CFML Reference
CONTENTS

INTRODUCTION .................................................. 5

CHAPTER 1: Reserved Words and Variables ......................... 7
    Reserved words ........................................... 7
    Scope-specific built-in variables .......................... 9
    Custom tag variables ..................................... 11
    ColdFusion tag-specific variables ........................ 11
    Standard CGI variables .................................. 16
    CGI environment variables ............................... 17

CHAPTER 2: ColdFusion Tags ................................... 21
    Tags by function ......................................... 26
    Tag changes since ColdFusion 5 .......................... 29

CHAPTER 3: ColdFusion Functions ............................... 449
    Functions by category .................................... 452
    Function changes since ColdFusion 5 .................... 457

CHAPTER 4: ColdFusion MX Flash Form Style Reference ............. 933
    Styles valid for all controls ............................ 934
    Styles for cfform ......................................... 936
    Styles for cfformgroup with horizontal or vertical type attributes .......................... 936
    Styles for box-style cfformgroup elements ................. 937
    Styles for cfformgroup with accordion type attribute .......................................... 938
    Styles for cfformgroup with tabnavigator type attribute ..................................... 939
    Styles for cfformitem with hrule or vrule type attributes .................................. 939
    Styles for cfformitem with type attributes ............... 940
    Styles for cfformitem with radioButton, checkbox, button, image, or submit type attributes .................................. 940
    Styles for cfinput with cfformitem attributes ............ 941
    Styles for cfformitem with password attributes ............ 941
    Styles for cfformitem with text, password, or hidden type attributes ......................... 941
    Styles for cfformitem with type attributes ............... 941
    Styles for cfformitem with size attribute value of 1 .............................................. 941
    Styles for cfformitem with size attribute value greater than 1 ................................ 942
    Styles for cfcalendar tag and cfinput with type attributes .................................. 942
    Styles for the cfgrid tag .................................. 943
    Styles for the cfgrid tag .................................. 943
CHAPTER 5: Application.CFC Reference .................................................. 945
  Application variables ................................................................. 945
  Method summary ................................................................. 946

CHAPTER 6: ColdFusion MX Event Gateway Reference .................. 963
  Gateway development interfaces and classes .......................... 963
  CFML CFEvent structure ......................................................... 1004
  IM gateway methods and commands ....................................... 1004
  SMS Gateway CFEvent structure and commands ....................... 1042

CHAPTER 7: ColdFusion C++ CFX Reference ............................... 1055
  C++ class overview ................................................................. 1055
  Deprecated class methods ...................................................... 1056
  CCFXException class ............................................................... 1056
  CCFXQuery class ................................................................. 1058
  CCFXRequest class ............................................................... 1062
  CCFXStringSet class ............................................................. 1072

CHAPTER 8: ColdFusion Java CFX Reference ............................... 1075
  Class libraries overview .......................................................... 1075
  Custom tag interface ............................................................. 1076
  Query interface ................................................................. 1077
  Request interface ............................................................... 1082
  Response interface .............................................................. 1087
  Debugging classes reference .................................................... 1090

CHAPTER 9: WDDX JavaScript Objects ........................................ 1091
  JavaScript object overview ................................................... 1091
  WddxSerializer object .......................................................... 1091
  WddxRecordset object ........................................................ 1092

CHAPTER 10: ColdFusion ActionScript Functions ......................... 1101
**INTRODUCTION**

*CFML Reference* is your primary ColdFusion Markup Language (CFML) reference. Use this manual to learn about CFML tags and functions, ColdFusion expressions, and using JavaScript objects for WDDX in Macromedia ColdFusion MX 7. It also provides detailed references for Java and C++ CFX interfaces.

**About Macromedia ColdFusion MX documentation**

The ColdFusion MX documentation is designed to provide support for the complete spectrum of participants.

**Documentation set**

The ColdFusion documentation set includes the following titles:

<table>
<thead>
<tr>
<th>Manual</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Installing and Using ColdFusion MX</td>
<td>Describes system installation and basic configuration for Windows, Solaris, Linux, and HP-UX.</td>
</tr>
<tr>
<td>Configuring and Administering ColdFusion MX</td>
<td>Part I describes how to manage the ColdFusion environment, including connecting to your data sources and configuring security for your applications. Part II describes Verity search tools and utilities that you can use for configuring the Verity K2 Server search engine, as well as creating, managing, and troubleshooting Verity collections.</td>
</tr>
<tr>
<td>ColdFusion MX Developer’s Guide</td>
<td>Describes how to develop your dynamic web applications, including retrieving and updating your data, and using structures and forms.</td>
</tr>
<tr>
<td>Getting Started Building ColdFusion MX Applications</td>
<td>Contains an overview of ColdFusion features and application development procedures. Includes a tutorial that guides you through the process of developing an example ColdFusion application.</td>
</tr>
<tr>
<td>CFML Reference</td>
<td>Provides descriptions, syntax, usage, and code examples for all ColdFusion tags, functions, and variables.</td>
</tr>
<tr>
<td>CFML Quick Reference</td>
<td>A brief guide that shows the syntax of ColdFusion tags, functions, and variables.</td>
</tr>
</tbody>
</table>
Viewing online documentation

All ColdFusion MX documentation is available online in HTML and Adobe Acrobat Portable Document Format (PDF) files. Go to the documentation home page for ColdFusion MX on the Macromedia website: www.macromedia.com.
CHAPTER 1
Reserved Words and Variables

This chapter provides information on Macromedia ColdFusion reserved words, and lists scope
variables.

Contents
Reserved words ................................................................. 7
Scope-specific built-in variables ........................................ 9
ColdFusion tag-specific variables ........................................ 11
CGI environment variables .................................................. 17

Reserved words
The following list indicates words you must not use for ColdFusion variables, user-defined
function names, or custom tag names. While some of these words can be used safely in some
situations, you can prevent errors by avoiding them entirely.

• Any name starting with cf. However, when you call a CFML custom tag directly, you prefix the
custom tag page name with cf_.
• Built-in function names, such as Now or Hash
• Scope names, such as Form or Session
• Operators, such as NE or IS
• The names of any built-in data structures, such as Error or File
• The names of any built-in variables, such as RecordCount or CGI variable names
• CFScript language element names such as for, default, or continue

Remember that ColdFusion is not case-sensitive. For example, all of the following are reserved
words: IS, Is, iS, and is.
Reserved words in forms

Reserved words in forms
You must also not create form field names ending in any of the following, except to specify a form field validation rule using a hidden form field name.

- \_integer
- \_float
- \_range
- \_date
- \_time
- \_eurodate

Reserved words in queries

Reserved words in queries
The following table lists SQL keywords that are reserved in ColdFusion queries of queries. This list includes all reserved words in the SQL standard, and should be avoided in variables used in all queries. Do not use these keywords as variable names in any queries.

Note: Many database management systems have additional reserved words that you cannot use as variable names in queries to their databases. For a detailed list, see your DBMS documentation.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Keyword</th>
<th>Keyword</th>
<th>Keyword</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABSOLUTE</td>
<td>ACTION</td>
<td>ADD</td>
<td>ALL</td>
</tr>
<tr>
<td>ALTER</td>
<td>AND</td>
<td>ANY</td>
<td>ARE</td>
</tr>
<tr>
<td>ASC</td>
<td>ASSERTION</td>
<td>AT</td>
<td>AUTHORIZATION</td>
</tr>
<tr>
<td>BEGIN</td>
<td>BETWEEN</td>
<td>BIT</td>
<td>BIT_LENGTH</td>
</tr>
<tr>
<td>BY</td>
<td>CASCADE</td>
<td>CASCADED</td>
<td>CASE</td>
</tr>
<tr>
<td>CATALOG</td>
<td>CHAR</td>
<td>CHARACTER</td>
<td>CHARACTER_LENGTH</td>
</tr>
<tr>
<td>CHECK</td>
<td>CLOSE</td>
<td>COALESCE</td>
<td>COLLATE</td>
</tr>
<tr>
<td>COLUMN</td>
<td>COMMIT</td>
<td>CONNECT</td>
<td>CONNNECTION</td>
</tr>
<tr>
<td>CONSTRAINTS</td>
<td>CONTINUE</td>
<td>CONVERT</td>
<td>CORRESPONDING</td>
</tr>
<tr>
<td>CREATE</td>
<td>CROSS</td>
<td>CURRENT</td>
<td>CURRENT_DATE</td>
</tr>
<tr>
<td>CURRENT_TIMESTAMP</td>
<td>CURRENT_USER</td>
<td>CURSOR</td>
<td>DATE</td>
</tr>
<tr>
<td>DEALLOCATE</td>
<td>DEC</td>
<td>DECIMAL</td>
<td>DECLARE</td>
</tr>
<tr>
<td>DEFERRABLE</td>
<td>DEFERRED</td>
<td>DELETE</td>
<td>DESC</td>
</tr>
<tr>
<td>DESCRIPTOR</td>
<td>DIAGNOSTICS</td>
<td>DISCONNECT</td>
<td>DISTINCT</td>
</tr>
<tr>
<td>DOUBLE</td>
<td>DROP</td>
<td>ELSE</td>
<td>END</td>
</tr>
<tr>
<td>ESCAPE</td>
<td>EXCEPT</td>
<td>EXCEPTION</td>
<td>EXEC</td>
</tr>
<tr>
<td>EXISTS</td>
<td>EXTERNAL</td>
<td>EXTRACT</td>
<td>FALSE</td>
</tr>
<tr>
<td>FIRST</td>
<td>FLOAT</td>
<td>FOR</td>
<td>FOREIGN</td>
</tr>
<tr>
<td>FROM</td>
<td>GET</td>
<td>GLOBAL</td>
<td>GO</td>
</tr>
<tr>
<td>------</td>
<td>-----</td>
<td>--------</td>
<td>----</td>
</tr>
<tr>
<td>GOTO</td>
<td>GROUP</td>
<td>HAVING</td>
<td>HOUR</td>
</tr>
<tr>
<td>IDENTITY</td>
<td>IN</td>
<td>INDICATOR</td>
<td>INITIALLY</td>
</tr>
<tr>
<td>INNER</td>
<td>INSENSITIVE</td>
<td>INSERT</td>
<td>INT</td>
</tr>
<tr>
<td>INTEGER</td>
<td>INTERVAL</td>
<td>INTO</td>
<td>IS</td>
</tr>
<tr>
<td>ISOLATION</td>
<td>JOIN</td>
<td>KEY</td>
<td>LANGUAGE</td>
</tr>
<tr>
<td>LEADING</td>
<td>LEVEL</td>
<td>LIKE</td>
<td>LOCAL</td>
</tr>
<tr>
<td>LOWER</td>
<td>MATCH</td>
<td>MAX</td>
<td>MIN</td>
</tr>
<tr>
<td>MODULE</td>
<td>MONTH</td>
<td>NAMES</td>
<td>NATIONAL</td>
</tr>
<tr>
<td>NCHAR</td>
<td>NEXT</td>
<td>NO</td>
<td>NOT</td>
</tr>
<tr>
<td>NULLIF</td>
<td>OCTET_LENGTH</td>
<td>OF</td>
<td>ON</td>
</tr>
<tr>
<td>ONLY</td>
<td>OPTION</td>
<td>OR</td>
<td>ORDER</td>
</tr>
<tr>
<td>OUTER</td>
<td>OVERLAPS</td>
<td>PAD</td>
<td>PARTIAL</td>
</tr>
<tr>
<td>POSITION</td>
<td>PRECISION</td>
<td>PREPARE</td>
<td>PRESERVE</td>
</tr>
<tr>
<td>PRIOR</td>
<td>PRIVILEGES</td>
<td>PROCEDURE</td>
<td>PUBLIC</td>
</tr>
<tr>
<td>REAL</td>
<td>REFERENCES</td>
<td>RELATIVE</td>
<td>RESTRICT</td>
</tr>
<tr>
<td>RIGHT</td>
<td>ROWS</td>
<td>SCHEMA</td>
<td>SCROLL</td>
</tr>
<tr>
<td>SECOND</td>
<td>SELECT</td>
<td>SESSION</td>
<td>SESSION_USER</td>
</tr>
<tr>
<td>SET</td>
<td>SMALLINT</td>
<td>SOME</td>
<td>SPACE</td>
</tr>
<tr>
<td>SQL</td>
<td>SQLCODE</td>
<td>SQLERROR</td>
<td>SQLSTATE</td>
</tr>
<tr>
<td>SUM</td>
<td>SYSTEM_USER</td>
<td>TABLE</td>
<td>TEMPORARY</td>
</tr>
<tr>
<td>TIME</td>
<td>TIMEZONE_HOUR</td>
<td>TIMEZONE_MINUTE</td>
<td>TO</td>
</tr>
<tr>
<td>TRAILING</td>
<td>TRANSLATE</td>
<td>TRANSLATION</td>
<td>TRIM</td>
</tr>
<tr>
<td>TRUE</td>
<td>UNIQUE</td>
<td>UNKNOWN</td>
<td>UPDATE</td>
</tr>
<tr>
<td>UPPER</td>
<td>USER</td>
<td>USING</td>
<td>VALUE</td>
</tr>
<tr>
<td>VALUES</td>
<td>VARCHAR</td>
<td>VARYING</td>
<td>VIEW</td>
</tr>
<tr>
<td>WHENCEVER</td>
<td>WHERE</td>
<td>WITH</td>
<td>WORK</td>
</tr>
<tr>
<td>YEAR</td>
<td>ZONE</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Scope-specific built-in variables**

ColdFusion returns variables, such as those returned in a `cfdirectory` or `cfftp` operation. A variable is usually referenced by scoping it according to its type: naming it according to the code context in which it is available; for example, `Session.varname`, or `Application.varname`. For more information on ColdFusion scopes, see Chapter 3, “Using ColdFusion Variables” in *ColdFusion MX Developer's Guide*.
You use the `cflock` tag to limit the scope of CFML constructs that modify shared data structures, files, and CFXs, to ensure that modifications occur sequentially. For more information, see `cflock` on page 270, and Chapter 15, “Using Persistent Data and Locking” in *ColdFusion MX Developer's Guide*.

