A list or the name of a list that is to be searched.

value
The string or number that the function is to find and count.

delimiter
Optional. Specify the character(s) used to delimit elements in the list. The default is a comma (,).

Example

```
<!--- This example uses ListValueCountNoCase to see how many
      employees are in a department --->
<!--- This example uses ListValueCountNoCase to see how many
      employees are in a department --->
<HTML>
<HEAD>
<TITLE>ListValueCountNoCase Example</TITLE>
</HEAD>

<BODY  bgcolor="#FFFFD5">

<CFQUERY Name="SearchByDepartment" DATASOURCE="HRApp">
SELECT Department_ID
FROM Employees
</CFQUERY>

<H3>ListValueCountNoCase Example</H3>
<P>This example uses ListValueCountNoCase to see how many employees
are in a department.

<FORM ACTION="listvaluecountnocase.cfm" METHOD="POST">
<P>Select a department:</P>
    <SELECT NAME="departmentName">
            <OPTION VALUE="1">
                Training
            </OPTION>
            <OPTION VALUE="2">
                Marketing
            </OPTION>
            <OPTION VALUE="3">
                HR
```

```
            </OPTION>
            <OPTION VALUE="4">
                Sales
            </OPTION>
    </SELECT>
<INPUT TYPE="Submit" NAME="Submit" VALUE="Search Employee List">
</FORM>
<!--- wait to have a string for searching defined --->
<CFIF IsDefined("form.Submit") and IsDefined("form.departmentName")>
    <CFSET myList = ValueList(SearchByDepartment.Department_ID)>
    <CFSET numberInDepartment = ListValueCountNoCase(myList,
form.departmentName)>

    <CFOUTPUT>
    <P>There are #numberInDepartment# people in
    </CFOUTPUT>
    <CFIF FORM.DepartmentName Is "1">
    Training.
    <CFELSEIF FORM.DepartmentName Is "2">
    Marketing.
    <CFELSEIF FORM.DepartmentName Is "3">
    HR.
    <CFELSEIF FORM.DepartmentName Is "4">
    Sales.
    <CFELSE>
    Try again.
    </CFIF>
</CFIF>

</BODY>
</HTML>
```

LJustify

Returns left-justified *string* of the specified field length by adding padding to the right of the string.

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 --->
<CFIF NOT IsDefined("jstring")>
    <CFSET jstring="">
</CFIF>

<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>

<FORM 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.
<INPUT TYPE="hidden" NAME=”number_Required”>
<INPUT TYPE="hidden" NAME=”number_Float” Value=”You must enter a
number.”>
<INPUT TYPE="Text" NAME="number">
<INPUT TYPE="Submit" NAME="">
</FORM>

</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>

<FORM 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.
<INPUT TYPE="hidden" NAME=”number_Required”>
<INPUT TYPE="hidden" NAME=”number_Float” Value=”You must enter a
number.”>
<INPUT TYPE="Text" NAME="number">
<INPUT TYPE="Submit" NAME="">
</FORM>


</BODY>
</HTML>
```

LSCurrencyFormat

Returns a currency value using the locale convention. Default value is "local."

See, also, LSEuroCurrencyFormat.

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`

Locale	Format Type Output (Continued)
English (US)	Local: $100,000.00 International: USD100,000.00 None: 100,000.00
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

Locale	Format Type Output (Continued)
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 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>
```

LSEuroCurrencyFormat

Returns a currency value using the convention of the locale and the Euro as the currency symbol. Default value is "local."

Note: The locale is set with the SetLocale function.

See, also, LSParseEuroCurrency, LSCurrencyFormat, and SetLocale.

Syntax

LSEuroCurrencyFormat(*currency-number* [, *type*])

currency-number
The currency value.

type
Currency type. Valid arguments are:

- `none` — (For example, 10.00)
- `local` — (Default. For example, 10.00 €)
- `international` — (For example, EUR10.00)

Usage

The LSEuroCurrencyFormat function can display the Euro symbol (€) only on Euro-enabled computers, such as Windows NT 4.0 SP4, that have Euro-enabled fonts installed.

This function is similar to LSCurrencyFormat except that LSEuroCurrencyFormat displays the Euro currency symbol (€) or the international Euro sign (EUR) if you specify the type as local or international , respectively, and the Euro is the accepted currency of the locale.

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 €`
	`International: EUR100.000,00`
	`None: 100.000,00`

Dutch (Standard)	`Local: € 100.000,00`
	`International: EUR100.000,00`
	`None: 100.000,00`

Currency Output by Locale (Continued)	
Locale	**Format Type Output**
English (Australian)	Local: €100,000.00 International: EUR100,000.00 None: 100,000.00
English (Canadian)	Local: €100,000.00 International: EUR100,000.00 None: 100,000.00
English (New Zealand)	Local: €100,000.00 International: EUR100,000.00 None: 100,000.00
English (UK)	Local: €100,000.00 International: EUR100,000.00 None: 100,000.00
English (US)	Local: €100,000.00 International: EUR100,000.00 None: 100,000.00
French (Belgian)	Local: 100.000,00 € International: EUR100.000,00 None: 100.000,00
French (Canadian)	Local: 100 000,00 € International: EUR100 000,00 None: 100 000,00
French (Standard)	Local: 100 000,00 € International: EUR100 000,00 None: 100 000,00
French (Swiss)	Local: € 100'000.00 International: EUR100'000.00 None: 100'000.00
German (Austrian)	Local: € 100.000,00 International: EUR100.000,00 None: 100.000,00
German (Standard)	Local: 100.000,00 € International: EUR100.000,00 None: 100.000,00
German (Swiss)	Local: € 100'000.00 International: EUR100'000.00 None: 100'000.00
Italian (Standard)	Local: € 10.000.000 International: EUR10.000.000 None: 10.000.000

Currency Output by Locale (Continued)	
Locale	**Format Type Output**
Italian (Swiss)	Local: € 100'000.00 International: EUR100'000.00 None: 100'000.00
Norwegian (Bokmal)	Local: € 100 000,00 International: EUR100 000,00 None: 100 000,00
Norwegian (Nynorsk)	Local: € 100 000,00 International: EUR100 000,00 None: 100 000,00
Portuguese (Brazilian)	Local: €100.000,00 International: EUR100.000,00 None: 100.000,00
Portuguese (Standard)	Local: €100.000,00 International: EUR100.000,00 None: 100.000,00
Spanish (Mexican)	Local: €100,000.00 International: EUR100,000.00 None: 100,000.00
Spanish (Modern)	Local: 10.000.000 € International: EUR10.000.000 None: 10.000.000
Spanish (Standard)	Local: 10.000.000 € International: EUR10.000.000 None: 10.000.000
Swedish	Local: 100.000,00 € International: EUR100.000,00 None: 100.000,00

Example

```
<!--- This shows LSEuroCurrencyFormat --->
<HTML>
<HEAD>
<TITLE>LSEuroCurrencyFormat Example</TITLE>
</HEAD>

<BODY>
<H3>LSEuroCurrencyFormat Example</H3>

<P>LSEuroCurrencyFormat 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: #LSEuroCurrencyFormat(100000, "local")#<BR>
        International: #LSEuroCurrencyFormat(100000,
"international")#<BR>
        None: #LSEuroCurrencyFormat(100000, "none")#<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 from 0 to 29 are interpreted as 21st century values. Years 30 to 99 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) "

Number	Mask	Result
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*.

See, also, LSCurrencyFormat and LSParseEuroCurrency.

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.

See also ParseDateTime and SetLocale.

Syntax **LSParseDateTime**(*date-time-string*)

date-time-string
String being converted to date/time object. This string must be in a form that is readable in the current locale setting. By default the locale is set to English (US).

Usage When passing a date/time value for the English (US) locale, the date-time string can be in any of the following FORMs:

Date-Time Formats for the English (US) Locale	
Date-Time Composition	**Example**
dd mmmm yyyy	`"25 January 1999"`
hh:mm:ss	`"8:30:00"`
hh:mm:ss	`"20:30:00"`
mmmm dd, yyyy hh:mm:ss	`"January 25, 1999 8:30:00"`
hh:mm:ss mmm. dd, yyyy	`"8:30:00 Jan. 25, 1999"`
m/dd/yyyy hh:mm:ss	`"1/25/1999 8:30:00"`

Note that if you specify a year in the date, you should specify the full year.

If the date is formatted for a locale other than the English (US) locale, add or subtract the conversion time, depending on the locale. LSParseDateTime does not accept POP dates, nor does it have the capacity to convert dates to Greenwich Mean Time.

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:
    #LSParseDateTime("#LSDateFormat(Now())# #LSTimeFormat(Now())#")#<BR>
<Hr noshade>
    </CFOUTPUT>
</CFLOOP>
</BODY>
</HTML>
```

LSParseEuroCurrency

Converts a locale-specific currency string that contains the Euro symbol (€) or sign (EUR) to a number. Attempts conversion through each of the three default currency formats (none, local, international). Returns the number matching the value of *string*.