**Variable scope**

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

**Caller scope**

**History**

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

**Client variables**

The following client variables are reserved:

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

**Server variables**

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

- `Server.ColdFusion.ProductName`
- `Server.ColdFusion.ProductVersion`
- `Server.ColdFusion.ProductLevel`
- `Server.ColdFusion.SerialNumber`
- `Server.ColdFusion.SupportedLocales`
- `Server.ColdFusion.AppServer`
- `Server.ColdFusion.Expiration`
- `Server.ColdFusion.RootDir`
- `Server.OS.Name`
- `Server.OS.AdditionalInformation`
- `Server.OS.Version`
- `Server.OS.BuildNumber`

**Application and session variables**

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

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

To ensure that modifications to shared data occur in the intended sequence, use the `cflock` tag. For more information, see `cflock` on page 270.
The predefined application and session variables are as follows:

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

**Custom tag variables**

A ColdFusion custom tag returns the following variables:

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

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

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

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

```cfoutput>$variable_name$</cfoutput>```

**Request variable**

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

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

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

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

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

**Form variable**

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

```Form.FieldNames```  

**ColdFusion tag-specific variables**

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

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

- `cfcatch`
- `cfdirectory`
- `cferror`
ColdFusion query variables

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

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

CFCATCH variables

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

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

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

- `CFCATCH.QueryError`
- `CFCATCH.SQL`
- `CFCATCH.Where`
- `CFCATCH.Datasource`

Within a `cfcatch` block, undefined variable exception properties can be accessed as the following variable:

- `CFCATCH.Name`

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

- `CFCATCH.TokenText`
- `CFCATCH.Snippet`
- `CFCATCH.Column`
- `CFCATCH.KnownColumn`
CFDIRECTORY variables

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

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

CFERROR variables

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

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

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

Error.ValidationHeader
Error.InvalidFields
Error.ValidationFooter

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

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

Note: You can substitute the prefix CFERROR for Error, if type = "Exception"; for example, CFERROR.Diagnostics, CFERROR.Mailto, or CFERROR.DateTime.
**CFFILE ACTION=Upload variables**

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

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

**CFFTP error variables**

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

CFFTP.Succeeded
CFFTP.ErrorCode
CFFTP.ErrorText

**CFFTP ReturnValue variable**

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

GetCurrentDir
GetCurrentURL
ExistsDir
ExistsFile
Exists

**CFFTP query object columns**

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

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

**CFHTTP variables**

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

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

**CFLDAP variables**

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

queryname.CurrentRow
queryname.RecordCount
queryname.ColumnList

**CFPOP variables**

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

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

**CFQUERY and CFSTOREDPROC variables**

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

CFQUERY.ExecutionTime

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

queryname.CurrentRow
queryname.RecordCount
queryname.ColumnList
The `cfstoredproc` tag returns the following variables:

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

**CFREGISTRY variables**

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

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

**CFSEARCH variables**

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

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

**Standard CGI variables**

The CGI variables that are available for your use vary with the web server and configuration. This section lists the CGI 1.1 variables that some web servers create when a CGI script is called. Some of the following variables may not be available.

**Request**

- `CGI.AUTH_TYPE`
- `CGI.CONTENT_LENGTH`
- `CGI.CONTENT_TYPE`
- `CGI.METHOD`
- `CGI.PATH_INFO`
- `CGI.PATH_TRANSLATED`
- `CGI.QUERY_STRING`
- `CGI.REMOTE_ADDR`
- `CGI.REMOTE_HOST`
- `CGI.REMOTE_USER`
- `CGI.REQUEST_METHOD`
- `CGI.SCRIPT_NAME`
Server
- CGI_GATEWAY_INTERFACE
- CGI_SERVER_NAME
- CGI_SERVER_PORT
- CGI_SERVER_PROTOCOL
- CGI_SERVER_SOFTWARE

Client
- CGI_CERT_ISSUER
- CGI_CERT_SUBJECT
- CGI_CLIENT_CERT_ENCODED
- CGI.HTTP_ACCEPT
- CGI.HTTP_IF_MODIFIED_SINCE
- CGI.HTTP_USER_AGENT

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

CGI environment variables

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

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

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

Testing for CGI variables

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

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

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

<table>
<thead>
<tr>
<th>CGI server variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SERVER_SOFTWARE</td>
<td>Name and version of the information server software answering the request (and running the gateway). Format: name/version.</td>
</tr>
<tr>
<td>SERVER_NAME</td>
<td>Server’s hostname, DNS alias, or IP address as it appears in self-referencing URLs.</td>
</tr>
<tr>
<td>GATEWAY_INTERFACE</td>
<td>CGI specification revision with which this server complies. Format: CGI/revision.</td>
</tr>
<tr>
<td>SERVER_PROTOCOL</td>
<td>Name and revision of the information protocol this request came in with. Format: protocol/revision.</td>
</tr>
<tr>
<td>SERVER_PORT</td>
<td>Port number to which the request was sent.</td>
</tr>
<tr>
<td>REQUEST_METHOD</td>
<td>Method with which the request was made. For HTTP, this is Get, Head, Post, and so on.</td>
</tr>
<tr>
<td>PATH_INFO</td>
<td>Extra path information, as given by the client. Scripts can be accessed by their virtual pathname, followed by extra information at the end of this path. The extra information is sent as PATH_INFO.</td>
</tr>
<tr>
<td>PATH_TRANSLATED</td>
<td>Translated version of PATH_INFO after any virtual-to-physical mapping.</td>
</tr>
<tr>
<td>SCRIPT_NAME</td>
<td>Virtual path to the script that is executing; used for self-referencing URLs.</td>
</tr>
<tr>
<td>QUERY_STRING</td>
<td>Query information that follows the ? in the URL that referenced this script.</td>
</tr>
<tr>
<td>REMOTE_HOST</td>
<td>Hostname making the request. If the server does not have this information, it sets REMOTE_ADDR and does not set REMOTE_HOST.</td>
</tr>
<tr>
<td>REMOTE_ADDR</td>
<td>IP address of the remote host making the request.</td>
</tr>
<tr>
<td>AUTH_TYPE</td>
<td>If the server supports user authentication, and the script is protected, the protocol-specific authentication method used to validate the user.</td>
</tr>
<tr>
<td>REMOTE_USER</td>
<td>If the server supports user authentication, and the script is protected, the username the user has authenticated as. (Also available as AUTH_USER.)</td>
</tr>
<tr>
<td>AUTH_USER</td>
<td></td>
</tr>
<tr>
<td>REMOTE_IDENT</td>
<td>If the HTTP server supports RFC 931 identification, this variable is set to the remote username retrieved from the server. Use this variable for logging only.</td>
</tr>
<tr>
<td>CONTENT_TYPE</td>
<td>For queries that have attached information, such as HTTP POST and PUT, this is the content type of the data.</td>
</tr>
<tr>
<td>CONTENT_LENGTH</td>
<td>Length of the content as given by the client.</td>
</tr>
</tbody>
</table>
CGI client variables

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

<table>
<thead>
<tr>
<th>CGI client variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTTP_REFERER</td>
<td>The referring document that linked to or submitted form data.</td>
</tr>
<tr>
<td>HTTP_USER_AGENT</td>
<td>The browser that the client is currently using to send the request.</td>
</tr>
<tr>
<td>HTTP_IF_MODIFIED_SINCE</td>
<td>The last time the page was modified. The browser determines whether to set</td>
</tr>
<tr>
<td></td>
<td>this variable, usually in response to the server having sent the LAST_</td>
</tr>
<tr>
<td></td>
<td>MODIFIED HTTP header. It can be used to take advantage of browser-side</td>
</tr>
<tr>
<td></td>
<td>caching.</td>
</tr>
</tbody>
</table>

CGI client certificate variables

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

<table>
<thead>
<tr>
<th>CGI client certificate variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CERT_SUBJECT</td>
<td>Client-specific information provided by the web server. This data typically</td>
</tr>
<tr>
<td></td>
<td>includes the client’s name, e-mail address, and so on; for example:</td>
</tr>
<tr>
<td></td>
<td>O = &quot;VeriSign, Inc.*&quot;, OU = VeriSign Trust Network, OU =</td>
</tr>
<tr>
<td></td>
<td>&quot;www.verisign.com/repository/RPA Incorp. by Ref..LIAB.LTD(c)98*&quot;, OU =</td>
</tr>
<tr>
<td></td>
<td>&quot;Persona Not Validated&quot;, OU = Digital ID Class 1 = Microsoft, CN = Matthew</td>
</tr>
<tr>
<td></td>
<td>Lund, E = <a href="mailto:mlund@macromedia.com">mlund@macromedia.com</a></td>
</tr>
<tr>
<td>CERT_ISSUER</td>
<td>Information about the authority that provided the client certificate; for</td>
</tr>
<tr>
<td></td>
<td>example:</td>
</tr>
<tr>
<td></td>
<td>O = &quot;VeriSign, Inc.*&quot;, OU = VeriSign Trust Network, OU =</td>
</tr>
<tr>
<td></td>
<td>&quot;www.verisign.com/repository/RPA Incorp. By Ref..LIAB.LTD(c)98*&quot;, CN =</td>
</tr>
<tr>
<td></td>
<td>VeriSign Class 1 CA Individual Subscriber-Persona Not Validated</td>
</tr>
<tr>
<td>CLIENT_CERT_ENCODED</td>
<td>The entire client certificate binary, base-64 encoded. This data is typically</td>
</tr>
<tr>
<td></td>
<td>of interest to developers, so they can integrate with other software that</td>
</tr>
<tr>
<td></td>
<td>uses client certificates.</td>
</tr>
</tbody>
</table>
ColdFusion Markup Language (CFML) includes a set of tags that you use in Macromedia ColdFusion MX 7 pages to interact with data sources, manipulate data, and display output. CFML tag syntax is similar to HTML element syntax.

This chapter contains categorized and alphabetical lists of the tags followed by the detailed tag descriptions.

Contents

Tag summary ................................................................. 21
Tags by function .......................................................... 26
Tag changes since ColdFusion 5 ...................................... 29
Tag descriptions ............................................................ 36

Tag summary

The following table briefly describes CFML tags:

<table>
<thead>
<tr>
<th>CFML tag</th>
<th>Category</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cfabort</td>
<td>Flow-control tags</td>
<td>Stops the processing of a ColdFusion page at the tag location</td>
</tr>
<tr>
<td>cfapplet</td>
<td>Forms tags</td>
<td>Embeds Java applets in a cfform tag</td>
</tr>
<tr>
<td>cfapplication</td>
<td>Application framework tags</td>
<td>Defines an application name; activates client variables; specifies client variable storage mechanism</td>
</tr>
<tr>
<td>cfargument</td>
<td>Extensibility tags</td>
<td>Creates a parameter definition within a component definition; defines a function argument</td>
</tr>
<tr>
<td>cfassociate</td>
<td>Application framework tags</td>
<td>Enables subtag data to be saved with a base tag</td>
</tr>
<tr>
<td>cfbreak</td>
<td>Flow-control tags</td>
<td>Breaks out of a CFML looping construct</td>
</tr>
<tr>
<td>cfcache</td>
<td>Page processing tags</td>
<td>Caches ColdFusion pages</td>
</tr>
<tr>
<td>CFML tag</td>
<td>Category</td>
<td>Description</td>
</tr>
<tr>
<td>-------------</td>
<td>---------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>cfcalendar</td>
<td>Forms tags</td>
<td>Provides a calendar from which to select a date</td>
</tr>
<tr>
<td>cfcase</td>
<td>Flow-control tags</td>
<td>Used with the cfswitch and cfdefaultcase tags</td>
</tr>
<tr>
<td>cfcatch</td>
<td>Exception handling tags, Flow-control tags</td>
<td>Catches exceptions in ColdFusion pages</td>
</tr>
<tr>
<td>cfchart</td>
<td>Data output tags</td>
<td>Generates and displays a chart</td>
</tr>
<tr>
<td>cfchartdata</td>
<td>Data output tags</td>
<td>Defines chart data points</td>
</tr>
<tr>
<td>cfchartseries</td>
<td>Data output tags</td>
<td>Defines style in which chart data displays</td>
</tr>
<tr>
<td>cfcol</td>
<td>Data output tags</td>
<td>Defines table column header, properties</td>
</tr>
<tr>
<td>cfcollection</td>
<td>Extensibility tags</td>
<td>Administers Verity collections</td>
</tr>
<tr>
<td>cfcomponent</td>
<td>Extensibility tags</td>
<td>Creates and defines a component object</td>
</tr>
<tr>
<td>cfcontent</td>
<td>Data output tags, Page processing tags</td>
<td>Defines content type and filename of a file to be downloaded by current page</td>
</tr>
<tr>
<td>cfcookie</td>
<td>Variable manipulation tags</td>
<td>Defines and sets cookie variables, including expiration and security options</td>
</tr>
<tr>
<td>cfdefaultcase</td>
<td>Flow-control tags</td>
<td>Receives control if there is no matching cfcase tag value</td>
</tr>
<tr>
<td>cfdirectory</td>
<td>File management tags</td>
<td>Performs typical directory-handling tasks from within a ColdFusion application</td>
</tr>
<tr>
<td>cfdocument</td>
<td>Data output tags</td>
<td>Creates PDF or FlashPaper output from a text block containing CFML and HTML</td>
</tr>
<tr>
<td>cfdocumentitem</td>
<td>Data output tags</td>
<td>Specifies action items, such as header, footer, and page break, for a PDF or FlashPaper document</td>
</tr>
<tr>
<td>cfdocumentsection</td>
<td>Data output tags</td>
<td>Divides a PDF or FlashPaper document into sections</td>
</tr>
<tr>
<td>cfdump</td>
<td>Debugging tags, Variable manipulation tags</td>
<td>Outputs variables for debugging</td>
</tr>
<tr>
<td>cfelse</td>
<td>Flow-control tags</td>
<td>Creates IF-THEN-ELSE constructs</td>
</tr>
<tr>
<td>cfelseif</td>
<td>Flow-control tags</td>
<td>Creates IF-THEN-ELSE constructs</td>
</tr>
<tr>
<td>cferror</td>
<td>Exception handling tags, Application framework tags</td>
<td>Displays custom HTML error pages when errors occur</td>
</tr>
<tr>
<td>cfexecute</td>
<td>Flow-control tags, Extensibility tags</td>
<td>Executes developer-specified process on server computer</td>
</tr>
<tr>
<td>cfexit</td>
<td>Flow-control tags</td>
<td>Aborts processing of executing CFML tag</td>
</tr>
<tr>
<td>cffile</td>
<td>File management tags</td>
<td>Performs typical file-handling tasks from within ColdFusion application</td>
</tr>
<tr>
<td>CFML tag</td>
<td>Category</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>----------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td><code>cfflush</code></td>
<td>Data output tags, Page processing tags</td>
<td>Flushes currently available data to client</td>
</tr>
<tr>
<td><code>cfform</code></td>
<td>Forms tags</td>
<td>Builds input form; performs client-side input validation</td>
</tr>
<tr>
<td><code>cfformgroup</code></td>
<td>Forms tags</td>
<td>Groups form control into a containing object</td>
</tr>
<tr>
<td><code>cfformitem</code></td>
<td>Forms tags</td>
<td>Adds text and dividing rules to Flash forms</td>
</tr>
<tr>
<td><code>cfftp</code></td>
<td>Forms tags, Extensibility tags, Internet Protocol tags</td>
<td>Permits FTP file operations</td>
</tr>
<tr>
<td><code>cffunction</code></td>
<td>Extensibility tags</td>
<td>Defines function that you build in CFML</td>
</tr>
<tr>
<td><code>cfgrid</code></td>
<td>Forms tags</td>
<td>Displays tabular grid control; used in <code>cform</code> tag</td>
</tr>
<tr>
<td><code>cfgridcolumn</code></td>
<td>Forms tags</td>
<td>Used in <code>cform</code>; defines columns in a <code>cfgrid</code></td>
</tr>
<tr>
<td><code>cfgridrow</code></td>
<td>Forms tags</td>
<td>Defines a grid row; used with <code>cfgrid</code></td>
</tr>
<tr>
<td><code>cfgridupdate</code></td>
<td>Forms tags</td>
<td>Directly updates ODBC data source from edited grid data</td>
</tr>
<tr>
<td><code>cfheader</code></td>
<td>Data output tags, Page processing tags</td>
<td>Generates HTTP headers</td>
</tr>
<tr>
<td><code>cfhtmlhead</code></td>
<td>Page processing tags</td>
<td>Performs GET and POST to upload file or post form, cookie, query, or CGI variable directly to server</td>
</tr>
<tr>
<td><code>cfhttp</code></td>
<td>Internet Protocol tags</td>
<td>Performs GET and POST to upload file or post form, cookie, query, or CGI variable directly to server</td>
</tr>
<tr>
<td><code>cfhtpparam</code></td>
<td>Internet Protocol tags</td>
<td>Specifies parameters required for a <code>cfhttp</code> POST operation; used with <code>cfhttp</code></td>
</tr>
<tr>
<td><code>cfif</code></td>
<td>Flow-control tags</td>
<td>Creates IF-THEN-ELSE constructs</td>
</tr>
<tr>
<td><code>cfimport</code></td>
<td>Application framework tags</td>
<td>Imports JSP tag libraries into a CFML page</td>
</tr>
<tr>
<td><code>cfinclude</code></td>
<td>Flow-control tags</td>
<td>Embeds references to ColdFusion pages</td>
</tr>
<tr>
<td><code>cfindex</code></td>
<td>Extensibility tags</td>
<td>Creates Verity search indexes</td>
</tr>
<tr>
<td><code>cfinput</code></td>
<td>Forms tags</td>
<td>Creates an input element (radio button, check box, text entry box); used in <code>cform</code></td>
</tr>
<tr>
<td><code>cfinsert</code></td>
<td>Database manipulation tags</td>
<td>Inserts records in a data source</td>
</tr>
<tr>
<td><code>cfinvoke</code></td>
<td>Extensibility tags</td>
<td>Invokes component methods from a ColdFusion page or component</td>
</tr>
<tr>
<td><code>cfinvokeargument</code></td>
<td>Extensibility tags</td>
<td>Passes a parameter to a component method or a web service</td>
</tr>
<tr>
<td><code>cfldap</code></td>
<td>Internet Protocol tags</td>
<td>Provides access to LDAP directory servers</td>
</tr>
<tr>
<td><code>cflocation</code></td>
<td>Flow-control tags</td>
<td>Controls execution of a page</td>
</tr>
<tr>
<td>CFML tag</td>
<td>Category</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>---------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>cflock</td>
<td>Application framework tags</td>
<td>Ensures data integrity and synchronizes execution of CFML code</td>
</tr>
<tr>
<td>cflog</td>
<td>Data output tags, Other tags</td>
<td>Writes a message to a log file</td>
</tr>
<tr>
<td>cflogin</td>
<td>Security tags</td>
<td>Defines a container for user login and authentication code</td>
</tr>
<tr>
<td>cfloginuser</td>
<td>Security tags</td>
<td>Identifies an authenticated user to ColdFusion</td>
</tr>
<tr>
<td>cflogout</td>
<td>Security tags</td>
<td>Logs the current user out</td>
</tr>
<tr>
<td>cfloop</td>
<td>Flow-control tags</td>
<td>Repeats a set of instructions based on conditions</td>
</tr>
<tr>
<td>cfmail</td>
<td>Internet Protocol tags</td>
<td>Assembles and posts an e-mail message</td>
</tr>
<tr>
<td>cfmailparam</td>
<td>Internet Protocol tags</td>
<td>Attaches a file or adds a header to an e-mail message</td>
</tr>
<tr>
<td>cfmailpart</td>
<td>Internet Protocol tags</td>
<td>Contains one part of a multi-part mail message</td>
</tr>
<tr>
<td>cfmodule</td>
<td>Application framework tags</td>
<td>Invokes a custom tag for use in a ColdFusion page</td>
</tr>
<tr>
<td>cfNTauthenticate</td>
<td>Security tags</td>
<td>Authenticates user information against an NT domain</td>
</tr>
<tr>
<td>cfobject</td>
<td>Extensibility tags</td>
<td>Creates COM, component, CORBA, Java, and web service objects</td>
</tr>
<tr>
<td>cfobjectcache</td>
<td>Database manipulation tags</td>
<td>Flushes the query cache</td>
</tr>
<tr>
<td>cfoutput</td>
<td>Data output tags</td>
<td>Displays the output of a database query or other operation</td>
</tr>
<tr>
<td>cfparam</td>
<td>Variable manipulation tags</td>
<td>Defines a parameter and its default value</td>
</tr>
<tr>
<td>cfpop</td>
<td>Internet Protocol tags</td>
<td>Gets and deletes messages from POP mail server</td>
</tr>
<tr>
<td>cfprocessingdirective</td>
<td>Data output tags</td>
<td>Suppresses white space and other output</td>
</tr>
<tr>
<td>cfprocparam</td>
<td>Database manipulation tags</td>
<td>Holds parameter information for stored procedure</td>
</tr>
<tr>
<td>cfprocresult</td>
<td>Database manipulation tags</td>
<td>Result set name that ColdFusion tags use to access result set of a stored procedure</td>
</tr>
<tr>
<td>cfproperty</td>
<td>Extensibility tags</td>
<td>Defines components</td>
</tr>
<tr>
<td>cfquery</td>
<td>Database manipulation tags</td>
<td>Passes SQL statements to a database</td>
</tr>
<tr>
<td>cfqueryparam</td>
<td>Database manipulation tags</td>
<td>Checks data type of a query parameter</td>
</tr>
<tr>
<td>CFML tag</td>
<td>Category</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>--------------------------------------------------</td>
<td>-------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>cfschedule</td>
<td>Variable manipulation tags</td>
<td>Schedules page execution; optionally, produces static pages</td>
</tr>
<tr>
<td>cfscript</td>
<td>Application framework tags</td>
<td>Encloses a set of cfscript statements</td>
</tr>
<tr>
<td>cfsearch</td>
<td>Extensibility tags</td>
<td>Executes searches against data indexed in Verity collections, using cfindex</td>
</tr>
<tr>
<td>cfselect</td>
<td>Forms tags</td>
<td>Creates a drop-down list box form element; used in cfform tag</td>
</tr>
<tr>
<td>cfset</td>
<td>Variable manipulation tags</td>
<td>Defines a variable</td>
</tr>
<tr>
<td>cfsetting</td>
<td>Variable manipulation tags</td>
<td>Defines and controls ColdFusion settings</td>
</tr>
<tr>
<td>cfscript</td>
<td>Page processing tags</td>
<td></td>
</tr>
<tr>
<td>cfsilent</td>
<td>Data output tags, Page output tags</td>
<td>Suppresses CFML output within tag scope</td>
</tr>
<tr>
<td>cfslider</td>
<td>Forms tags</td>
<td>Creates slider control; used in cfform</td>
</tr>
<tr>
<td>cfthrow</td>
<td>Exception handling tags</td>
<td>Throws a developer-specified exception</td>
</tr>
<tr>
<td>cftrace</td>
<td>Debugging tags</td>
<td>Displays and logs application debugging data</td>
</tr>
<tr>
<td>cftransaction</td>
<td>Database manipulation tags</td>
<td>Groups cfquery operations into one transaction; performs rollback processing</td>
</tr>
<tr>
<td>cfrealm</td>
<td>Other tags, Variable manipulation tags</td>
<td>Reads, writes, and deletes keys and values in a Windows system registry</td>
</tr>
<tr>
<td>cfreport</td>
<td>Exception handling tags</td>
<td>Embeds a ColdFusion Report Builder or Crystal Reports report</td>
</tr>
<tr>
<td>cfreportparam</td>
<td>Exception handling tags</td>
<td>Passes an input parameter to a ColdFusion Report Builder report</td>
</tr>
<tr>
<td>cfreturn</td>
<td>Extensibility tags</td>
<td>Returns results from a component method</td>
</tr>
<tr>
<td>cfsavecontent</td>
<td>Variable manipulation tags</td>
<td>Saves generated content inside tag body in a variable</td>
</tr>
<tr>
<td>cfsetting</td>
<td>Variable manipulation tags</td>
<td></td>
</tr>
<tr>
<td>cfcsilent</td>
<td>Data output tags, Page output tags</td>
<td></td>
</tr>
<tr>
<td>cfslider</td>
<td>Forms tags</td>
<td></td>
</tr>
<tr>
<td>cfthrow</td>
<td>Exception handling tags</td>
<td></td>
</tr>
<tr>
<td>cftrace</td>
<td>Debugging tags</td>
<td></td>
</tr>
<tr>
<td>cftransaction</td>
<td>Database manipulation tags</td>
<td></td>
</tr>
</tbody>
</table>
## Tags by function

The following tables list Tags by their function or purpose.

- **Application framework tags** ............................................. 26
- **Database manipulation tags** ............................................. 27
- **Data output tags** .......................................................... 27
- **Debugging tags** .............................................................. 27
- **Exception handling tags** .................................................. 27
- **Extensibility tags** ........................................................... 27
- **File management tags** ....................................................... 27
- **Flow-control tags** ............................................................ 28
- **Forms tags** ................................................................. 28
- **Internet Protocol tags** ...................................................... 28
- **Page processing tags** ....................................................... 28
- **Security tags** ................................................................. 28
- **Variable manipulation tags** .............................................. 28
- **Other tags** ................................................................. 29

### Application framework tags

<table>
<thead>
<tr>
<th>CFML tag</th>
<th>Category</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cftree</td>
<td>Forms tags</td>
<td>Creates tree control element; used in cfform</td>
</tr>
<tr>
<td>cftreeitem</td>
<td>Forms tags</td>
<td>Populates a tree control element in a form; used with cftree</td>
</tr>
<tr>
<td>cftry</td>
<td>Exception handling tags, Flow-control tags</td>
<td>Catches exceptions in ColdFusion pages</td>
</tr>
<tr>
<td>cfupdate</td>
<td>Database manipulation tags</td>
<td>Updates rows in a database data source</td>
</tr>
<tr>
<td>cfwddx</td>
<td>Extensibility tags</td>
<td>Serializes and de-serializes CFML data structures to XML-based WDDX format</td>
</tr>
<tr>
<td>cfxml</td>
<td>Extensibility tags</td>
<td>Creates an XML document object</td>
</tr>
</tbody>
</table>
Database manipulation tags

cfinsert    cfprocresult    cfstoredproc
cfobjectcache cfquery    cftransaction
cfproccparam cfqueryparam cfupdate

Data output tags

cfchart    cfdocumentitem    cfoutput
cfchartdata cfdocumentsection cfprocesingdirective
cfchartseries cfflush    cfreport
cfcollection cfheader    cfreportparam
cfcontent cflog    cfsilent
cfdocument cfoutput    cftable

Debugging tags

cfdump    cftime    cftime

cfcatch cfthrow    cftimer cftrace

cferror cfthrow

Exception handling tags

Extensibility tags

cfchart    cffunction    cfreportparam
cfchartdata cfindex    cfreturn
cfchartseries cfinvoke    cssearch
cfcollection cfinvokeargument cfwddx
cfcomponent cfobject    cfxml
cfexecute cfproperty
cfftp cfftp

File management tags

cfdirectory cffile cfftp
Flow-control tags

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

Forms tags

cfapplet  cfgrid  cfselect
 cfcalendar  cfgridcolumn  cfslider
 cfform  cfgridrow  cftextarea
 cfformgroup  cfgridupdate  cftree
 cfformitem  cfinput  cftreeitem

Internet Protocol tags

cfttp  cfldap  cfmailpart
 cfhttp  cfmail  cfpop
 cfhttpparam  cfmailparam

Page processing tags

cfcache  cfheader  cfprocessingdirective
 cfcontent  cfhtmlhead  cfsetting
 cfflush  cfinclude  cfsilent

Security tags

cflogin  cflogout
 cfloginuser  cfNTauthenticate

Variable manipulation tags

cfcookie  cfregistry  cfset
 cfdump  cfsavecontent  cfsetting
 cfparam  cfschedule
Other tags

`cflog`  `cfregistry`

Tag changes since ColdFusion 5

The following tables list Tags, attributes, and values that have changed since ColdFusion 5.0 and indicate the specific release in which the change was made.