See, also, LSParseCurrency, LSEuroCurrencyFormat and SetLocale.

Syntax

LSParseEuroCurrency(*currency-string*)

currency-string
The locale-specific string you want to convert to a number.

Usage

The LSParseEuroCurrency function can read the Euro symbol (€) only on Euro-enabled computers, such as Windows NT 4.0 SP4, that have Euro-enabled fonts installed.

This function is similar to LSParseCurrency except that LSParseEuroCurrency parses only the Euro currency symbol (€) or the international Euro sign (EUR), not other currency symbols such as the dollar sign ($) or the pound sign (£).

Example

```
<!--- This example shows LSParseEuroCurrency --->
<HTML>
<HEAD>
<TITLE>LSParseEuroCurrency Example</TITLE>
</HEAD>

<BODY>
<H3>LSParseEuroCurrency Example</H3>

<P>LSParseEuroCurrency converts a locale-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: #LSEuroCurrencyFormat(123456, "local")#<BR>
        Currency Number:
        #LSParseEuroCurrency("EUR123456")#<BR>
        International: #LSEuroCurrencyFormat(123456,
"international")#<BR>

        None: #LSEuroCurrencyFormat(123456, "none")#<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 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 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 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

```
<!--- 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(yourDat"))>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)"

Number	Mask	Result
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=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>
```

ParseDateTime

Returns a date/time object from a string.

See also LSParseDateTime, IsDate and IsNumericDate.

Syntax **ParseDateTime**(*date-time-string* [, *pop-conversion*])

date-time-string
String being converted to date/time object.

pop-conversion
POP or STANDARD. If you specify POP, the function takes the date/time string passed from a POP mail server and converts it to GMT (Greenwich Mean Time) for the English (US) locale. If you specify STANDARD or nothing, the function provides no conversion. See the Note for more information about parsing date-time strings that are not from the English (US) locale.

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.

Note If the date is formatted for a locale other than the English (US) locale, you need to use the LSParseDateTime() function, then add or subtract the conversion time, depending on the locale. LSParseDateTime does not accept POP dates, nor does it have the capacity to convert dates to Greenwich Mean Time.

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>#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 ASin, 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 QuotedValueList, ValueList, and
PreserveSingleQuotes.
-------------------------------------------------------------->
<HTML>
<HEAD>
<TITLE>
PreserveSingleQuotes Example
</TITLE>
</HEAD>

<BODY>
<H3>PreserveSingleQuotes Example</H3>

<!--- use the contents of one query to create another
dynamically --->
<CFSET List="'Sales','Training','HR'">

<!--- first, get the department ids in our list --->
<CFQUERY NAME="GetDepartments" DATASOURCE="HRApp">
SELECT Department_ID
FROM Departments
WHERE Department_Name IN (#PreserveSingleQuotes(List)#)
</CFQUERY>

<!-----------------------------------------------------
now,  find the employees in the departments based
on the  previous query.
------------------------------------------------------>
<CFQUERY NAME="GetEmployees" DATASOURCE="HRApp">
SELECT Employee_ID, FirstName, LastName, Department_ID
FROM Employees
WHERE Department_ID IN (#ValueList(GetDepartments.Department_ID)#)
</CFQUERY>
<!-----------------------------------------------------
Now,  find the employees in the departments based
on the  previous query, using QuotedValueList.
------------------------------------------------------>
```

```
<CFQUERY NAME="GetEmployeeInfo" DATASOURCE="HRApp">
SELECT FirstName, LastName, Department_ID, StartDate
FROM Employees
WHERE LastName IN (#QuotedValueList(GetEmployees.LastName)#)
        AND FirstName IN (#QuotedValueList(GetEmployees.FirstName)#)

</CFQUERY>
<!--- now, output the results --->
<CFOUTPUT QUERY="GetEmployeeInfo" >
#FirstName##LastName#(#Department_ID# )<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 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 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>
```

QueryAddColumn

Adds a new column to a specified query and populates the column's rows with the contents of a one-dimensional array. Returns the query object with the additional column. Padding is added, if necessary, on the query columns to ensure that all columns have the same number of rows.

See also QueryNew, QueryAddRow and QuerySetCell.

Syntax

QueryAddColumn(*query*, *column-name*, *array-name*)

query
Name of a query that was created with QueryNew.

column-name
The name of the new column.

array-name
The name of the array whose elements are to populate the new column.

Usage

This function is particularly useful if you are an Oracle developer and would like to generate a query object from the arrays of output parameters which Oracle stored procedures can generate. Padding is added, if necessary, on the query columns to ensure that all columns have the same number of rows.

Example

```
<!--- This example shows the use of QueryAddColumn --->

<HTML>

<HEAD>
<TITLE>
QueryAddColumn Example
</TITLE>
</HEAD>

<BASEFONT FACE="Arial, Helvetica" SIZE=2>

<BODY  bgcolor="#FFFFD5">

<BODY>

<H3>QueryAddColumn Example</H3>

<P>This example adds three columns to a query object and then populates
the columns with the contents of three arrays.</P>

<P>After populating the query, the example shows, in tabular format, the
contents of the columns.</P>
```

```
<!--- make a new query --->
<CFSET myQuery=QueryNew("")>

<!--- create an array --->
<CFSET FastFoodArray=ArrayNew(1)>
<CFSET FastFoodArray[1]="French Fries">
<CFSET FastFoodArray[2]="Hot Dogs">
<CFSET FastFoodArray[3]="Fried Clams">
<CFSET FastFoodArray[4]="Thick Shakes">

<!--- add a column to the query --->
<CFSET nColumnNumber=QueryAddColumn(myQuery, "FastFood", FastFoodArray)>

<!--- create a second array --->
<CFSET FineCuisineArray=ArrayNew(1)>
<CFSET FineCuisineArray[1]="Lobster">
<CFSET FineCuisineArray[2]="Flambe">

<!--- add a second column to the query --->
<CFSET nColumnNumber2=QueryAddColumn(myQuery, "FineCuisine",
FineCuisineArray)>

<!--- create a third array --->
<CFSET HealthFoodArray=ArrayNew(1)>
<CFSET HealthFoodArray[1]="Bean Curd">
<CFSET HealthFoodArray[2]="Yogurt">
<CFSET HealthFoodArray[3]="Tofu">

<!--- add a third column to the query --->
<CFSET nColumnNumber3=QueryAddColumn(myQuery, "HealthFood",
HealthFoodArray)>

<table cellspacing="2" cellpadding="2" border="0">
<tr>
    <th align="left">Fast Food</th>
    <th align="left">Fine Cuisine</th>
    <th align="left">Health Food</th>
</tr>
<CFOUTPUT query="myQuery">
<tr>
    <td>#FastFood#</td>
    <td>#FineCuisine#</td>
    <td>#HealthFood#</td>
</tr>
</CFOUTPUT>
</table>

<P><B>Note:</B> Because there are fewer elements in the Fine Cuisine and
Health Food arrays, QueryAddColumn
added padding to the corresponding columns in the query.</P>

</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 QueryAddColumn 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="GetEmp" DATASOURCE="HRApp">
SELECT *
FROM Departments
</CFQUERY>

<P>The query object "GetEmp" has <CFOUTPUT>#GetEmp.RecordCount#
</CFOUTPUT> rows.<BR>

<CFSET CountRecs=#GetEmp.RecordCount# +1>

<!--- Add a row. --->
<CFSET Temp=QueryAddRow(GetEmp)>
<!--- Set the values of the cells in the roll. --->
<CFSET Temp=QuerySetCell(GetEmp, "Department_ID", CountRecs)>
<CFSET Temp=QuerySetCell(GetEmp, "Department_Name", "Engineering")>
<CFSET Temp=QuerySetCell(GetEmp, "Location", "Cambridge")>

<CFOUTPUT QUERY="GetEmp">
    #Department_ID#. #Department_Name#, #Location#<BR>
</CFOUTPUT>
</BODY>
</HTML>
```

QueryNew

Returns an empty query with a set of columns or an empty query with no columns. See Usage for more information.