New tags, attributes, and values ........................................... 29
Deprecated tags, attributes, and values ................................ 34
Obsolete tags, attributes, and values .................................. 35

New tags, attributes, and values

This table lists tags, attributes, and attribute values that have been added since the ColdFusion MX release:

<table>
<thead>
<tr>
<th>Tag</th>
<th>Attribute or value</th>
<th>Added in this ColdFusion release</th>
</tr>
</thead>
<tbody>
<tr>
<td>cfapplication</td>
<td>scriptProtect</td>
<td>ColdFusion MX 7</td>
</tr>
<tr>
<td></td>
<td>loginStorage</td>
<td>ColdFusion MX 6.1</td>
</tr>
<tr>
<td>cfargument</td>
<td>xml value of type attribute</td>
<td>ColdFusion MX 7</td>
</tr>
<tr>
<td></td>
<td>All</td>
<td>ColdFusion MX</td>
</tr>
<tr>
<td>cfcache</td>
<td>cachedirectory, timespan attributes</td>
<td>ColdFusion MX</td>
</tr>
<tr>
<td>cfcalendar</td>
<td>All</td>
<td>ColdFusion MX 7</td>
</tr>
<tr>
<td>cfchart</td>
<td>style, title attributes</td>
<td>ColdFusion MX 7</td>
</tr>
<tr>
<td></td>
<td>xAxisType, yAxisType attributes</td>
<td>ColdFusion MX 6.1</td>
</tr>
<tr>
<td></td>
<td>All</td>
<td>ColdFusion MX</td>
</tr>
<tr>
<td>cfchartdata</td>
<td>All</td>
<td>ColdFusion MX</td>
</tr>
<tr>
<td>cfchartseries</td>
<td>datalabelstyle attribute</td>
<td>ColdFusion MX 7</td>
</tr>
<tr>
<td></td>
<td>column value of type attribute</td>
<td>ColdFusion MX</td>
</tr>
<tr>
<td></td>
<td>All</td>
<td>ColdFusion MX</td>
</tr>
<tr>
<td>cfcollection</td>
<td>categories attribute</td>
<td>ColdFusion MX 7</td>
</tr>
<tr>
<td></td>
<td>New values of the language attribute</td>
<td></td>
</tr>
<tr>
<td></td>
<td>list and categoryList values of action attribute</td>
<td></td>
</tr>
<tr>
<td></td>
<td>name attribute</td>
<td>ColdFusion MX</td>
</tr>
<tr>
<td>Tag</td>
<td>Attribute or value</td>
<td>Added in this ColdFusion release</td>
</tr>
<tr>
<td>--------------</td>
<td>------------------------------------------------------------------------------------</td>
<td>----------------------------------</td>
</tr>
<tr>
<td>cfcomponent</td>
<td>style, namespace, serviceportname, porttypename, wsdlfile, and bindingname attributes</td>
<td>ColdFusion MX 7</td>
</tr>
<tr>
<td></td>
<td>Extended functionality for the hint and displayname attributes when publishing document-literal style web services</td>
<td></td>
</tr>
<tr>
<td>cfcontent</td>
<td>variable attribute</td>
<td>ColdFusion MX 7</td>
</tr>
<tr>
<td>cfdirectory</td>
<td>recurse attribute for list and delete actions</td>
<td>ColdFusion MX 7</td>
</tr>
<tr>
<td>cfdocument</td>
<td>All</td>
<td>ColdFusion MX 7</td>
</tr>
<tr>
<td>cfdocumentitem</td>
<td>All</td>
<td>ColdFusion MX 7</td>
</tr>
<tr>
<td>cfdocumentsection</td>
<td>All</td>
<td>ColdFusion MX 7</td>
</tr>
<tr>
<td>cfexecute</td>
<td>variable attribute</td>
<td>ColdFusion MX 6.1</td>
</tr>
<tr>
<td>cffile</td>
<td>result attribute for upload action</td>
<td>ColdFusion MX 7</td>
</tr>
<tr>
<td></td>
<td>fixnewline attribute for action=&quot;append&quot; and action=&quot;write&quot;</td>
<td></td>
</tr>
<tr>
<td>cfform</td>
<td>name and action attributes are optional</td>
<td>ColdFusion MX 7</td>
</tr>
<tr>
<td></td>
<td>accessible, format, height, width, method, onError, preloader, scriptsrc, skin, style, tooltip, wMode attributes</td>
<td></td>
</tr>
<tr>
<td></td>
<td>onReset event</td>
<td></td>
</tr>
<tr>
<td>cfformgroup</td>
<td>All</td>
<td>ColdFusion MX 7</td>
</tr>
<tr>
<td>cfformitem</td>
<td>All</td>
<td>ColdFusion MX 7</td>
</tr>
<tr>
<td>cftp</td>
<td>result attribute</td>
<td>ColdFusion MX 7</td>
</tr>
<tr>
<td>cffunction</td>
<td>description attributes</td>
<td>ColdFusion MX 7</td>
</tr>
<tr>
<td></td>
<td>All</td>
<td>ColdFusion MX</td>
</tr>
<tr>
<td>cfgrid</td>
<td>format attribute and support for Flash and XML output enabled, onChange, style, tooltip, visible attributes (Flash format only)</td>
<td>ColdFusion MX 7</td>
</tr>
<tr>
<td>cfgridcolumn</td>
<td>mask attribute</td>
<td>ColdFusion MX 7</td>
</tr>
<tr>
<td></td>
<td>currency value of type attribute</td>
<td>ColdFusion MX 7</td>
</tr>
</tbody>
</table>

30 Chapter 2: ColdFusion Tags
<table>
<thead>
<tr>
<th>Tag</th>
<th>Attribute or value</th>
<th>Added in this ColdFusion release</th>
</tr>
</thead>
<tbody>
<tr>
<td>cfhttp</td>
<td>result attribute</td>
<td>ColdFusion MX 7</td>
</tr>
<tr>
<td></td>
<td>HEAD, PUT, DELETE, OPTIONS, and TRACE values of method attribute</td>
<td>ColdFusion MX 6.1</td>
</tr>
<tr>
<td></td>
<td>multipart, getasbinary, proxyUser, proxyPassword attributes</td>
<td></td>
</tr>
<tr>
<td></td>
<td>charset, firstrowasheaders attributes</td>
<td>ColdFusion MX</td>
</tr>
<tr>
<td>cfhttpparam</td>
<td>header and body values of type attribute</td>
<td>ColdFusion MX 6.1</td>
</tr>
<tr>
<td></td>
<td>encoded, mimeType attributes</td>
<td></td>
</tr>
<tr>
<td>cfimport</td>
<td>All</td>
<td>ColdFusion MX</td>
</tr>
<tr>
<td>cfindex</td>
<td>custom3, custom4, category, and categorytree attributes for update and refresh actions</td>
<td>ColdFusion MX 7</td>
</tr>
<tr>
<td></td>
<td>status attribute for Update, Refresh, Delete, Purge actions</td>
<td></td>
</tr>
<tr>
<td></td>
<td>New values of the language attribute</td>
<td></td>
</tr>
<tr>
<td>cfinput</td>
<td>height and width attributes (all except checkbox and radiobutton)</td>
<td>ColdFusion MX 7</td>
</tr>
<tr>
<td></td>
<td>bind attribute (text and password)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>label attribute (all but radiobutton, image, reset, and submit)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>mask attribute (text only)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>validateAt attribute (all but radiobutton, image, reset, and submit)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>datefield, button, file, hidden, image, reset, and submit values of type attribute</td>
<td></td>
</tr>
<tr>
<td></td>
<td>daynames and monthnames attributes (type=&quot;datefield&quot; only)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>boolean, email, guid, maxlength, noblanks, range, submitOnce, URL, USdate, and uuid values of the validate attribute</td>
<td></td>
</tr>
<tr>
<td></td>
<td>tooltip, visible, and enabled attributes (Flash forms only)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>src attribute (image only)</td>
<td></td>
</tr>
</tbody>
</table>

Tag changes since ColdFusion 5
<table>
<thead>
<tr>
<th>Tag</th>
<th>Attribute or value</th>
<th>Added in this ColdFusion release</th>
</tr>
</thead>
<tbody>
<tr>
<td>cfinvoke</td>
<td>servicePort attribute for web services</td>
<td>ColdFusion MX 7</td>
</tr>
<tr>
<td></td>
<td>All</td>
<td>ColdFusion MX</td>
</tr>
<tr>
<td>cfinvokeargument</td>
<td>omit attribute</td>
<td>ColdFusion MX 7</td>
</tr>
<tr>
<td></td>
<td>All</td>
<td>ColdFusion MX</td>
</tr>
<tr>
<td>cfldap</td>
<td>returnAsBinary attribute</td>
<td>ColdFusion MX 7</td>
</tr>
<tr>
<td>cflogin</td>
<td>All</td>
<td>ColdFusion MX</td>
</tr>
<tr>
<td>cfloginuser</td>
<td>All</td>
<td>ColdFusion MX</td>
</tr>
<tr>
<td>cflogout</td>
<td>All</td>
<td>ColdFusion MX</td>
</tr>
<tr>
<td>cfmail</td>
<td>spoolEnable attribute</td>
<td>ColdFusion MX</td>
</tr>
<tr>
<td></td>
<td>charset, failto, replyTo, userName, password, wrapText attributes</td>
<td>ColdFusion MX 6.1</td>
</tr>
<tr>
<td>cfmailparam</td>
<td>contentId, disposition attributes</td>
<td>ColdFusion MX 7</td>
</tr>
<tr>
<td></td>
<td>type attribute</td>
<td>ColdFusion MX 6.1</td>
</tr>
<tr>
<td>cfmailpart</td>
<td>All</td>
<td>ColdFusion MX 6.1</td>
</tr>
<tr>
<td>cfNauthenticate</td>
<td>All</td>
<td>ColdFusion MX 7</td>
</tr>
<tr>
<td>cfobject</td>
<td>All</td>
<td>ColdFusion MX</td>
</tr>
<tr>
<td>cfobjectcache</td>
<td>All</td>
<td>ColdFusion MX</td>
</tr>
<tr>
<td>cfparam</td>
<td>min, max, pattern attributes</td>
<td>ColdFusion MX 7</td>
</tr>
<tr>
<td></td>
<td>creditcard, email, eurodate, float, integer, range, regex, regular_expression, ssn, social_security_number, time, URL, USdate, XML, zipcode attributes of the type attribute</td>
<td></td>
</tr>
<tr>
<td>cfprocessingdirective</td>
<td>pageEncoding attribute</td>
<td>ColdFusion MX</td>
</tr>
<tr>
<td>cfproperty</td>
<td>All</td>
<td>ColdFusion MX</td>
</tr>
<tr>
<td>cfquery</td>
<td>result attribute</td>
<td>ColdFusion MX 7</td>
</tr>
<tr>
<td>cfreturn</td>
<td>All</td>
<td>ColdFusion MX</td>
</tr>
<tr>
<td>cfreport</td>
<td>template, format, name, filename, query, overwrite attributes</td>
<td>ColdFusion MX 7</td>
</tr>
<tr>
<td>cfreportparam</td>
<td>All</td>
<td>ColdFusion MX 7</td>
</tr>
<tr>
<td>Tag</td>
<td>Attribute or value</td>
<td>Added in this ColdFusion release</td>
</tr>
<tr>
<td>-----------</td>
<td>-----------------------------------------------------------------------------------</td>
<td>----------------------------------</td>
</tr>
<tr>
<td>cfsearch</td>
<td>category, categoryTree, status, suggestions, contextPassages, contextBytes, contextHighlightBegin, contextHighlightEnd, previousCriteria attributes natural, internet, and internet_basic values of type attribute</td>
<td>ColdFusion MX 7</td>
</tr>
<tr>
<td>cfselect</td>
<td>selected attribute can take a list enabled, group, height, label, onKeyUp, onKeyDown, onMouseUp, onMouseDown, onChange, onClick, queryPosition, tooltip, visible, and width attributes</td>
<td>ColdFusion MX 7</td>
</tr>
<tr>
<td>cfsetting</td>
<td>requestTimeOut attribute</td>
<td>ColdFusion MX</td>
</tr>
<tr>
<td>cfstoredproc</td>
<td>result attribute</td>
<td>ColdFusion MX 7</td>
</tr>
<tr>
<td>cftextarea</td>
<td>All</td>
<td>ColdFusion MX 7</td>
</tr>
<tr>
<td>cfthrow</td>
<td>object attribute</td>
<td>ColdFusion MX</td>
</tr>
<tr>
<td>cftimer</td>
<td>All</td>
<td>ColdFusion MX 7</td>
</tr>
<tr>
<td>cftree</td>
<td>format, onChange, style attributes</td>
<td>ColdFusion MX 7</td>
</tr>
<tr>
<td>cftrace</td>
<td>All</td>
<td>ColdFusion MX</td>
</tr>
<tr>
<td>cfxml</td>
<td>All</td>
<td>ColdFusion MX</td>
</tr>
</tbody>
</table>
### Deprecated tags, attributes, and values

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

<table>
<thead>
<tr>
<th>Tag</th>
<th>Attribute or value</th>
<th>Deprecated as of this ColdFusion release</th>
</tr>
</thead>
<tbody>
<tr>
<td>cfcache</td>
<td>cachedirectory, timeout attributes</td>
<td>ColdFusion MX</td>
</tr>
<tr>
<td>cfcollection</td>
<td>map and repair options of the action attribute</td>
<td>ColdFusion MX 7</td>
</tr>
<tr>
<td>cferror</td>
<td>monitor option of the exception attribute</td>
<td>ColdFusion MX</td>
</tr>
<tr>
<td>cffile</td>
<td>system value for attributes attribute</td>
<td>ColdFusion MX</td>
</tr>
<tr>
<td></td>
<td>temporary value for attributes attribute</td>
<td>ColdFusion MX</td>
</tr>
<tr>
<td>cfform</td>
<td>passthrough attribute</td>
<td>ColdFusion MX 7</td>
</tr>
<tr>
<td></td>
<td>enableCAB attribute</td>
<td>ColdFusion MX</td>
</tr>
<tr>
<td>cfftp</td>
<td>agentname attribute</td>
<td>ColdFusion MX</td>
</tr>
<tr>
<td>cfgraph</td>
<td>All</td>
<td>ColdFusion MX</td>
</tr>
<tr>
<td>cfgraphdata</td>
<td>All</td>
<td>ColdFusion MX</td>
</tr>
<tr>
<td>cfgridupdate</td>
<td>connectString, dbName, dbServer, dbType, provider, providerDSN attributes</td>
<td>ColdFusion MX</td>
</tr>
<tr>
<td>cfinput</td>
<td>passthrough attribute</td>
<td>ColdFusion MX 7</td>
</tr>
<tr>
<td>cfinstert</td>
<td>connectString, dbName, dbServer, dbType, provider, providerDSN attributes</td>
<td>ColdFusion MX</td>
</tr>
<tr>
<td>cfldap</td>
<td>filterFile attribute</td>
<td>ColdFusion MX</td>
</tr>
<tr>
<td>cflog</td>
<td>date, thread, time attributes</td>
<td>ColdFusion MX</td>
</tr>
<tr>
<td>cfquery</td>
<td>connectString, dbName, dbServer, provider, providerDSN, sql attributes</td>
<td>ColdFusion MX</td>
</tr>
<tr>
<td></td>
<td>The following dbType attribute values: dynamic, ODBC, Oracle73, Oracle80, Sybase11, OLEDB, DB2</td>
<td>ColdFusion MX</td>
</tr>
<tr>
<td>cfregistry</td>
<td>All, on UNIX only</td>
<td>ColdFusion MX</td>
</tr>
<tr>
<td>cfsearch</td>
<td>external, language attributes</td>
<td>ColdFusion MX</td>
</tr>
<tr>
<td>cfselect</td>
<td>passthrough attribute</td>
<td>ColdFusion MX 7</td>
</tr>
<tr>
<td>Tag</td>
<td>Attribute or value</td>
<td>Deprecated as of this ColdFusion release</td>
</tr>
<tr>
<td>-----------------</td>
<td>------------------------------------------------------------------------------------</td>
<td>------------------------------------------</td>
</tr>
<tr>
<td>cfauthenticalet</td>
<td>All</td>
<td>ColdFusion MX</td>
</tr>
<tr>
<td>cfchart</td>
<td>rotated attribute</td>
<td>ColdFusion MX 7</td>
</tr>
<tr>
<td>cffile</td>
<td>attributes attribute value archive</td>
<td>ColdFusion MX</td>
</tr>
<tr>
<td>cfimpersonate</td>
<td>All</td>
<td>ColdFusion MX</td>
</tr>
<tr>
<td>cfindex</td>
<td>action attribute value optimize</td>
<td>ColdFusion MX</td>
</tr>
<tr>
<td>cfldap</td>
<td>fileterConfig attribute</td>
<td>ColdFusion MX</td>
</tr>
<tr>
<td>cfnewinternaladminsecurity</td>
<td>All</td>
<td>ColdFusion MX This tag did not appear in CFML Reference.</td>
</tr>
<tr>
<td>cfsetting</td>
<td>catchExceptionsByPattern attribute</td>
<td>ColdFusion MX</td>
</tr>
</tbody>
</table>