See also QueryAddColumn, QueryAddRow and QuerySetCell.

Syntax **QueryNew**(*columnlist*)

columnlist
Comma-separated list of columns you want to add to the new query or an empty string.

Usage If you specify an empty string, you can add a new column to the query and populate its rows with the contents of a one-dimensional array using QueryAddColumn.

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:
<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 QueryAddColumn and QueryAddRow.

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="GetEmp" DATASOURCE="HRApp">
SELECT *
FROM Departments
</CFQUERY>

<P>The Query "GetEmp" has <CFOUTPUT>#GetEmp.RecordCount#
</CFOUTPUT> rows.<BR>

<CFSET CountRecs=#GetEmp.RecordCount# +1>
<CFSET Temp=QueryAddRow(GetEmp)>
<CFSET Temp=QuerySetCell(GetEmp, "Department_ID", CountRecs)>
<CFSET Temp=QuerySetCell(GetEmp, "Department_Name", "Engineering")>
<CFSET Temp=QuerySetCell(GetEmp, "Location", "Cambridge")>

<CFOUTPUT QUERY="GetEmp">
    #Department_ID#. #Department_Name#, #Location#<BR>
</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, ValueList, and
PreserveSingleQuotes.
-------------------------------------------------------------->
<HTML>
<HEAD>
<TITLE>
QuotedValueList Example
</TITLE>
</HEAD>

<BODY>
<H3>QuotedValueList Example</H3>

<!--- use the contents of one query to create another
dynamically --->
<CFSET List="'Sales','Training','HR'">

<!--- first, get the department ids in our list --->
<CFQUERY NAME="GetDepartments" DATASOURCE="HRApp">
SELECT Department_ID
FROM Departments
WHERE Department_Name IN (#PreserveSingleQuotes(List)#)
</CFQUERY>

<!-------------------------------------------------------
now,  find the employees in the departments based
on the  previous query.
-------------------------------------------------------->
<CFQUERY NAME="GetEmployees" DATASOURCE="HRApp">
SELECT Employee_ID, FirstName, LastName, Department_ID
FROM Employees
WHERE Department_ID IN (#ValueList(GetDepartments.Department_ID)#)
</CFQUERY>
```

```
<!-------------------------------------------------------
Now,  find the employees in the departments based
on the  previous query, using QuotedValueList.
-------------------------------------------------------->
<CFQUERY NAME="GetEmployeeInfo" DATASOURCE="HRApp">
SELECT FirstName, LastName, Department_ID, StartDate
FROM Employees
WHERE LastName IN (#QuotedValueList(GetEmployees.LastName)#)
        AND FirstName IN (#QuotedValueList(GetEmployees.FirstName)#)

</CFQUERY>
<!--- now, output the results --->
<CFOUTPUT QUERY="GetEmployeeInfo" >
#FirstName##LastName#(#Department_ID# )<BR>
</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>

<FORM ACTION="randRange.cfm" METHOD="POST">
<P>Enter a number to seed the randomizer:
<INPUT TYPE="hidden" NAME="MyInt_Range" Value="Min=1 Max=100000000">
<INPUT TYPE="hidden" NAME="MyInt_Required">
<INPUT TYPE="hidden" NAME="MyInt2_Range" Value="Min=1 Max=100000000">
<INPUT TYPE="hidden" NAME="MyInt2_Required">
<INPUT TYPE="Text" NAME="MyInt" VALUE="1">
<INPUT TYPE="Text" NAME="MyInt2" VALUE="500">
<P><INPUT TYPE="Submit" NAME="">
</FORM>
</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.

Returns the position and length of the first occurrence of a regular expression in a string if the *returnsubexpressions* parameter is set to True. See the description of the *returnsubexpressions* parameter and the "Usage" section for details.

See also Find, REFindNoCase, and REReplace.

Syntax

```
REFind(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
Optional. Starting position for the search. Default is 1.

returnsubexpressions
Optional. A Boolean value indicating whether a substring is returned. If you set this parameter to TRUE, the function returns a CFML structure composed of two arrays containing the position and length of the first substring that matches the criteria of the search. You can retrieve the position and length of the matching subexpression by using the keys "pos" and "len." If there are no occurrences of the regular expression, the "pos" and the "len" arrays each contain one element that has a value of zero. If you set this parameter to FALSE, a scalar value is returned indicating the position of the first occurrence of a regular expression. The default value of this parameter is FALSE.

Usage

In order to find multiple instances of a substring, you must call REFind more than once, each time with a different starting position. To determine the next starting position for the function, use the *returnsubexpressions* parameter and add the value returned in the position key to the value in the length key.

If you do not use parentheses in the regular expression, the *returnsubexpressions* parameter returns single element arrays that denote the position and length of the first match found in the string.

If you do use parentheses to denote subexpressions within the regular expression, the *returnsubexpressions* parameter returns the position and length of the first match of the regular expression in the first element of the respective arrays; the position and

length of the first instance of each subexpression within the regular expression are returned in subsequent elements of the arrays.

Examples

```
<!--- This example shows the use of REFind --->

<HTML>

<HEAD>
<TITLE>
REFind Example
</TITLE>
</HEAD>

<BODY>

<H3>REFind Example</H3>

<P>This example demonstrates the use of the REFind function with and
without the <i>returnsubexpressions</i> parameter set to True.</P>

If you do not use the <i>returnsubexpressions</i> parameter, REFind
returns the position of the first occurrence of a regular expression in a
string starting from the specified position.  Returns 0 if no occurrences
are found.
</P>

<P>REFind("a+c+", "abcaaccdd"):
<CFOUTPUT>#REFind("a+c+", "abcaaccdd")#</CFOUTPUT></P>
<P>REFind("a+c*", "abcaaccdd"):
<CFOUTPUT>#REFind("a+c*", "abcaaccdd")#</CFOUTPUT></P>
<P>REFind("[[:upper:]]", "abcaacCDD"):
<CFOUTPUT>#REFind("[[:upper:]]", "abcaacCDD")#</CFOUTPUT></P>
<P>REFind("[\?&]rep=", "report.cfm?rep=1234&u=5"):
<CFOUTPUT>#REFind("[\?&]rep=", "report.cfm?rep=1234&u=5")#</CFOUTPUT>
</P>
<!--- Set startPos to one; returnMatchedSubexpressions = TRUE --->
<HR size="2" color="#0000A0">

<P>If you do use the <i>returnssubexpression</i> parameter, REFind
returns the position and length of the first occurrence of a regular
expression in a string starting from the specified position. The position
and length variables are stored in a structure. In order to access
the position and length information, you must use the keys <i>pos</i> and
<i>len</i>, respectively.</P>

<CFSET teststring ="The cat in the hat hat came back!">
<P>The string in which the function is to search is:
<CFOUTPUT><b>#teststring#</b></CFOUTPUT>.</P>
<P>The first call to REFind to search this string is: <b>REFind("[A-Za-
z]+",testString,1,"TRUE")</b></P>
<P>This function returns a structure that contains two arrays: pos and
len.</P>
```

```
<P>In order to create this structure you can use a CFSET statement, for
example: </P>
&lt;CFSET st = REFind("[[:alpha:]]",testString,1,"TRUE")&gt;
<CFSET st = REFind("[[:alpha:]]",testString,1,"TRUE")>
<P>
    <CFOUTPUT>
   The number of elements in each array: #ArrayLen(st.pos)#.
   </CFOUTPUT>
</P>
<P><b>The number of elements in the pos and len arrays will always be one
if you do not use parentheses in the regular expression.</b></P>
<P>The value of st.pos[1] is: <CFOUTPUT>#st.pos[1]#.</CFOUTPUT></P>
<P>The value of st.len[1] is: <CFOUTPUT>#st.len[1]#.</CFOUTPUT></P>
<P>
    <CFOUTPUT>
    Substring is <b>[#Mid(testString,st.pos[1],st.len[1])#]</B>
    </CFOUTPUT>
</P>

<HR size="2" color="#0000A0">

<P>However, if you use parentheses in the regular expression, you will
find that the first element contains the position and length of the first
instance of the whole expression. The position and length of the first
instance of each parenthesized subexpression within will be included in
additional array elements.</P>

<P>For example:
&lt;CFSET st1 = REFind("([[:alpha:]])[
]+(\1)",testString,1,"TRUE")&gt;</P>