### Obsolete tags, attributes, and values

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

<table>
<thead>
<tr>
<th>Tag</th>
<th>Attribute or value</th>
<th>Obsolete as of this ColdFusion release</th>
</tr>
</thead>
<tbody>
<tr>
<td>cfauthenticate</td>
<td>All</td>
<td>ColdFusion MX</td>
</tr>
<tr>
<td>cfchart</td>
<td>rotated attribute</td>
<td>ColdFusion MX 7</td>
</tr>
<tr>
<td>cffile</td>
<td>attributes attribute value archive</td>
<td>ColdFusion MX</td>
</tr>
<tr>
<td>cfimpersonate</td>
<td>All</td>
<td>ColdFusion MX</td>
</tr>
<tr>
<td>cfindex</td>
<td>action attribute value optimize</td>
<td>ColdFusion MX</td>
</tr>
<tr>
<td>cfldap</td>
<td>fileterConfig attribute</td>
<td>ColdFusion MX</td>
</tr>
<tr>
<td>cfnewinternaladminsecurity</td>
<td>All</td>
<td>ColdFusion MX This tag did not appear in CFML Reference.</td>
</tr>
<tr>
<td>cfsetting</td>
<td>catchExceptionsByPattern attribute</td>
<td>ColdFusion MX</td>
</tr>
</tbody>
</table>
cfabort

Description

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

Category

Flow-control tags

Syntax

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

See also


Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Req/Opt</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>showError</td>
<td>Optional</td>
<td>Error to display, in a standard ColdFusion error page, when tag executes.</td>
<td></td>
</tr>
</tbody>
</table>

Usage

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

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

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

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

Example

This example shows the use of cfabort to stop processing. In the second example, where cfabort is used, the result never displays.

```<h3>Example A: Let the instruction complete itself</h3>
<!--- first, set a variable --->
<cfset myVariable = 3>
<!--- now, perform a loop that increments this value --->
<cfloop from = "1" to = "4" index = "Counter">
    <cfset myVariable = myVariable + 1>
</cfloop>
<cfoutput>
```
Example B: Use cfabort to halt the instructions with showmessage attribute and cferror.

```cfml
<!--- Reset the variable and show the use of cfabort. --->
<cfset myVariable = 3>
<!--- now, perform a loop that increments this value --->
<cfloop from = "1" to = "4" index = "Counter">
<!--- on the second time through the loop, cfabort --->
<cfif Counter is 2>
<!--- Take out the cferror line to see cfabort error processed by CF error page --->
    <cferror type="request" template="request_err.cfm">
        <cfabort showerror="CFABORT has been called for no good reason">
    </cferror>
<!--- Processing is stopped, and subsequent operations are not carried out.--->
<cfelse>
    <cfset myVariable = myVariable + 1>
</cfif>
</cfloop>
```
<p>The value of myVariable after incrementing through the loop #Counter# times is: #myVariable#</p>
</cfoutput>
<p>The value of myVariable after incrementing through the loop #Counter# times is: #myVariable#</p>
</cfoutput>
**cfapplet**

**Description**

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

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

**Category**

Forms tags

**Syntax**

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

**See also**

cfform, cfformgroup, cfformitem, cfgrid, cfinput, cfobject, cfselect, cf servlet, cfslider, cftextarea, cftree

**History**

ColdFusion MX:

- Removed the requirement that you use this tag within a `cfform` tag.
- Changed the behavior when this tag is used within a `cfform` tag; if the `method` attribute is defined in the Administrator, the return value of the applet's method is incorporated into the form.

**Attributes**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Req/Opt</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>appletSource</td>
<td>Required</td>
<td>Name of registered applet</td>
<td></td>
</tr>
<tr>
<td>name</td>
<td>Required</td>
<td>Form variable name for applet</td>
<td></td>
</tr>
<tr>
<td>height</td>
<td>Optional</td>
<td>Height of applet, in pixels</td>
<td></td>
</tr>
<tr>
<td>width</td>
<td>Optional</td>
<td>Width of applet, in pixels</td>
<td></td>
</tr>
<tr>
<td>vSpace</td>
<td>Optional</td>
<td>Space above and below applet, in pixels</td>
<td></td>
</tr>
<tr>
<td>hSpace</td>
<td>Optional</td>
<td>Space on left and right of applet, in pixels</td>
<td></td>
</tr>
</tbody>
</table>
Usage

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

If Java applet components are stored in a JAR file, enter the filename in the ColdFusion Administrator, Java Applets window, Archive text box. For more information, see “Embedding Java applets” in Chapter 27, “Building Dynamic Forms with cfform Tags” in ColdFusion MX Developer’s Guide

Example

```cfcfapplet lets you reference custom Java applets that have been registered using the ColdFusion Administrator.
<cfapplet appletsource = "copytext" name = "copytext">
</cfapplet>
<cfform action = "index.cfm">
    <cfapplet appletsource = "copytext" name = "copytext">
        <p>This example applet copies text that you type into a form. Type some text, and then click "copy" to see the copied text.

        <cfform action = "index.cfm">
            <cfapplet appletsource = "copytext" name = "copytext">
                <p>Text to display if a page that contains a Java applet-based cfform control is opened by a browser that does not support Java or has Java support disabled. For example:

                notSupported = "<b>Browser must support Java to view ColdFusion Java Applets</b>"

                Default:

                <b>Browser must support Java to view ColdFusion Java Applets</b>

                param_n

                Registered parameter for applet. Specify only to override values for applet in ColdFusion Administrator.
```
**cfapplication**

**Description**

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

**Category**

Application framework tags

**Syntax**

```cfapplication
name = "application_name"
loginStorage = "cookie" or "session"
clientManagement = "yes" or "no"
clientStorage = "datasource_name" or "Registry" or "Cookie"
setClientCookies = "yes" or "no"
sessionManagement = "yes" or "no"
sessionTimeout = #CreateTimeSpan(days, hours, minutes, seconds)#
applicationTimeout = #CreateTimeSpan(days, hours, minutes, seconds)#
setDomainCookies = "yes" or "no"
scriptProtect = "none", "all", or list>
```

**See also**

cfassociate, cferror, cflock, cfmodule; Chapter 5, “Application.CFC Reference,” on page 945; Chapter 13, “Designing and Optimizing a ColdFusion Application” and Chapter 37, “Integrating J2EE and Java Elements in CFML Applications,” in ColdFusion MX Developer’s Guide

**History**

ColdFusion MX 7: Added scriptProtect attribute

ColdFusion MX 6.1: Added loginStorage attribute

ColdFusion MX:

- Changed how persistent scopes are available: Server, Session, and Application scope variables are stored in memory as structures. In earlier releases, only Session and Application scope variables were stored this way. You cannot access the UDF function scope as a structure.
- Changed the algorithm for setting the CFTOKEN variable value: if the registry key UUIDToken is a non-zero value, ColdFusion uses a number constructed from the UUID plus a random number. Otherwise, ColdFusion sets the CFTOKEN variable default value using a positive random integer. (In earlier releases, ColdFusion always used a number constructed from the UUID plus a random number.)
<table>
<thead>
<tr>
<th>Attribute</th>
<th>Req/Opt</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>See Description</td>
<td>Name of application. Up to 64 characters. For Application and Session variables: Required. For Client variables: Optional</td>
<td></td>
</tr>
</tbody>
</table>
| loginStorage      | Optional           | cookie  | • cookie: store login information in the Cookie scope.  
                            • session: store login information in the Session scope. |
| clientManagement  | Optional           | no      | • yes: enables client variables.  
                            • no |
| clientStorage     | Optional           | registry | How client variables are stored:  
                            • datasource_name: in ODBC or native data source. You must create storage repository in the Administrator.  
                            • registry: in the system registry.  
                            • cookie: on client computer in a cookie. Scalable. If client disables cookies in the browser, client variables do not work. |
| setClientCookies  | Optional           | yes     | • yes: enables client cookies.  
                            • no: ColdFusion does not automatically send CFID and CFTOKEN cookies to client browser; you must manually code CFID and CFTOKEN on the URL for every page that uses Session or Client variables. |
| sessionManagement | Optional           | no      | • yes: enables session variables.  
                            • no |
| sessionTimeout    | Optional           | Specified in Variables page of ColdFusion Administrator | Lifespan of session variables. CreateTimeSpan function and values in days, hours, minutes, and seconds, separated by commas. |
| applicationTimeout| Optional           | Specified in Variables page of ColdFusion Administrator | Lifespan of application variables. CreateTimeSpan function and values in days, hours, minutes, and seconds, separated by commas. |
This tag is typically used in the Application.cfm file, to set defaults for a ColdFusion application.

**Note:** You can also set the application defaults in the Application.cfc file. For more information, see “Application variables” on page 945.

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

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

ColdFusion generates an error if the application name is longer than 64 characters.

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

**Note:** If you specify `ClientStorage=cookie`, any Client scope variables set following a `cfflush` tag are not saved in the Client browser.

### Protecting variables from cross-site scripting attacks

The `ScriptProtect` attribute lets you protect one or more variable scopes from cross-site scripting attacks, where a client attempts to get your application to send malicious code back to a user’s browser. In these attacks, user input (for example, from form fields or from URL variables) sets a CF variable which is destined for user output. The submitted data includes malicious code, such as JavaScript or an applet or object reference, which then executes on the user’s system.

**Note:** The ColdFusion MX Administrator Settings page Enable Global Script Protection option determines the default script protection setting. You can use the `scriptProtect` attribute to override the Administrator setting. You can also use the Application.cfc initialization code to set the protection value.
The ColdFusion MX cross-site scripting protection operation is done when ColdFusion MX processes the application settings at the beginning of a request. Thus, it can process the URL, and Cookie, CGI, and Form variables in a user's request. By default, it replaces occurrences of the following tag names with the text *InvalidTag*: object, embed, script, applet, and meta. It allows these names in plain text, replaces the words if they are used as tag names.

You can specify any or all ColdFusion scopes for protection, but only the Form, URL, CGI, and Cookie scopes have variables that are often provided by unknown sources. Also, protecting a scope requires additional processing. For these reasons, the `all` attribute value applies protection to only the four scopes.

The script protection mechanism applies a regular expression that is defined in the `cf_root/lib/neo-security.xml` file in the server configuration, or the `cf_root/WEB-INF/cfusion/lib/neo-security.xml` file in the J2EE configuration to the variable value. You can customize the patterns that ColdFusion replaces by modifying the regular expression in the `CrossSiteScriptPatterns` variable.

**Locking server, application, and session variables**

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

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

In some cases, you should also lock code that reads variables in these scopes. For information about locking scopes, see `cflock` on page 270.

**Example**

```cfc
<!--- This example shows how to use cflock to prevent race conditions during
data updates to variables in Application, Server, and Session scopes. --->
<h3>cfapplication Example</h3>
<p>cfapplication defines scoping for a ColdFusion application and enables or
disables application and/or session variable storage. This tag is placed in
a special file called Application.cfm that automatically runs before any
other CF page in a directory (or subdirectory) where the Application.cfm
file appears.</p>

<cfapplication name = "ETurtle"
    sessionTimeout = #CreateTimeSpan(0, 0, 0, 60)#
    sessionManagement = "Yes" #

<!--- Initialize session and application variables used by E-Turtleneck. --->
<cfparam name="application.number" default="1">
<cfparam name="session.color" default= "">
<cfparam name="session.size" default="">
<cfif IsDefined("session.numPurchased") AND
    IsNumeric(trim(session.cartTotal)))>
```

```cfc```
<!--- Use the application scope for the application variable to prevent race condition. This variable keeps track of total number of turtlenecks sold. --->
<cflock scope = "Application" timeout = "30" type = "Exclusive">
<cfset application.number = application.number + session.numPurchased>
</cflock>
</cfif>

<cfoutput>
E-Turtleneck is proud to say that we have sold #application.number# turtlenecks to date.
</cfoutput>
<!--- End of Application.cfm --->

<!--- End of Application.cfm --->
**cfargument**

**Description**

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

**Category**

*Extensibility tags*

**Syntax**

```xml
<cfargument
  name="string"
  type="data type"
  required="yes" or "no"
  default="default value"
  displayname="descriptive name"
  hint="extended description">
```

**See also**

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

**Attributes**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Req/Opt</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Required</td>
<td>String; an argument name.</td>
<td></td>
</tr>
<tr>
<td>type</td>
<td>Optional</td>
<td>any</td>
<td>String; a type name; data type of the argument.</td>
</tr>
</tbody>
</table>

- any
- array
- binary
- boolean
- date
- guid: the argument must be a UUID or GUID of the form `xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx` where each `x` is a character representing a hexadecimal number (0-9A-F).
- numeric
- query
- string
- struct
- uuid: the argument must be a ColdFusion UUID of the form `xxxxxxxx-xxxx-xxxx-xxxxxxxxxxxxxxxx` where each `x` is a character representing a hexadecimal number (0-9A-F).
- variableName: a string formatted according to ColdFusion variable naming conventions.
- xml: XML objects and XML strings
- a component name: if the type attribute value is not one of the preceding items, ColdFusion treats it as the name of a ColdFusion component. When the function executes, it generates an error if the argument that is passed in is not a CFC with the specified name.
Usage

This tag must be in a `<cffunction>` tag, before any other tags in the `<cffunction>` tag body.

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

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

Example