<CFSET st1 = REFind("([[:alpha:]]+)[ ]+(\1)",testString,1,"TRUE")>

<P>The number of elements in each array is
<CFOUTPUT>#ArrayLen(st1.pos)#</CFOUTPUT>.</P>

<P>First whole expression match; position is <CFOUTPUT>#st1.pos[1]#;
length is #st1.len[1]#; whole expression match is
<B>[#Mid(testString,st1.pos[1],st1.len[1])#]</B></CFOUTPUT></P>

<P>Subsequent elements of the arrays provide the position and length of
the first instance of each parenthesized subexpression therein.</P>
 <CFLOOP index="i" from="2" to="#ArrayLen(st1.pos)#">
    <p><CFOUTPUT>Position is #st1.pos[i]#; Length is #st1.len[i]#;
Substring is <B>[#Mid(testString,st1.pos[i],st1.len[i])#]</B></
CFOUTPUT></p>
</CFLOOP><BR>

</BODY>
</HTML>
```

REFindNoCase

Returns the position of the first occurrence of a regular expression in a string starting from the specified position if the *returnsubexpressions* parameter is not set to True. Returns 0 if no occurrences are found.The search is case-insensitive.

Returns the position and length of the first occurrence of a regular expression in a string if the *returnsubexpressions* parameter is set to True. See the description of the *returnsubexpressions* parameter and the "Usage" section for details.

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
Optional. Starting position for the search. Default is 1.

returnsubexpressions
Optional. A Boolean value indicating whether a substring is returned. If you set this parameter to TRUE, the function returns a CFML structure composed of two single-element arrays containing the position and length of the first substring that matches the criteria of the search. You can retrieve the position and length of the matching subexpression by using the keys "pos" and "len." If there are no occurrences of the regular expression, the "pos" and the "len" arrays each contain one element that has a value of zero. If you set this parameter to FALSE, a scalar value is returned indicating the position of the first occurrence of a regular expression. The default value of this parameter is FALSE.

Usage

In order to find multiple instances of a substring, you must call REFind more than once, each time with a different starting position. To determine the next starting position for the function, use the *returnsubexpressions* parameter and add the value returned in the position key to the value in the length key.

If you do not use parentheses in the regular expression, the *returnsubexpressions* parameter returns single element arrays that denote the position and length of the first match found in the string.

If you do use parentheses to denote subexpressions within the regular expression, the *returnsubexpressions* parameter returns the position and length of the first match of the regular expression in the first element of the respective arrays; the position and

length of the first instance of each subexpression within the regular expression are returned in subsequent elements of the arrays.

Examples

```
<!--- This example shows the use of REFindNoCase --->

<HTML>

<HEAD>
<TITLE>
REFindNoCase Example
</TITLE>
</HEAD>

<BODY>

<H3>REFindNoCase Example</H3>

<P>This example demonstrates the use of the REFindNoCase function with
and without the <i>returnsubexpressions</i> parameter set to True.</P>

<P>If you do not use the <i>returnsubexpressions</i> parameter,
REFindNoCase returns the position of the first occurrence of a regular
expression in a string starting from the specified position.  Returns 0
if no occurrences are found.
</P>

<P>REFindNoCase("a+c+", "abcaaccdd"):
<CFOUTPUT>#REFindNoCase("a+c+", "abcaaccdd")#</CFOUTPUT></P>
<P>REFindNoCase("a+c*", "abcaaccdd"):
<CFOUTPUT>#REFindNoCase("a+c*", "abcaaccdd")#</CFOUTPUT></P>
<P>REFindNoCase("[[:alpha:]]+", "abcaacCDD"):
<CFOUTPUT>#REFindNoCase("[[:alpha:]]+", "abcaacCDD")#</CFOUTPUT></P>
<P>REFindNoCase("[\?&]rep=", "report.cfm?rep=1234&u=5"):
<CFOUTPUT>#REFindNoCase("[\?&]rep=", "report.cfm?rep=1234&u=5")#</
CFOUTPUT>
</P>
<!--- Set startPos to one; returnMatchedSubexpressions = TRUE --->
<HR size="2" color="#0000A0">

<P>If you do use the <i>returnssubexpression</i> parameter, REFindNoCase
returns the position and length of the first occurrence of a regular
expression in a string starting from the specified position. The position
and length variables are stored in a structure. In order to access the
position and length information, you must use the keys <i>pos</i> and
<i>len</i>, respectively.</P>

<CFSET teststring ="The cat in the hat hat came back!">
<P>The string in which the function is to search is:
<CFOUTPUT><b>#teststring#</b></CFOUTPUT>.</P>
<P>The first call to REFindNoCase to search this string is:
<b>REFindNoCase("[[:alpha:]]+",testString,1,"TRUE")</b></P>
<P>This function returns a structure that contains two arrays: pos and
len.</P>
```

```
<P>In order to create this structure you can use a CFSET statement, for
example:</P>
&lt;CFSET st = REFindNoCase("[[:alpha:]]+",testString,1,"TRUE")&gt;
<CFSET st = REFindNoCase("[[:alpha:]]+",testString,1,"TRUE")>
<P>
    <CFOUTPUT>
    The number of elements in each array: #ArrayLen(st.pos)#.
    </CFOUTPUT>
</P>
<P><b>The number of elements in the pos and len arrays will always be one
if you do not use parentheses to denote subexpressions in the regular
expression.</b></P>
<P>The value of st.pos[1] is: <CFOUTPUT>#st.pos[1]#.</CFOUTPUT></P>
<P>The value of st.len[1] is: <CFOUTPUT>#st.len[1]#.</CFOUTPUT></P>
<P>
    <CFOUTPUT>
    Substring is <b>[#Mid(testString,st.pos[1],st.len[1])#]</B>
    </CFOUTPUT>
</P>
<HR size="2" color="#0000A0">

<P>However, if you use parentheses to denote subexpressions in the
regular expression, you will find that the first element contains the
position and length of the first instance of the whole expression. The
position and length of the first instance of each subexpression within
will be included in additional array
elements.</P>

<P>For example:
&lt;CFSET st1 = REFindNoCase("([[:alpha:]]+)[
]+(\1)",testString,1,"TRUE")&gt;</P>

<CFSET st1 = REFindNoCase("([[:alpha:]]+)[ ]+(\1)",testString,1,"TRUE")>

<P>The number of elements in each array is
<CFOUTPUT>#ArrayLen(st1.pos)#</CFOUTPUT>.</P>

<P>First whole expression match; position is
<CFOUTPUT>
#st1.pos[1]#; length is #st1.len[1]#;
whole expression match is
<B>[#Mid(testString,st1.pos[1],st1.len[1])#]</B>
</CFOUTPUT></P>

<P>Subsequent elements of the arrays provide the position and length of
the first instance of each parenthesized subexpression therein.</P>
 <CFLOOP index="i" from="2" to="#ArrayLen(st1.pos)#">
    <P><CFOUTPUT>Position is #st1.pos[i]#; Length is #st1.len[i]#;
    Substring is <B>[#Mid(testString,st1.pos[i],st1.len[i])#]</B>
    </CFOUTPUT></P>
</CFLOOP><BR>
</BODY>
</HTML>
```

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 and ReplaceNoCase.

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 and Replace.

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, ReplaceList, and Replace.

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. To justify the string, RJustify adds padding to the left of the string.

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 --->
<CFIF NOT IsDefined("jstring")>
    <CFSET jstring="">
</CFIF>

<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>
</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>
```

SetProfileString

Sets the value of a profile entry in an initialization file. This function returns an empty string if the operation succeeds or an error message if the operation fails.

See also GetProfileString.

Syntax **SetProfileString**(*IniPath, Section, Entry, Value*)

IniPath
Fully qualified path (drive, directory, filename, and extension) of the initialization file.

Section
The section of the initialization file in which the entry is to be set.

Entry
The name of the entry that is to be set.

Value
The value to which to set the entry.

Example

```
<!---This example uses SetProfileString to set the
     timeout value in an initialization file. --->

<HTML>
<HEAD>
<TITLE>SetProfileString Example</TITLE>
</HEAD>

<BODY  bgcolor="#FFFFD5">

<H3>SetProfileString Example</H3>

This example uses SetProfileString to set the value of timeout in an
initialization file. Enter the full path of your initialization file,
specify the timeout value, and submit the form.

<!---  This section of code checks to see if the form was submitted.
       If the form was submitted, this section sets the initialization
       path and timeout value to the path and timeout value specified
       in the form                                                    --->
<CFIF Isdefined("Form.Submit")>

    <CFSET IniPath=FORM.iniPath>
    <CFSET Section="boot loader">
    <CFSET MyTimeout=FORM.MyTimeout>
    <CFSET timeout=GetProfileString(IniPath, Section, "timeout")>

    <CFIF timeout Is Not MyTimeout>
```

```
        <CFIF MyTimeout Greater Than 0>
            <HR size="2" color="#0000A0">
            <P>Setting the timeout value to <CFOUTPUT>#MyTimeout#</
CFOUTPUT></P>
            <CFSET code=SetProfileString(IniPath, Section, "timeout",
MyTimeout)>
            <P>Value returned from SetProfileString: <CFOUTPUT>#code#</
CFOUTPUT></P>
        <CFELSE>
            <HR size="2"  color="red">
            <P>Timeout value should be greater than zero in order to
provide time for user response.</P>
            <HR size="2"  color="red">
        </CFIF>
    <CFELSE>
        <P>The timeout value in your initialization file is already
<CFOUTPUT>#MyTimeout#</CFOUTPUT>.</P>
    </CFIF>

    <CFSET timeout=GetProfileString(IniPath, Section, "timeout")>
    <CFSET default= GetProfileString(IniPath, Section, "default")>

    <H4>Boot Loader</H4>
    <P>Timeout is set to: <CFOUTPUT>#timeout#</CFOUTPUT>.</P>
    <P>Default directory is: <CFOUTPUT>#default#</CFOUTPUT>.</P>

</CFIF>

<FORM ACTION="setprofilestring.cfm" METHOD="POST">
<HR size="2" color="#0000A0">
<table cellspacing="2" cellpadding="2" border="0">
<tr>
    <td>Full Path of Init File</td>
    <td><INPUT type="Text" name="IniPath" value="C:\myboot.ini"></td>
</tr>
<tr>
    <td>Timeout</td>
    <td><INPUT type="Text" name="MyTimeout" value="30"></td>
</tr>
<tr>
    <td><INPUT type="Submit" name="Submit" value="Submit"></td>
    <td></td>
</tr>
</table>

</FORM>
<HR size="2" color="#0000A0">
</BODY>
</HTML>
```

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 ASin, 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
         </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, StructKeyArray, StructCount, StructKeyExists, and StructUpdate.

Syntax

StructClear(*structure*)

structure
Structure to be cleared.

Example

```
<!---------------------------------------------------------------------
This example shows how to use the StructNew, StructInsert and StructClear
functions. It first calls StructInsert to assign values to the
fields in the structure, and then uses StructClear to clear the
fields.
---------------------------------------------------------------------->
<HTML>
<HEAD>
<title>Add New Employees</title>
</HEAD>
<BODY>
<H1>Add New Employees</H1>
<!--- Establish parms for first time through  --->
<CFIF NOT IsDefined("FORM.firstname")>
  <CFSET FORM.firstname="">
</CFIF>
<CFIF NOT IsDefined("FORM.lastname")>
  <CFSET FORM.lastname="">
</CFIF>
<CFIF NOT IsDefined("FORM.email")>
  <CFSET FORM.email="">
</CFIF>
<CFIF NOT IsDefined("FORM.phone")>
  <CFSET FORM.phone="">
</CFIF>
<CFIF NOT IsDefined("FORM.department")>
  <CFSET FORM.department="">
</CFIF>

<CFIF FORM.firstname EQ "">
 <P>Please fill out the form.
<CFELSE>
 <CFSET employee=StructNew()>
 <CFSET retCode=StructInsert(employee, "firstname", FORM.firstname)>
 <CFSET retCode=StructInsert(employee, "lastname", FORM.lastname)>
 <CFSET retCode=StructInsert(employee, "email", FORM.email)>
 <CFSET retCode=StructInsert(employee, "phone", FORM.phone)>
 <CFSET retCode=StructInsert(employee, "department", FORM.department)>
<!---------------------------------------------------------------------
```

```
    Now clear the structure.
-------------------------------------------------------------------------->
 <CFSET retCode=StructClear(employee)>
</CFIF>
...
```

StructCopy

Returns a new structure with the all keys and values of the specified structure.

See also StructClear, StructDelete, StructFind, StructInsert, StructIsEmpty, StructKeyArray, StructKeyExists, and StructUpdate.

Syntax

StructCopy(*structure*)

structure
Structure to be copied.

Usage

This function throws an exception if *structure* does not exist.

Example

```
<!--- This example shows the use of StructCopy --->
<!-------------------------------------------------------------
This example assumes that you have created and assigned values
to the fields of a structure named EMPINFO.
-------------------------------------------------------------->
<CFIF StructIsEmpty(EMPINFO)>
        <CFOUTPUT>Error. EMPINFO is empty.</CFOUTPUT>
<CFELSE>
        <CFSET tempStruct=StructCopy(EMPINFO)>
        <CFQUERY NAME="AddEmployee" DATASOURCE="HRAPP">
          INSERT INTO Employees
             (Employee_ID, FirstName, LastName, Department_ID, StartDate,
Salary, Contract)
            VALUES
              <CFOUTPUT>
                (
                '#StructFind(tempStruct, "Employee_ID")#' ,
                '#StructFind(tempStruct, "FirstName")#' ,
                '#StructFind(tempStruct, "LastName")#' ,
                '#StructFind(tempStruct, "Department_ID")#' ,
                '#StructFind(tempStruct, "StartDate")#' ,
                '#StructFind(tempStruct, "Salary")#'
                '#StructFind(tempStruct, "Contract")#'
               )
              </CFOUTPUT>
          </CFQUERY>
</CFIF>
<HR>Employee Add Complete
<CFOUTPUT>
         <P>#StructCount(tempStruct)# columns added.
</CFOUTPUT>
...
```

StructCount

Returns the number of keys in the specified structure.

See also StructClear, StructDelete, StructFind, StructInsert, StructIsEmpty, StructKeyArray, StructKeyExists, and StructUpdate.

Syntax

StructCount(*structure*)

structure
Structure to be accessed.

Usage

This function throws an exception if *structure* does not exist.

Example

```
<!---------------------------------------------------------------------
 This example shows how to use the StructInsert, StructNew, and
 StructCount functions.
 ---------------------------------------------------------------------
>
<html>
<head>
<title>StructCount Function</title>
</head>

<BASEFONT FACE="Arial, Helvetica" SIZE=2>
<BODY  bgcolor="#FFFFD5">

<h3>StructCount Example</h3>
<h3>Add New Employees</h3>
<P>
<!---------------------------------------------------------------------
This example shows how to use the StructNew, StructInsert, and
StructCount
functions.
---------------------------------------------------------------------->
<HTML>
<HEAD>
<title>Add New Employees</title>
</HEAD>

<BODY>
<H3>Add New Employees</H3>
<!--- Establish parms for first time through  --->
<CFIF NOT IsDefined("FORM.firstname")>
    <CFSET FORM.firstname="">
</CFIF>
<CFIF NOT IsDefined("FORM.lastname")>
    <CFSET FORM.lastname="">
</CFIF>
<CFIF NOT IsDefined("FORM.email")>
```

```
    <CFSET FORM.email="">
</CFIF>
<CFIF NOT IsDefined("FORM.phone")>
    <CFSET FORM.phone="">
</CFIF>
<CFIF NOT IsDefined("FORM.department")>
    <CFSET FORM.department="">
</CFIF>

<CFIF FORM.firstname EQ "">
 <P>Please fill out the form.
<CFELSE>
    <CFSET employee=StructNew()>
    <CFSET retCode=StructInsert(employee, "firstname", FORM.firstname)>
    <CFSET retCode=StructInsert(employee, "lastname", FORM.lastname)>
    <CFSET retCode=StructInsert(employee, "email", FORM.email)>
    <CFSET retCode=StructInsert(employee, "phone", FORM.phone)>
    <CFSET retCode=StructInsert(employee, "department", FORM.department)>

    <CFOUTPUT>
    <P>There are #StructCount(employee)# fields in the new structure.</P>
    <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>
<!---------------------------------------------------------------------
    Write code to add the employee in the database.
---------------------------------------------------------------------->
</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>
```

StructDelete

Removes the specified item from the specified structure.