```cfml
<!--- This example defines a function that takes a course number parameter
and returns the course description. --->
<cffunction name="getDescription">
  <!--- Identify argument --->
  <cfargument name="Course_Number" type="numeric" required="true">
  <!--- Use the argument to get a course description from the database --->
  <cfquery name="Description" datasource="cfdocexamples">
    SELECT Descript
    FROM Courses
    WHERE Number = '#Course_Number#'  
  </cfquery>
  <!--- Specify the variable that the function returns --->
  <cfreturn Description.Descript>
</cffunction>
```

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Req/Opt</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>required</td>
<td>Optional</td>
<td>no</td>
<td><strong>Note:</strong> All arguments are required when invoked as a web service, irrespective of how they are defined. Specifies whether the parameter is required to execute the component method. The parameter is not required if you specify a <code>default</code> attribute.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• true or &quot;yes&quot;</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• false or &quot;no&quot;</td>
</tr>
<tr>
<td>default</td>
<td>Optional</td>
<td></td>
<td>If no argument is passed, specifies a default argument value.</td>
</tr>
<tr>
<td>displayname</td>
<td>Optional</td>
<td></td>
<td>Meaningful only for CFC method parameters. A value to be displayed when using introspection to show information about the CFC.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>name</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>attribute value</td>
<td>Meaningful only for CFC method parameters. Text to be displayed when using introspection to show information about the CFC. The <code>hint</code> attribute value follows the <code>displayname</code> attribute value in the parameter description line. This attribute can be useful for describing the purpose of the parameter.</td>
</tr>
</tbody>
</table>
**cfassociate**

**Description**

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

**Category**

Application framework tags

**Syntax**

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

**See also**


**Attributes**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Req/Opt</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>baseTag</td>
<td>Required</td>
<td></td>
<td>Base tag name.</td>
</tr>
<tr>
<td>dataCollection</td>
<td>Optional</td>
<td>AssocAttribs</td>
<td>Structure in which base tag stores subtag data.</td>
</tr>
</tbody>
</table>

**Usage**

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

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

Within the custom tag code, the attributes passed to the tag by using the `attributeCollection` attribute are saved as independent values, with no indication that they are grouped into a structure by the custom tag’s caller. Therefore, in the called tag, if you assign a value to a specific attribute, it replaces the value passed in the `attributeCollection` attribute that you used when calling the subtag.

**Example**

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

    <!--- Define defaults for attributes. --->
    <cfparam name = "attributes.happy" default = "yes">
    <cfparam name = "attributes.sad" default = "no">
    ...
```
cfauthenticate

Description

This tag is obsolete. Use the newer security tools; see “Conversion functions” on page 453 and Chapter 16, “Securing Applications” in *ColdFusion MX Developer’s Guide*.

History

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

**Description**

Used within a `cfloop` or `cfswitch` tag. Breaks out of a loop or switch block.

**Category**

Flow-control tags

**Syntax**

```html
<cfbreak>
```

**See also**

`cfabort`, `cfexecute`, `cfif`, `cflocation`, `cfloop`, `cfswitch`, `cfthrow`, `cftry`; “cfloop and `cfbreak`” in Chapter 2, “Elements of CFML,” in *ColdFusion MX Developer's Guide*

**Example**

```html
<!--- This shows the use of cfbreak to exit a loop when a condition is met. --->
<!--- Select courses; use cfloop to find a condition; then break the loop. --->
<!--- Check that number is numeric. --->
<cfif IsDefined("form.course_number")>
    <cfif Not IsNumeric(form.course_number)>
        <cfabort>
    </cfif>
</cfif>
<cfquery name="GetCourses" datasource="cfdocexamples">
    SELECT *
    FROM Courses
    ORDER by course_number
</cfquery>

<p> This example uses CFLOOP to cycle through a query to find a value. (In our example, a list of values corresponding to courses in the Snippets datasource). When the conditions of the query are met, CFBREAK stops the loop.</p>

<p> Please enter a Course Number, and hit the "submit" button:</p>

```html
<form action="cfbreak.cfm" method="POST">
    <select name="courseNum">
        <cfoutput query="GetCourses">
            <option value="#course_number#">#course_number#</option>
        </cfoutput>
    </select>
    <input type="Submit" name="" value="Search on my Number">
</form>
```

<!--- If the courseNum variable is not defined, don't loop through the query.--->
<cfif IsDefined (*form.courseNum*) IS "True">
<!--- Loop through query until value found, then use CFBREAK to exit query.--->
<cfloop query="GetCourses">
    <cfif GetCourses.course_number IS form.courseNum>
        <cfoutput>
            <h4>Your Desired Course was found:</h4>
            <pre>#course_number# #descript#</pre>
        </cfoutput>
        <cfbreak>
    </cfif>
</cfloop>
```
<cfelse>
  <br> Searching...
  </cfif>
</cfloop>
</cfif>
cfcache

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

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

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

Category
Page processing tags

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

See also
cfflush, cfheader, cfhtmlhead, cfsetting, cfsilent

History
ColdFusion MX:

- Deprecated the cachedirectory and timeout attributes. They might not work, and might cause an error, in later releases.
- Added the timespan attribute.
- Changed how pages are cached: the default action attribute value, cache, caches a page on the server and the client. (In earlier releases, this option cached a page only on the server.)
- Changed the source of the protocol and port values: the default protocol and port values are now taken from the current page URL. (In earlier releases, they were "http" and "80", respectively.)
- Changed how session state is handled when caching a page: this tag can cache pages that depend on session state, including pages that are secured with a ColdFusion login. (In earlier releases, the session state was cleared when caching the page, causing authentication to be lost.)
- Changed how files are cached: this tag uses a hash() of the URL for the filename to cache a file. (In earlier releases, ColdFusion used the cfcache.map file.)
Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Req/Opt</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>action</td>
<td>Optional</td>
<td>cache</td>
<td>• cache: server-side and client-side caching.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• flush: refresh cached page(s).</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• clientcache: browser-side caching only.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• servercache: server-side caching only.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• optimal: same as &quot;cache&quot;.</td>
</tr>
<tr>
<td>directory</td>
<td>Optional</td>
<td>cf_root/cache</td>
<td>Absolute path of cache directory.</td>
</tr>
<tr>
<td>timespan</td>
<td>Optional</td>
<td></td>
<td>The interval until the page is flushed from the cache.</td>
</tr>
<tr>
<td>expireURL</td>
<td>Optional</td>
<td></td>
<td>Used with action=&quot;flush&quot;. A URL reference.</td>
</tr>
<tr>
<td>username</td>
<td>Optional</td>
<td></td>
<td>A username. Provide this if the page requires</td>
</tr>
<tr>
<td>password</td>
<td>Optional</td>
<td></td>
<td>A password. Provide this if the page requires</td>
</tr>
<tr>
<td>port</td>
<td>Optional</td>
<td></td>
<td>Port number of the web server from which the URL is</td>
</tr>
<tr>
<td>protocol</td>
<td>Optional</td>
<td></td>
<td>Protocol that is used to create URL from cache.</td>
</tr>
</tbody>
</table>

Usage

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

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

To enable a simple form of caching, put a cfcache tag, specifying the timespan attribute, at the top of a page. Each time the specified time span passes, ColdFusion flushes (deletes) the copy of the page from the cache and caches a new copy for users to access.
You can specify client-side caching or a combination of client-side and server-side caching (the default), using the action attribute. The advantage of client-side caching is that it requires no ColdFusion resources; the browser stores pages in its own cache, improving performance. The advantage of combination caching is that it optimizes server performance; if the browser does not have a cache of the page, the server can get data from its own cache. (Macromedia recommends that you do not use server-side only caching. Macromedia recommends that you use combination caching.)

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

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

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

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

For more information, see “Caching ColdFusion pages that change infrequently” in Chapter 13, “Optimizing ColdFusion applications,” in *ColdFusion MX Developer’s Guide*.

**Example**

<!-- This example produces as many cached files as there are URL parameter permutations. You can see that the page is cached when the timestamp doesn't change.--->

```cfml
<cfcache timespan="#createTimeSpan(0,0,10,0)#">
<body>
<h3>This is a test of some simple output</h3>
<cfoutput>
This page was generated at #now()#
</cfoutput>
<cfparam name="URL.x" default="no URL parm passed"><cfoutput>
The value of URL.x = # URL.x #</cfoutput>
</cfoutput>
</cfparam>
</body>
</cfcache>
```
cfcalendar

Description

Puts an interactive Macromedia Flash format calendar in an HTML or Flash form. Not supported in XML format forms. The calendar lets a user select a date for submission as a form variable.

Category

Forms tags

Syntax

```html
<cfcalendar
    name = "name of calendar"
    height = "height"
    width = "width"
    selectedDate = "date"
    startRange = "first disabled date"
    endRange = "last disabled date"
    disabled = "true", "false", or no attribute value
    mask = "character pattern"
    dayNames = "days-of-the-week labels"
    monthNames = "month labels"
    visible = "Yes" or "No"
    enabled = "Yes" or "No"
    tooltip = "Tip text"
    onChange = "actionscript to invoke">
```

See also


History

ColdFusion MX 7: Added tag.

Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Req/Opt</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Required</td>
<td>Determined by Flash</td>
<td>The name of the calendar.</td>
</tr>
<tr>
<td>height</td>
<td>Optional</td>
<td>Determined by Flash</td>
<td>The vertical dimension of the calendar specified in pixels.</td>
</tr>
<tr>
<td>width</td>
<td>Optional</td>
<td>Determined by Flash</td>
<td>The horizontal dimension of the calendar specified in pixels.</td>
</tr>
<tr>
<td>selectedDate</td>
<td>Optional</td>
<td>None (Flash shows the current month)</td>
<td>The date that is initially selected. It is highlighted in a color determined by the form skin. Must be in mm/dd/yyyy or dd/mm/yyyy format, depending on the current locale. (Use the setlocale tag to set the locale, if necessary.)</td>
</tr>
<tr>
<td>Attribute</td>
<td>Req/Opt</td>
<td>Default</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------</td>
<td>---------</td>
<td>---------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>startRange</td>
<td>Optional</td>
<td></td>
<td>The start of a range of dates that are disabled. Users cannot select dates from this date through the date specified by the <code>endRange</code> attribute.</td>
</tr>
<tr>
<td>endRange</td>
<td>Optional</td>
<td></td>
<td>The end of a range of dates that are disabled. Users cannot select dates from the date specified by the <code>startRange</code> attribute through this date.</td>
</tr>
<tr>
<td>disabled</td>
<td>Optional</td>
<td>not disabled</td>
<td>Disables all user input, making the control read-only. To disable input, specify <code>disabled</code> without an attribute or <code>disabled=&quot;yes&quot;</code> (or any ColdFusion positive boolean value, such as True). To enable input, omit the attribute or specify <code>disabled=&quot;no&quot;</code> (or any ColdFusion negative boolean value, such as False).</td>
</tr>
<tr>
<td>mask</td>
<td>Optional</td>
<td>MM/DD/YYYY</td>
<td>A pattern that specifies the format of the submitted date. Mask characters are:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• D = day; can use 0-2 mask characters</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• M = month; can use 0-4 mask characters</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Y = year; can use 0, 2, or 4 characters</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• E = day in week; can use 0-4 characters</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Any other character = put the character in the specified location</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>For more information on masking, see &quot;Masking input data&quot; in the <code>cfinput</code> reference page.</td>
</tr>
<tr>
<td>firstDayOfWeek</td>
<td>Optional</td>
<td>0</td>
<td>Integer in the range 0-6 specifying the first day of the week in the calendar: 0 indicates Sunday; 6 indicates Saturday.</td>
</tr>
<tr>
<td>dayNames</td>
<td>Optional</td>
<td>S, M, T, W, Th, F, S</td>
<td>A comma-delimited list that sets the names of the weekdays displayed in the calendar. Sunday is the first day and the rest of the weekday names follow in the normal order.</td>
</tr>
<tr>
<td>monthNames</td>
<td>Optional</td>
<td>January, February, March, April, May, June, July, August, September, October, November, December</td>
<td>A comma-delimited list of the month names that are displayed at the top of the calendar.</td>
</tr>
<tr>
<td>style</td>
<td>Optional</td>
<td></td>
<td>Flash ActionScript style or styles to apply to the calendar. For more information, see Chapter 29, &quot;Setting styles and skins in Flash forms&quot; in ColdFusion MX Developer’s Guide.</td>
</tr>
<tr>
<td>enabled</td>
<td>Optional</td>
<td>Yes</td>
<td>Flash only. Boolean value specifying whether the control is enabled. A disabled control appears in light gray. This is the inverse of the disabled attribute.</td>
</tr>
</tbody>
</table>
The `cfcalendar` tag displays a calendar month, showing the month, the year, a grid of the days of the month, and headings for the days of the week. The calendar contains forward and back arrow buttons to let you change the month and year that are displayed.

If you include a value for the `selectedDate` attribute, that date is highlighted in green and determines the month and year that display initially. Changing the month and year display does not change the selected date. A user can change the selected date by clicking a different date on the calendar. The `onChange` attribute can specify an ActionScript event handler function that runs when the user selects a date.

The current date is highlighted in reverse (that is, a white number on a black background). If the selected date is in a different month or year, however, the current date does not appear unless you move to it by clicking the forward or back arrows.

The `mask` attribute lets you specify the format of the selected date that is returned to the application.

You can use the keyboard to access and select dates from a `cfcalendar` control:

- Use the Up, Down, Left, and Right Arrow keys to change the selected date.
- Use the Home and End keys to reach the first and last enabled date in a month, respectively.
- Use the Page Up and Page Down keys to reach the previous and next month, respectively.

**Note:** The `cfcalendar` tag is not supported in XML format forms.

### Example

This example produces a 200-pixel by 150-pixel calendar with a Flash haloBlue skin. It displays abbreviated month names and two-character days of the week. It initially displays today's date as determined by the `selectedDate` attribute. When you click the Save button, the form submits back to the current page, which displays the submitted information.

The example also has three `dateField` controls that let the user change the initial selected date that displays on the calendar and a blocked-out date range. The initial blocked-out date is a four-day period immediately preceding today's date.

**Note:** This example must be modified to work in locales that do not use mm/dd/yyyy date formats. To do so, use the `LSDateFormat` function in place of the `DateFormat` function and a mask that is appropriate for your locale, such as dd/mm/yyyy.

```html
<!--- Set initial selected and blocked-out dates.--->
<cfparam name="Form.startdate" default="#dateformat(now()-5, 'mm/dd/yyyy')#">
```

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Req/Opt</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>visible</td>
<td>Optional</td>
<td>Yes</td>
<td>Flash only: Boolean value specifying whether to show the control. Space that would be occupied by an invisible control is blank.</td>
</tr>
<tr>
<td>tooltip</td>
<td>Optional</td>
<td></td>
<td>Flash only: Text to display when the mouse pointer hovers over the control.</td>
</tr>
<tr>
<td>onChange</td>
<td>Optional</td>
<td></td>
<td>ActionScript that runs when the user selects a date.</td>
</tr>
</tbody>
</table>
<cfparam name="Form.enddate" default="#dateformat(now()-1, 'mm/dd/yyyy')#">
<cfparam name="Form.selectdate" default="#dateformat(now(), 'mm/dd/yyyy')#">

<!--- If the form has been submitted, display the selected date. --->
<cfif isDefined("Form.submitit")>
    <cfoutput><b>You selected #Form.selectedDate#</b><br><br></cfoutput>
</cfif>

<b>Please select a date on the calendar and click Save.</b><br>
<br>
<cfform name="form1" format="Flash" skin="haloBlue" width="375" height="350" >
    <cfcalendar name="selectedDate" 
                selectedDate="#Form.selectdate#" 
                startRange="#Form.startdate#" 
                endRange="#Form.enddate#" 
                mask="mmm dd, yyyy" 
                dayNames="SU,MO,TU,WE,TH,FR,SA" 
                monthNames="JAN, FEB, MAR, APR, MAY, JUN, JUL, AUG, SEP, OCT, NOV, DEC" 
                style="rollOverColor:##FF0000" 
                width="200" height="150" >
        <cfinput type="dateField" name="startdate" label="Block out starts" width="100" value="#Form.startdate#"> 
        <cfinput type="dateField" name="enddate" label="Block out ends" width="100" value="#Form.enddate#"> 
        <cfinput type="dateField" name="selectdate" label="Initial date" width="100" value="#Form.selectdate#"> 
        <cfinput type="Submit" name="submitit" value="Save" width="100">
    </cfform>
cfcasase

Description

Used only inside the cfswitch tag body. Contains code to execute when the expression specified in the cfswitch tag has one or more specific values.

Category

Flow-control tags

Syntax

<cfcase
   value = "value or delimited set of values"
   delimiters = "delimiter characters">

See also


Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Req/Opt</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>Required</td>
<td></td>
<td>The value or values that the expression attribute of the cfswitch tag must match. To specify multiple matching values, separate the values with the delimiter character. The value or values must be simple constants or constant expressions, not variables.</td>
</tr>
<tr>
<td>delimiter</td>
<td>Optional</td>
<td>,(comma)</td>
<td>Specifies the delimiter character or characters that separate multiple values to match. If you specify multiple delimiter characters, you can use any of them to separate the values to be matched.</td>
</tr>
</tbody>
</table>

Usage

The contents of the cfcasase tag body executes only if the expression attribute of the cfswitch tag evaluates to a value specified by the value attribute. The contents of the cfcasase tag body can include HTML and text, and CFML tags, functions, variables, and expressions. You do not have to explicitly break out of the cfcasase tag, as you do in some languages.

One cfcasase tag can match multiple expression values. To do this, separate the matching values with the delimiter character, which is the comma by default. For example the following line matches "red", "blue", or "green":
<cfcase value="red,blue,green">

You can use the delimiter attribute to specify one or more delimiters to use in place of the comma. For example, the following line matches "cargo, live", "cargo, liquid", and "cargo, solid":
<cfcase value="cargo, live;cargo, liquid-cargo, solid" delimiters=";:-">

Example

The following example displays a grade based on a 1-10 score. Several of the cfcasase tags match more than one score. For simplicity, the example sets the score to 7.
<cfset score="7" />
<cfswitch expression="#score#">
  <cfcase value="10">
    <cfset grade="A" />
  </cfcase>
  <cfcase value="9:8" delimiters=":">
    <cfset grade="B" />
  </cfcase>
  <cfcase value="7:6" delimiters=":">
    <cfset grade="C" />
  </cfcase>
  <cfcase value="5:4;" delimiters=";" />
    <cfset grade="D" />
  </cfcase>
  <cfdefaultcase>
    <cfset grade="F" />
  </cfdefaultcase>
</cfswitch>
<cfoutput>
  Your grade is #grade#
</cfoutput>
**cfcatch**

**Description**

Used inside a `cftry` tag. Together, they catch and process exceptions in ColdFusion pages. **Exceptions** are events that disrupt the normal flow of instructions in a ColdFusion page, such as failed database operations, missing include files, and developer-specified events.

**Category**

*Exception handling tags*

**Syntax**

```html
<cfcatch type = "exceptiontype">
  Exception processing code here
</cfcatch>
```

**See also**

*cftry, cfferror, cffrethrow, cffthrow, onError; Chapter 14, “Handling Errors,” in *ColdFusion MX Developer’s Guide* *

**History**

ColdFusion MX:

- Changed SQLSTATE value behavior: the SQLSTATE return value in a `cfcatch` tag depends on the database driver type:
  - Type 1 (JDBC-ODBC bridge): the value is the same as in ColdFusion 5
  - Type 4 (100% Java, no native methods): the value might be different
  If your application depends on SQLSTATE values for flow control, the application might produce unexpected behavior with ColdFusion MX.
- Changed the behavior of this tag when `type="any"`: it is not necessary, when you include a `cfcatch` tag with `type="any"`, to do so in the last `cfcatch` tag in the block, to ensure that all other tests are executed before it. ColdFusion finds the best-match `cfcatch` block.
- Changed the behavior of the `cfscript` tag; it includes `try` and `catch` statements that are equivalent to the `cftry` and `cfcatch` tags.
- Changed object modification: you cannot modify the object returned by `cfcatch`.
- Changed thrown exceptions: the `cfcollection`, `cfindex`, and `cfsearch` tags can throw the SEARCHENGINE exception. In earlier releases, an error in processing these tags threw only an UNKNOWN exception.
Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Req/Opt</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>Optional</td>
<td>Any</td>
<td></td>
</tr>
</tbody>
</table>

- application: catches application exceptions
- database: catches database exceptions
- template: catches ColdFusion page exceptions
- security: catches security exceptions
- object: catches object exceptions
- missingInclude: catches missing include file exceptions
- expression: catches expression exceptions
- lock: catches lock exceptions
- custom_type: catches the specified custom exception type that is defined in a cfthrow tag
- searchengine: catches Verity search engine exceptions
- any: catches all exception types

Usage

You must code at least one cfcatch tag within a cftry block. Put cfcatch tags at the end of a cftry block. ColdFusion MX tests cfcatch tags in the order in which they appear. This tag requires an end tag.

If type="any", ColdFusion MX catches exceptions from any CFML tag, data source, or external object. To get the exception type use code such as the following:

```cfml
#cfcatch.type#
```

Applications can use the cfthrow tag to throw developer-defined exceptions. Catch these exceptions with any of these type options:

- "custom_type"
- "Application"
- "Any"

The custom_type type is a developer-defined type specified in a cfthrow tag. If you define a custom type as a series of strings concatenated by periods (for example, "MyApp.BusinessRuleException.InvalidAccount"), ColdFusion MX can catch the custom type by its character pattern. ColdFusion MX searches for a cfcatch tag in the cftry block with a matching exception type, starting with the most specific (the entire string), and ending with the least specific.

For example, you could define a type as follows:

```cfml
cfthrow type = "MyApp.BusinessRuleException.InvalidAccount"
```

If you have the following cfcatch tag, it will handle the exception:

```cfml
cfcatch type = "MyApp.BusinessRuleException.InvalidAccount"
```

Otherwise, if you have the following cfcatch tag, it will handle the exception:

```cfml
cfcatch type = "MyApp.BusinessRuleException"
```

Finally, if you have the following cfcatch tag, it will handle the exception:

```cfml
cfcatch type = "MyApp"
```
You can code `cfcatch` tags in any order to catch a custom exception type.

If you specify `type = "Application"`, the `cfcatch` tag catches only custom exceptions that have the `Application` type in the `cfthrow` tag that defines them.

The `cfinclude`, `cfmodule`, and `cferror` tags throw an exception of `type = "template"`.

An exception that is thrown within a `cfcatch` block cannot be handled by the `cftry` block that immediately encloses the `cfcatch` tag. However, you can rethrow the currently active exception with the `cfrethrow` tag.

The `cfcatch` variables provide the following exception information:

<table>
<thead>
<tr>
<th><code>cfcatch</code> variable</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>cfcatch.type</code></td>
<td>Type: Exception type, as specified in <code>cfcatch</code>.</td>
</tr>
<tr>
<td><code>cfcatch.message</code></td>
<td>Message: Exception's diagnostic message, if provided; otherwise, an empty string; in the <code>cfcatch.message</code> variable.</td>
</tr>
<tr>
<td><code>cfcatch.detail</code></td>
<td>Detailed message from the CFML interpreter or specified in a <code>cfthrow</code> tag. When the exception is generated by ColdFusion (and not <code>cfthrow</code>), the message can contain HTML formatting and can help determine which tag threw the exception.</td>
</tr>
<tr>
<td><code>cfcatch.tagcontext</code></td>
<td>An array of tag context structures, each representing one level of the active tag context at the time of the exception.</td>
</tr>
<tr>
<td><code>cfcatch.SQLState</code></td>
<td>Applies to <code>type = &quot;database&quot;</code>. SQLState associated with exception. Database drivers typically provide error codes to help diagnose failing database operations. Default: -1.</td>
</tr>
<tr>
<td><code>cfcatch.Sql</code></td>
<td>Applies to <code>type = &quot;database&quot;</code>. The SQL statement sent to the data source.</td>
</tr>
<tr>
<td><code>cfcatch.queryError</code></td>
<td>Applies to <code>type = &quot;database&quot;</code>. The error message as reported by the database driver.</td>
</tr>
<tr>
<td><code>cfcatch.where</code></td>
<td>Applies to <code>type = &quot;database&quot;</code>. If the query uses the <code>cfqueryparam</code> tag, query parameter name-value pairs.</td>
</tr>
<tr>
<td><code>cfcatch.ErrNumber</code></td>
<td>Applies to <code>type = &quot;expression&quot;</code>. Internal expression error number.</td>
</tr>
<tr>
<td><code>cfcatch.MissingFileName</code></td>
<td>Applies to <code>type = &quot;missingInclude&quot;</code>. Name of file that could not be included.</td>
</tr>
<tr>
<td><code>cfcatch.LockName</code></td>
<td>Applies to <code>type = &quot;lock&quot;</code>. Name of affected lock (if the lock is unnamed, the value is &quot;anonymous&quot;).</td>
</tr>
<tr>
<td><code>cfcatch.LockOperation</code></td>
<td>Applies to <code>type = &quot;lock&quot;</code>. Operation that failed (Timeout, Create Mutex, or Unknown).</td>
</tr>
<tr>
<td><code>cfcatch.ErrorCode</code></td>
<td>Applies to <code>type = &quot;custom&quot;</code>. String error code.</td>
</tr>
<tr>
<td><code>cfcatch.ExtendedInfo</code></td>
<td>Applies to <code>type = &quot;application&quot; and &quot;custom&quot;</code>. Custom error message; information that the default exception handler does not display.</td>
</tr>
</tbody>
</table>
### Advanced Exception types

You can specify the following advanced exception types in the `type` attribute:

<table>
<thead>
<tr>
<th>ColdFusion advanced exception type</th>
</tr>
</thead>
<tbody>
<tr>
<td>COM.Allaire.ColdFusion.CFEXECUTE.OutputError</td>
</tr>
<tr>
<td>COM.Allaire.ColdFusion.CFEXECUTE.Timeout</td>
</tr>
<tr>
<td>COM.Allaire.ColdFusion.FileException</td>
</tr>
<tr>
<td>COM.Allaire.ColdFusion.HTTPAccepted</td>
</tr>
<tr>
<td>COM.Allaire.ColdFusion.HTTPAuthFailure</td>
</tr>
<tr>
<td>COM.Allaire.ColdFusion.HTTPBadGateway</td>
</tr>
<tr>
<td>COM.Allaire.ColdFusion.HTTPBadRequest</td>
</tr>
<tr>
<td>COM.Allaire.ColdFusion.HTTPCFHTTPRequestEntityTooLarge</td>
</tr>
<tr>
<td>COM.Allaire.ColdFusion.HTTPCGIValueNotPassed</td>
</tr>
<tr>
<td>COM.Allaire.ColdFusion.HTTPConflict</td>
</tr>
<tr>
<td>COM.Allaire.ColdFusion.HTTPContentLengthRequired</td>
</tr>
<tr>
<td>COM.Allaire.ColdFusion.HTTPContinue</td>
</tr>
<tr>
<td>COM.Allaire.ColdFusion.HTTPCookieValueNotPassed</td>
</tr>
<tr>
<td>COM.Allaire.ColdFusion.HTTPCreated</td>
</tr>
<tr>
<td>COM.Allaire.ColdFusion.HTTPFailure</td>
</tr>
<tr>
<td>COM.Allaire.ColdFusion.HTTPFileInvalidPath</td>
</tr>
<tr>
<td>COM.Allaire.ColdFusion.HTTPFileNotFound</td>
</tr>
<tr>
<td>COM.Allaire.ColdFusion.HTTPFileNotPassed</td>
</tr>
<tr>
<td>COM.Allaire.ColdFusion.HTTPFileNotRenderable</td>
</tr>
<tr>
<td>COM.Allaire.ColdFusion.HTTPForbidden</td>
</tr>
<tr>
<td>COM.Allaire.ColdFusion.HTTPGatewayTimeout</td>
</tr>
<tr>
<td>COM.Allaire.ColdFusion.HTTPGone</td>
</tr>
<tr>
<td>COM.Allaire.ColdFusion.HTTPMethodNotAllowed</td>
</tr>
<tr>
<td>COM.Allaire.ColdFusion.HTTPMovedPermanently</td>
</tr>
<tr>
<td>COM.Allaire.ColdFusion.HTTPMovedTemporarily</td>
</tr>
<tr>
<td>COM.Allaire.ColdFusion.HTTPMultipleChoices</td>
</tr>
<tr>
<td>COM.Allaire.ColdFusion.HTTPNoContent</td>
</tr>
<tr>
<td>COM.Allaire.ColdFusion.HTTPNonAuthoritativeInfo</td>
</tr>
<tr>
<td>COM.Allaire.ColdFusion.HTTPNotAcceptable</td>
</tr>
<tr>
<td>COM.Allaire.ColdFusion.HTTPNotFound</td>
</tr>
<tr>
<td>COM.Allaire.ColdFusion.HTTPNotImplemented</td>
</tr>
</tbody>
</table>
Example