See also StructClear, StructFind, StructInsert, StructIsEmpty, StructKeyArray, StructCount, StructKeyExists, 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.
------------------------------------------------------------------------>
<html>
<head>
<title>StructDelete Function</title>
</head>
<BODY  bgcolor="#FFFFD5">
<h3>StructDelete Function</h3>
<h3>Delete a field from a structure</h3>

<!--- This example shows how to use the StructDelete function. --->

<h3>StructDelete Function</h3><P>
This example uses the StructInsert and StructDelete functions.
<!--- Establish parms for first time through  --->
<CFSET firstname="Mary">
<CFSET lastname="Torvath">
<CFSET email="mtorvath@allaire.com">
<CFSET phone="777-777-7777">
<CFSET department="Documentation">
<CFIF IsDefined("FORM.Delete")>
     <CFOUTPUT>
      <P>Field to be deleted: #form.field#</P>
     </CFOUTPUT>
      <CFSET employee=StructNew()>
      <CFSET rc=StructInsert(employee, "firstname", firstname)>
      <CFSET rc=StructInsert(employee, "lastname", lastname)>
      <CFSET rc=StructInsert(employee, "email", email)>
```

```
      <CFSET rc=StructInsert(employee, "phone", phone)>
      <CFSET rc=StructInsert(employee, "department", department)>
      <CFOUTPUT>employee is a structure: #IsStruct(employee)#
      <P>It has #StructCount(employee)# fields.</P>
      <CFSET rc=StructDelete(employee, form.field, "True")>
     <P>Did I delete the field "#form.field#"? The code indicates: #rc#
</P>
       <P>It has #StructCount(employee)# fields now.</P>
      </CFOUTPUT>
</CFIF>

<CFIF NOT IsDefined("FORM.Delete")>
<FORM action="structdelete.cfm" method="post">
        <P>Select the field to be deleted: 
<   SELECT name="field">
        <OPTION VALUE="firstname">first name
        <OPTION VALUE="lastname">last name
       <OPTION VALUE="email">email
        <OPTION VALUE="phone">phone
       <OPTION VALUE="department">department
   </SELECT>
   <INPUT type="submit" name="Delete" value="Delete">
 </FORM>
 </CFIF>
 </BODY>
 </HTML>
```

StructFind

Returns the value associated with the specified key in the specified structure.

See also StructClear, StructDelete, StructInsert, StructIsEmpty, StructKeyArray, StructCount, StructKeyExists, 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 example shows how to use the StructNew, StructInsert, and StructFind
functions.
---------------------------------------------------------------------->
<HTML>
<HEAD>
<title>Add New Employees</title>
</HEAD>

<BODY>
<H3>Add New Employees</H3>
<!--- Establish parms for first time through  --->
<CFIF NOT IsDefined("FORM.firstname")>
    <CFSET FORM.firstname="">
</CFIF>
<CFIF NOT IsDefined("FORM.lastname")>
    <CFSET FORM.lastname="">
</CFIF>
<CFIF NOT IsDefined("FORM.email")>
    <CFSET FORM.email="">
</CFIF>
<CFIF NOT IsDefined("FORM.phone")>
    <CFSET FORM.phone="">
</CFIF>
<CFIF NOT IsDefined("FORM.department")>
    <CFSET FORM.department="">
</CFIF>

<CFIF FORM.firstname EQ "">
 <P>Please fill out the form.
<CFELSE>
    <CFSET employee=StructNew()>
    <CFSET retCode=StructInsert(employee, "firstname", FORM.firstname)>
```

```
    <CFSET retCode=StructInsert(employee, "lastname", FORM.lastname)>
    <CFSET retCode=StructInsert(employee, "email", FORM.email)>
    <CFSET retCode=StructInsert(employee, "phone", FORM.phone)>
    <CFSET retCode=StructInsert(employee, "department", FORM.department)>

    <CFOUTPUT>
    <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>
<!---------------------------------------------------------------------
    Write code to add the employee in the database.
---------------------------------------------------------------------->
</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>
```

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, StructCount, StructFind, StructIsEmpty, StructKeyArray, StructDelete, StructKeyExists, 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 StructNew, StructInsert, and StructFind
functions.
---------------------------------------------------------------------->
<HTML>
<HEAD>
<title>Add New Employees</title>
</HEAD>

<BODY>
<H3>Add New Employees</H3>
<!--- Establish parms for first time through  --->
<CFIF NOT IsDefined("FORM.firstname")>
    <CFSET FORM.firstname="">
</CFIF>
<CFIF NOT IsDefined("FORM.lastname")>
    <CFSET FORM.lastname="">
</CFIF>
<CFIF NOT IsDefined("FORM.email")>
    <CFSET FORM.email="">
</CFIF>
<CFIF NOT IsDefined("FORM.phone")>
    <CFSET FORM.phone="">
```

```
</CFIF>
<CFIF NOT IsDefined("FORM.department")>
    <CFSET FORM.department="">
</CFIF>

<CFIF FORM.firstname EQ "">
 <P>Please fill out the form.
<CFELSE>
    <CFSET employee=StructNew()>
    <CFSET retCode=StructInsert(employee, "firstname", FORM.firstname)>
    <CFSET retCode=StructInsert(employee, "lastname", FORM.lastname)>
    <CFSET retCode=StructInsert(employee, "email", FORM.email)>
    <CFSET retCode=StructInsert(employee, "phone", FORM.phone)>
    <CFSET retCode=StructInsert(employee, "department", FORM.department)>

    <CFOUTPUT>
    <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>
<!-----------------------------------------------------------------------
    Write code to add the employee in the database.
------------------------------------------------------------------------>
</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, StructKeyArray, StructCount, StructKeyExists, and StructUpdate.

Syntax

StructIsEmpty(*structure*)

structure
Structure to be tested.

Usage

This function throws an exception if *structure* does not exist.

Example

```
<!--- This example illustrates usage of StructIsEmpty. --->
<!-------------------------------------------------------------------
This example assumes that you have created and assigned values
to the fields of a structure named EMPINFO.
-------------------------------------------------------------------->
<CFIF StructIsEmpty(EMPINFO)>
        <CFOUTPUT>Error. EMPINFO is empty.</CFOUTPUT>
<CFELSE>
        tempStruct=StructCopy(EMPINFO)
        <CFQUERY NAME="AddEmployee" DATASOURCE="HRApp">
          INSERT INTO Employees
              (FirstName, LastName, StartDate, Salary, Contract,
               Department_ID)
            VALUES
              <CFOUTPUT>
                (
                  '#StructFind(tempStruct, "FirstName")#' ,
                  '#StructFind(tempStruct, "LastName")#' ,
                  '#StructFind(tempStruct, "StartDate")#' ,
                  '#StructFind(tempStruct, "Salary")#' , (
                  '#StructFind(tempStruct, "Contract")#' ,
                  '#StructFind(tempStruct, "Department_ID")#'
                )
            </CFOUTPUT>
        </CFQUERY>
</CFIF>
<HR>Employee Add Complete
<CFOUTPUT>
         <P>#StructCount(tempStruct)# columns added.
</CFOUTPUT>
...
```

StructKeyArray

Returns an array of the keys in the specified ColdFusion structure.

See also StructClear, StructDelete, StructFind, StructInsert, StructIsEmpty, StructKeyList, StructKeyExists, StructCount, and StructUpdate.

Syntax

StructKeyArray(*structure*)

structure
Structure from which the list of keys is to be extracted.

Usage

The array of keys returned by StructKeyArray is not in any particular order. In order to sort keys alphabetically or numerically, use ArraySort.

Note that this function throws an exception if *structure* does not exist.

Example

```
<!--- This example shows how to use the StructKeyArray
      function to copy the keys from a specified structure to an array.
      It also uses the StructNew function to create the structure
      and fills its fields with the information the user types
      into the corresponding form fields. --->
<HTML>
<HEAD>
<title>StructKeyArray Function</title>
</HEAD>

<BODY  bgcolor="#FFFFD5">

<H3>StructKeyArray Example</H3>
<H3>Extracting the Keys from the Employee Structure</H3>

<!-----------------------------------------------------------------
This section of code creates the new structure and checks to
see if the submit button has been pressed.  If it has been
pressed, the code defines fields in the employee structure
with what the user has entered from the form.
------------------------------------------------------------------>

<CFSET employee=StructNew()>
<CFIF Isdefined("Form.Submit")>
    <CFIF Form.Submit is "OK">
        <CFSET employee.firstname=FORM.firstname>
        <CFSET employee.lastname=FORM.lastname>
        <CFSET employee.email=FORM.email>
        <CFSET employee.phone=FORM.phone>
        <CFSET employee.company=FORM.company>
    </CFIF>
</CFIF>

<P>
```

```
This example uses the StructNew function to create a structure
that supplies employee information.  The data structure is called
"employee" and its fields are filled with the contents of the
following form. After you have entered employee information into the
structure, the example uses the <b>StructKeyArray</b> function to copy
all of the keys from the structure into an array.
</P>

<HR size="2" color="#0000A0">

<FORM action="structkeyarray.cfm" method="post">
<table cellspacing="2" cellpadding="2" border="0">
    <tr>
    <td>First Name:</td>
    <td><INPUT name="firstname" type="text" value="" hspace="30"
maxlength="30"></td>
    </tr>
    <tr>
    <td>Last Name:</td>
    <td><INPUT name="lastname" type="text" value="" hspace="30"
maxlength="30"></td>
    </tr>
    <tr>
    <td>EMail</td>
    <td><INPUT name="email" type="text" value="" hspace="30"
maxlength="30"></td>
    </tr>
    <tr>
    <td>Phone:</td>
    <td><INPUT name="phone" type="text" value="" hspace="20"
maxlength="20"></td>
    </tr>
    <tr>
    <td>Company:</td>
    <td><INPUT name="company" type="text" value="" hspace="30"
maxlength="30"></td>
    </tr>
    <tr>
    <td><INPUT type="submit" name="submit" value="OK"></td>
    <td><b>After you submit the FORM, scroll down to see the array.</b>
    </td>
    </tr>
</table>
</FORM>

<CFIF NOT StructISEmpty(employee)>
    <HR size="2" color="#0000A0">
    <CFSET keysToStruct=StructKeyArray(employee)>
    <CFLOOP index="i" from="1" to="#ArrayLen(keysToStruct)#">
        <p><CFOUTPUT>Key#i# is #keysToStruct[i]#</CFOUTPUT></p>
        <p><CFOUTPUT>Value#i# is #employee[keysToStruct[i]]#</CFOUTPUT>
        </p>
    </CFLOOP>
</CFIF>
```

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

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, StructKeyArray, 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 example illustrates the use of IsStruct and StructIsEmpty.
It assumes that you have created and assigned values to the fields of a
structure named EMPINFO within an include file. Sometimes when
you have included a file that contains variable declarations, you
may not know if a variable is a structure or a scalar value.
This code not only checks to see if EMPINFO is a structure, it
also checks to see if its fields have been assigned values, and
checks for the existence of a particular field.
---------------------------------------------------------------->
<CFIF IsStruct(EMPINFO)>
    <CFIF StructIsEmpty(EMPINFO)>
        <P>Error. EMPINFO is empty.</P>
    <CFELSE>
        <CFOUTPUT>
          <P>EMPINFO contains #StructCount(EMPINFO)# fields.
        </CFOUTPUT>
        <CFIF StructKeyExists(EMPINFO, phone)>
        <!-------------------------------------------------------
        Use CFQUERY to find out if this is a new number.
        ------------------------------------------------------->
        </CFIF>
    </CFIF>
<CFELSE>
    <P>Error. EMPINFO is not a structure.</P>
</CFIF>
...
```

StructKeyList

Returns the list of keys that are in the specified ColdFusion structure.

See also StructKeyArray, StructClear, StructDelete, StructFind, StructInsert, StructIsEmpty, StructCount, and StructUpdate.

Syntax

StructKeyList(*structure*, [*delimiter*])

structure
Structure from which the list of keys are to be extracted.

delimiter
Optional. The value of this parameter indicates the character that will separate the keys in the list. By default, a comma (,) is used.

Usage

The list of keys returned by StructKeyList is not in any particular order. In order to sort keys alphabetically or numerically, use ListSort.

Note that this function throws an exception if *structure* does not exist.

Example

```
<!---------------------------------------------------------------------
This example shows how to use the StructKeyList function to list the keys
within a specified structure. It also uses the StructNew function to
create the structure and fills its fields with the information the user
type into the corresponding form fields.
---------------------------------------------------------------------->

<!---------------------------------------------------------------------
This section of code creates the new structure and checks to see if the
submit button has been pressed.  If it has been pressed, the code defines
fields in the employee structure with what the user has entered from the
form.
---------------------------------------------------------------------->
<CFSET employee=StructNew()>
<CFIF Isdefined("Form.Submit")>
    <CFIF Form.Submit is "OK">
        <CFSET employee.firstname=FORM.firstname>
        <CFSET employee.lastname=FORM.lastname>
        <CFSET employee.email=FORM.email>
        <CFSET employee.phone=FORM.phone>
        <CFSET employee.company=FORM.company>
    </CFIF>
</CFIF>

<HTML>
<HEAD>
    <TITLE>StructKeyList Function</TITLE>
</HEAD>
<BODY  bgcolor="#FFFFD5">
```

```
<H3>StructKeyList Function</H3>
<H3>Listing the Keys in the Employees Structure</H3>
<P>
This example uses the StructNew function to create a structure
that supplies employee information.  The data structure is called
"employee" and its fields are filled with the contents of the following
form.
</P>
<P>
After you have entered employee information into the structure, the
example uses the <b>StructKeyList</b> function to list all of the keys in
the structure.
</P>
<P>
This code does not show how to insert this information into a database.
See CFQUERY for more information
about database insertion.

<HR size="2" color="#0000A0">
<FORM action="structkeylist.cfm" method="post">
<table cellspacing="2" cellpadding="2" border="0">
    <tr>
    <td>First Name:</td>
    <td><INPUT name="firstname" type="text" value="" hspace="30"
maxlength="30"></td>
    </tr>
    <tr>
    <td>Last Name:</td>
    <td><INPUT name="lastname" type="text" value="" hspace="30"
maxlength="30"></td>
    </tr>
    <tr>
    <td>EMail</td>
    <td><INPUT name="email" type="text" value="" hspace="30"
maxlength="30"></td>
    </tr>
    <tr>
    <td>Phone:</td>
    <td><INPUT name="phone" type="text" value="" hspace="20"
maxlength="20"></td>
    </tr>
    <tr>
    <td>Company:</td>
    <td><INPUT name="company" type="text" value="" hspace="30"
maxlength="30"></td>
    </tr>
    <tr>
    <td><INPUT type="submit" name="submit" value="OK"></td>
    <td><b>After you submit the FORM, scroll down to see the list.</b></
td>
    </tr>
</table>
</FORM>
```

```
<CFIF NOT StructISEmpty(employee)>
    <HR size="2" color="#0000A0">
    <CFSET keysToStruct=StructKeyList(employee,"<LI>")>
    <P>Here are the keys to the structure:</P>
    <UL>
    <LI>
    <CFOUTPUT>#keysToStruct#</CFOUTPUT>
    </UL>

    <P>
    If these fields are correct, we can process your new employee
    information. If they are not correct, you should consider rewriting
    your application.
    </P>
</CFIF>

</BODY>
</HTML>
```

StructNew

Returns a new structure.

See also StructClear, StructDelete, StructFind, StructInsert, StructIsEmpty, StructKeyArray, StructCount, and StructUpdate.

Syntax `StructNew()`

Example

```
<!---------------------------------------------------------------------
This example shows how to use the StructNew, StructInsert, and StructFind
functions.
---------------------------------------------------------------------->
<HTML>
<HEAD>
<title>Add New Employees</title>
</HEAD>

<BODY>
<H1>Add New Employees</H1>
 <CFSET employee= StructNew()>

<!--- Establish parameters for first time through  --->
<CFIF NOT IsDefined("FORM.firstname")>
  <CFSET FORM.firstname="">
</CFIF>
<CFIF NOT IsDefined("FORM.lastname")>
  <CFSET FORM.lastname="">
</CFIF>
<CFIF NOT IsDefined("FORM.email")>
  <CFSET FORM.email="">
</CFIF>
<CFIF NOT IsDefined("FORM.phone")>
  <CFSET FORM.phone="">
</CFIF>
<CFIF NOT IsDefined("FORM.department")>
  <CFSET FORM.department="">
</CFIF>

<CFIF FORM.firstname EQ "">
 <P>Please fill out the form.
<CFELSE>
 <CFSET retCode=StructInsert(employee, "firstname", FORM.firstname)>
 <CFSET retCode=StructInsert(employee, "lastname", FORM.lastname)>
 <CFSET retCode=StructInsert(employee, "email", FORM.email)>
 <CFSET retCode=StructInsert(employee, "phone", FORM.phone)>
 <CFSET retCode=StructInsert(employee, "department", FORM.department)>

 <CFOUTPUT>
 <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>

</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, StructKeyArray, StructCount, and StructKeyExists.

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 example illustrates the use of StructUpdate.
It assumes that you have created and assigned values to the fields of a
structure named EMPINFO within an include file.
---------------------------------------------------------------------->

<CFIF StructIsEmpty(EMPINFO)>
        <CFOUTPUT>Error. No employee data was passed.</CFOUTPUT>
<CFELSEIF StructFind(EMPINFO, "department") EQ "">
        <CFSET rCode=StructUpdate(EMPINFO, "department", "Unassigned")>
<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>#FORM.theTestValue#</CFOUTPUT>
      can be converted to a number:
    <CFOUTPUT>#Val(FORM.theTestValue)#</CFOUTPUT></H3>
    <CFELSE>
    <H3>The beginning of the string <CFOUTPUT>#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 QuotedValueList, ValueList, and
PreserveSingleQuotes.
-------------------------------------------------------------->
<HTML>
<HEAD>
<TITLE>
ValueList Example
</TITLE>
</HEAD>

<BODY>
<H3>ValueList Example</H3>

<!--- use the contents of one query to create another
dynamically --->
<CFSET List="'Sales','Training','HR'">

<!--- first, get the department ids in our list --->
<CFQUERY NAME="GetDepartments" DATASOURCE="HRApp">
SELECT Department_ID
FROM Departments
WHERE Department_Name IN (#PreserveSingleQuotes(List)#)
</CFQUERY>

<!-------------------------------------------------------
now,  find the employees in the departments based
on the  previous query.
-------------------------------------------------------->
<CFQUERY NAME="GetEmployees" DATASOURCE="HRApp">
SELECT Employee_ID, FirstName, LastName, Department_ID
FROM Employees
WHERE Department_ID IN (#ValueList(GetDepartments.Department_ID)#)
</CFQUERY>
```

```
<!-------------------------------------------------------
Now,  find the employees in the departments based
on the  previous query, using QuotedValueList.
-------------------------------------------------------->
<CFQUERY NAME="GetEmployeeInfo" DATASOURCE="HRApp">
SELECT FirstName, LastName, Department_ID, StartDate
FROM Employees
WHERE LastName IN (#QuotedValueList(GetEmployees.LastName)#)
        AND FirstName IN (#QuotedValueList(GetEmployees.FirstName)#)

</CFQUERY>
<!--- now, output the results --->
<CFOUTPUT QUERY="GetEmployeeInfo" >
#FirstName##LastName#(#Department_ID# )<BR>
</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 from 0 to 29 are interpreted as 21st century values. Years 30 to 99 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.

Note: When within the CFQUERY tag, the WriteOutput function does not output to the current page, but instead writes to the current SQL statement. Do not use WriteOutput within CFQUERY.

Syntax

WriteOutput(*string*)

string
Text to be appended to the page output stream.

Examples

```
...
<CFSET rCode=employee=StructNew()>
<CFSET rCode=StructInsert(employee, "firstname", "#FORM.firstname#")>
<CFSET rCode=StructInsert(employee, "lastname", "#FORM.lastname#")>
<CFSET rCode=StructInsert(employee, "email", "#FORM.email#")>
<CFSET rCode=StructInsert(employee, "phone", "#FORM.phone#")>
<CFSET rCode=StructInsert(employee, "department", "#FORM.department#")>
<CFSET rCode=WriteOutput("About to add " & "#FORM.firstname#" & " " &
      "#FORM.lastname#")>
...
```

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 from 0 to 29 are interpreted as 21st century values. Years 30 to 99 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 3

ColdFusion Operators

This chapter describes the CFML operators, shorthand notation, and operator precedence.

Contents

Using Operators to build expressions

Operators fall into four category types:

- Arithmetic - perform operations on numeric values.
- String - perform operations on text values.
- Comparison - compare values and return true or false.

  Frequently used when coding conditional logic.
- Compound boolean - perform logical connective and negation operations and return true or false.

  Frequently used when coding conditional logic.

For examples and usage information, refer to Chapter s 6 and 8 of *Developing Web Applications with ColdFusion Express*.

The table below describes operators by type and symbol.

Type	Symbol	Description
Arithmetic	+, -, *, /	Add, subtract, multiply, and divide. In the case of division, the right operand cannot be zero. For example, `9/4` is `2.25`
	\	Divides two integer values and returns a whole number. For example, `9\4` is `2`.
	^	Returns the result of a number raised to a power (exponent). Use the ^ (caret) to separate the number from the power. The left operand cannot be zero. For example, `2^3` is `8`.
	MOD	Returns the remainder (modulus) after a number is divided. The result has the same sign as the divisor. The right operand cannot be zero. For example, `11 MOD 4` is `3`.
	+ or -	Set a number to either positive or negative. For example, `+2` is `2` and `-2 is (-1)*2`.
String	&	Concatenates text strings including those returned from variables and functions.

Type	Symbol	Description
Comparison	IS	Performs a case-insensitive comparison of two values. Returns true or false. For example, this code tests for equal values. The condition is true only if the FirstName value is Jeremy. `<CFIF Form.FirstName IS "Jeremy">`
	IS NOT	Opposite behavior of *IS*. For example, this code tests for inequal values. The condition is true only if the FirstName value is not Jeremy. `<CFIF Form.FirstName IS NOT "Jeremy">`
	CONTAINS	Checks to see if the value on the left is contained in the value on the right. Returns true or false. For example this code tests for a value condition. The condition is true only if the FirstName value contains Jeremy. `<CFIF Form.FirstName CONTAINS "Jeremy">`
	DOES NOT CONTAIN	Opposite behavior of *CONTAINS*.
	GREATER THAN	Checks to see if the value on the left is greater than the value on the right. Returns true or false.
	LESS THAN	Opposite behavior of *GREATER THAN*.
	GREATER THAN OR EQUAL TO	Checks to see if the value on the left is greater than or equal to the value on the right. Returns true or false.
	LESS THAN OR EQUAL TO	Checks to see if the value on the left is less than or equal to the value on the right. Returns true or false.

Type	Symbol	Description
Compound Boolean	NOT	Reverses the value of an argument. For example, NOT TRUE is FALSE and vice versa.
	AND	Returns true if both arguments are true; returns false otherwise. For example, TRUE AND TRUE is true, but TRUE AND FALSE is false.
	OR	Returns true if any argument is TRUE and returns false otherwise. For example, TRUE OR FALSE is true, but FALSE OR FALSE is false
	XOR	Returns TRUE if either argument is (exclusively) TRUE. For example, `TRUE XOR TRUE` is `FALSE`, but `TRUE XOR FALSE` is `TRUE`.
	EQV	Returns TRUE if both arguments are TRUE or both are FALSE (Equivalent). The EQV operator is the opposite of the XOR operator. For example, `TRUE EQV TRUE` is `TRUE`, but `TRUE EQV FALSE` is `FALSE`.

Shorthand notation for Boolean operators

You can replace some Boolean operators with shorthand notations to make your CFML more compact, as shown in the following table:

Shorthand Notation for Boolean Operators	
Operator	**Alternative name(s)**
IS	EQUAL, EQ
IS NOT	NOT EQUAL, NEQ
CONTAINS	Not available
DOES NOT CONTAIN	Not available
GREATER THAN	GT
LESS THAN	LT

Shorthand Notation for Boolean Operators (Continued)	
Operator	**Alternative name(s)**
GREATER THAN OR EQUAL TO	GTE, GE
LESS THAN OR EQUAL TO	LTE, LE

Operator precedence

The order of precedence controls which operator is evaluated first in an expression. Operators on the same line have the same precedence.

Operator precedence, highest to lowest

```
Unary +, Unary -
^
*, /
\
MOD
+, -
&
EQ, NEQ, LT, LTE, GT, GTE, CONTAINS, DOES NOT CONTAIN
NOT
AND
OR
XOR
EQV
IMP
```

To enforce a specific non-standard order of evaluation, you must parenthesize expressions. For example:

- 6 - 3 * 2 is equal to 0
- (6 - 3) * 2 is equal to 6

Parenthesized expressions can be arbitrarily nested. When in doubt about the order in which operators in an expression will be evaluated, always use parentheses.