<!--- cfcatch example, using TagContext to display the tag stack. --->
<h3>cftry Example</h3>
<!--- Open a cftry block. --->
<cftry>
<!--- Note misspelled tablename "employees" as "employeeas". --->
<cfquery name = "TestQuery" dataSource = "cfdocexamples">
SELECT * FROM EMPLOYEEAS
</cfquery>
<!--- Other processing goes here. --->
<!--- Specify the type of error for which we search. --->
<cfcatch type = "Database">
<!--- the message to display. --->
<h3>You've Thrown a Database <b>Error</b></h3>
<cfoutput>
<!--- The diagnostic message from ColdFusion MX. --->
<p>#cfcatch.message#</p>

---

ColdFusion advanced exception type

COM.Allaire.ColdFusion.HTTPNotModified
COM.Allaire.ColdFusion.HTTPPartialContent
COM.Allaire.ColdFusion.HTTPPaymentRequired
COM.Allaire.ColdFusion.HTTPPreconditionFailed
COM.Allaire.ColdFusion.HTTPProxyAuthenticationRequired
COM.Allaire.ColdFusion.HTTPRequestURITooLarge
COM.Allaire.ColdFusion.HTTPResetContent
COM.Allaire.ColdFusion.HTTPSeeOther
COM.Allaire.ColdFusion.HTTPServerError
COM.Allaire.ColdFusion.HTTPServiceUnavailable
COM.Allaire.ColdFusion.HTTPSwitchingProtocols
COM.Allaire.ColdFusion.HTTPUnsupportedMediaType
COM.Allaire.ColdFusion.HTTPUrlValueNotPassed
COM.Allaire.ColdFusion.HTTPUseProxy
COM.Allaire.ColdFusion.HTTPVersionNotSupported
COM.Allaire.ColdFusion.POPAuthFailure
COM.Allaire.ColdFusion.POPConnectionFailure
COM.Allaire.ColdFusion.POPDeleteError
COM.Allaire.ColdFusion.Request.Timeout
COM.SERVLETJRunError
COM.COM.COM.Allaire.ColdFusion.HTTPConnectionTimeout
Caught an exception, type = #CFCATCH.TYPE#  

The contents of the tag stack are:

<cfdump var="#cfcatch.tagcontext#"/>
cfchart

Description
Generates and displays a chart.

Category
Data output tags, Extensibility tags; “Controlling chart appearance” in Chapter 31, “Creating Charts and Graphs,” in ColdFusion MX Developer’s Guide

Syntax
Syntax 1
<!--- This syntax uses an XML file or string to specify the chart style. --->
<cfchart
   style = "XML string or filename">
</cfchart>

Syntax 2
<!--- This syntax uses the attributes of the cfchart tag to specify the chart style. --->
<cfchart
   backgroundColor = "Hex value or Web color"
   chartHeight = "integer number of pixels"
   chartWidth = "integer number of pixels"
   dataBackgroundColor = "Hex value or Web color"
   font = "font name"
   fontBold = "yes" or "no"
   fontItalic = "yes" or "no"
   fontSize = "integer font size"
   foregroundColor = "Hex value or Web color"
   format = "flash" or "jpg" or "png"
   gridlines = "integer number of lines"
   labelFormat = "number, currency, percent, date"
   markerSize = "integer number of pixels"
   name = "String"
   pieSliceStyle = "solid, sliced"
   scaleFrom = "integer minimum value"
   scaleTo = "integer maximum value"
   seriesPlacement = "default, cluster, stacked, percent"
   show3D = "yes" or "no"
   showBorder = "yes" or "no"
   showLegend = "yes" or "no"
   showMarkers = "yes" or "no"
   showXGridlines = "yes" or "no"
   showYGridlines = "yes" or "no"
   sortXAxis = "yes" or "no"
   tipBGColor = "hex value or web color"
   tipStyle = "MouseDown, MouseOver, none"
   title = "title of chart"
   url = "onClick destination page"
   xAxisTitle = "title text"
   xAxisType = "scale or category"
   xOffset = "number between -1 and 1"
   yAxisTitle = "title text"
   yAxisType = "scale or category"
   yOffset = "number between -1 and 1"
</cfchart>
See also
cfchartdata, cfchartseries

History
ColdFusion MX 7:
• Added style and title attributes.
• Added support for eight-digit hexadecimal values to specify RGB color and transparency.
• Removed the rotated attribute.
• Renamed the column chart type to be horizontalbar chart type.

ColdFusion MX 6.1:
• Added the xAxisType and yAxisType attributes.
• Changed interpolation behavior: the tag now interpolates data points on line charts with multiple series.

ColdFusion MX: Added this tag.

Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Req/Opt</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>backgroundColor</td>
<td>Optional</td>
<td></td>
<td>Color of the area between the data background and the chart border, around labels and around the legend. Hexadecimal value or supported named color; see the name list in Usage. For a hexadecimal value, use the form &quot;##xxxxxx&quot; or &quot;##xxxxxxxx&quot;, where x = 0-9 or A-F; use two number signs or none.</td>
</tr>
<tr>
<td>chartHeight</td>
<td>Optional</td>
<td>240</td>
<td>Chart height; integer number of pixels.</td>
</tr>
<tr>
<td>chartWidth</td>
<td>Optional</td>
<td>320</td>
<td>Chart width; integer number of pixels.</td>
</tr>
<tr>
<td>dataBackgroundColor</td>
<td>Optional</td>
<td>white</td>
<td>Color of area around chart data. Hexadecimal value or supported named color; see the name list in Usage. For a hexadecimal value, use the form &quot;##xxxxxx&quot; or &quot;##xxxxxxxx&quot;, where x = 0-9 or A-F; use two number signs or none.</td>
</tr>
<tr>
<td>font</td>
<td>Optional</td>
<td>arial</td>
<td>Name of text font: • arial • times • courier • arialunicodeMS. This option is required, if you are using a double-byte character set on UNIX, or using a double-byte character set in Windows with a file type of Flash.</td>
</tr>
<tr>
<td>Attribute</td>
<td>Req/Opt</td>
<td>Default</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------</td>
<td>---------</td>
<td>---------</td>
<td>----------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
</tbody>
</table>
| fontBold             | Optional| no      | Whether to make the text bold:  
• yes  
• no  |
| fontItalic           | Optional| no      | Whether to make the text italicized:  
• yes  
• no  |
| fontSize             | Optional| 11      | Font size; integer.  |
| foregroundColor      | Optional| black   | Color of text, grid lines, and labels. Hexadecimal value or supported named color; see name list in Usage. For a hexadecimal value, use the form  
*##xxxxxx* or *##xxxxxxxx*, where x = 0-9 or A-F; use two number signs or none.  |
| format               | Optional| flash   | File format in which to save the graph:  
• flash  
• jpg  
• png  |
| gridlines            | Optional| 10      | Number of grid lines to display on the value axis, including axis; positive integer.  |
| labelFormat          | Optional| number  | Format for y-axis labels:  
• number  
• currency  
• percent  
• date  |
| markerSize           | Optional| (Automatic) | Size of data point marker in pixels; integer.  |
| name                 | Optional|         | Page variable name; string. Generates the graph as binary data and assigns it to the specified variable. Suppresses chart display. You can use the name value in the cffile tag to write the chart to a file.  |
| pieSliceStyle        | Optional| sliced  | Applies to the cfformvalue attribute value pie.  
• solid: displays pie as if unsliced.  
• sliced: displays pie as if sliced.  |
| rotated              | Optional| no      | Whether to rotate the chart 90 degrees:  
• yes  
• no  |
<p>| scaleFrom            | Optional|         | Y-axis minimum value; integer.  |
| scaleTo              | Optional|         | Y-axis maximum value; integer.  |</p>
<table>
<thead>
<tr>
<th>Attribute</th>
<th>Req/Opt</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>seriesPlacement</td>
<td>Optional</td>
<td>default</td>
<td>Relative positions of series in charts that have more than one data series.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• default: ColdFusion determines relative positions, based on graph types.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• cluster</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• stacked</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• percent</td>
</tr>
<tr>
<td>show3D</td>
<td>Optional</td>
<td>yes</td>
<td>Whether to display the chart with three-dimensional appearance:</td>
</tr>
<tr>
<td>showBorder</td>
<td>Optional</td>
<td>no</td>
<td>Whether to display a border around the chart:</td>
</tr>
<tr>
<td>showLegend</td>
<td>Optional</td>
<td>yes</td>
<td>Whether to display the legend if the chart contains more than one data series:</td>
</tr>
<tr>
<td>showMarkers</td>
<td>Optional</td>
<td>yes</td>
<td>Whether to display markers at data points in line, curve, and scatter graphs:</td>
</tr>
<tr>
<td>showXGridlines</td>
<td>Optional</td>
<td>no</td>
<td>Whether to display x-axis gridlines:</td>
</tr>
<tr>
<td>showYGridlines</td>
<td>Optional</td>
<td>yes</td>
<td>Whether to display y-axis gridlines:</td>
</tr>
<tr>
<td>sortXAxis</td>
<td>Optional</td>
<td>no</td>
<td>Whether to display column labels in alphabetic order along the x-axis:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Ignored if the xAxisType attribute is scale.</td>
</tr>
<tr>
<td>style</td>
<td>Optional</td>
<td></td>
<td>XML file or string to use to specify the style of the chart.</td>
</tr>
<tr>
<td>title</td>
<td>Optional</td>
<td></td>
<td>Title of the chart.</td>
</tr>
<tr>
<td>tipbgcolor</td>
<td>Optional</td>
<td>white</td>
<td>Background color of tips. Applies only to Flash format graph files.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Hexadecimal value or supported named color; see the name list in the Usage section.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>For a hexadecimal value, use the form &quot;##xxxxxx&quot;, where x = 0-9 or A-F; use two number signs or none.</td>
</tr>
<tr>
<td>Attribute</td>
<td>Req/Opt</td>
<td>Default</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>---------</td>
<td>-----------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>tipStyle</td>
<td>Optional</td>
<td>mouseOver</td>
<td>Determines the action that opens a pop-up window to display information about the current chart element.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• mouseDown: display if the user positions the cursor at the element and clicks the mouse. Applies only to Flash format graph files. (For other formats, this option functions the same as mouseOver.)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• mouseOver: displays if the user positions the cursor at the element</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• none: suppresses display</td>
</tr>
<tr>
<td>url</td>
<td>Optional</td>
<td></td>
<td>URL to open if the user clicks item in a data series; the onClick destination page. You can specify variables within the URL string: ColdFusion passes current values of the variables.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• $VALUE$: the value of the selected row. If none, the value is an empty string.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• $ITEMLABEL$: the label of the selected item. If none, the value is an empty string.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• $SERIESLABEL$: the label of the selected series. If none, the value is an empty string.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>For example:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• &quot;javascript:...&quot;: executes a client-side script</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• &quot;somepage.cfm?item=$ITEMLABEL$&amp;series=$SERIESLABEL$&amp;value=$VALUE$&quot;</td>
</tr>
<tr>
<td>xAxisTitle</td>
<td>Optional</td>
<td></td>
<td>Title that appears on the x-axis; text.</td>
</tr>
<tr>
<td>xAxisType</td>
<td>Optional</td>
<td>category</td>
<td>Whether the x-axis indicates data or is numeric:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• category: The axis indicates the data category. Data is sorted according to the sortXAxis attribute.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• scale: The axis is numeric. All cfchartdata item attribute values must be numeric. The x-axis is automatically sorted numerically.</td>
</tr>
<tr>
<td>xOffset</td>
<td>Optional</td>
<td>0.1</td>
<td>Number of units by which to display the chart as angled, horizontally. Applies if show3D=&quot;yes&quot;. The number can be between -1 and 1, where -1 specifies 90 degrees left and 1 specifies 90 degrees right.</td>
</tr>
<tr>
<td>yAxisTitle</td>
<td>Optional</td>
<td></td>
<td>Title of the y-axis; text.</td>
</tr>
</tbody>
</table>
The `cfchart` tag defines a container in which a graph displays: its height, width, background color, labels, and so on. The `cfchartseries` tag defines the chart style in which data displays: bar, line, pie, and so on. The `cfchartdata` tag defines a data point.

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

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

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

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

The following table lists W3C HTML 4 named color value or hexadecimal values that the `color` attribute accepts:

<table>
<thead>
<tr>
<th>Color name</th>
<th>RGB value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aqua</td>
<td>#00FFFF</td>
</tr>
<tr>
<td>Black</td>
<td>#000000</td>
</tr>
<tr>
<td>Blue</td>
<td>#0000FF</td>
</tr>
<tr>
<td>Fuchsia</td>
<td>#FF00FF</td>
</tr>
<tr>
<td>Gray</td>
<td>#808080</td>
</tr>
<tr>
<td>Green</td>
<td>#008000</td>
</tr>
<tr>
<td>Lime</td>
<td>#00FF00</td>
</tr>
<tr>
<td>Maroon</td>
<td>#800000</td>
</tr>
<tr>
<td>Navy</td>
<td>#000080</td>
</tr>
<tr>
<td>Olive</td>
<td>#808000</td>
</tr>
<tr>
<td>Purple</td>
<td>#800080</td>
</tr>
<tr>
<td>Red</td>
<td>#FF0000</td>
</tr>
<tr>
<td>Silver</td>
<td>#C0C0C0</td>
</tr>
</tbody>
</table>
For all other color values, you must enter the hexadecimal value. You can enter a six-digit value, which specifies the RGB value, or an eight-digit value, which specifies the RGB value and the transparency. The first two digits of an eight-digit hexadecimal value specify the degree of transparency, with FF indicating opaque and 00 indicating transparent. Values between 00 and FF are allowed.

For more color names that are supported by popular browsers, go to [www.w3.org/TR/css3-color](http://www.w3.org/TR/css3-color)

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

### Example

<!--- The following example analyzes the salary data in the cfdocexamples database and generates a bar chart showing average salary by department. The body of the cfchartseries tag includes one cfchartdata tag to include data that is not available from the query. --->

<!--- Get the raw data from the database. --->
<cfquery name="GetSalaries" datasource="cfdocexamples">
  SELECT Departmt.Dept_Name, Employee.Dept_ID, Employee.Salary
  FROM Departmt, Employee
  WHERE Departmt.Dept_ID = Employee.Dept_ID
</cfquery>

<!--- Use a query of queries to generate a new query with --->
<!--- statistical data for each department. --->
<!--- AVG and SUM calculate statistics. --->
<!--- GROUP BY generates results for each department. --->
<cfquery dbtype = "query" name = "DataTable">
  SELECT Dept_Name, AVG(Salary) AS avgSal, SUM(Salary) AS sumSal
  FROM GetSalaries
  GROUP BY Dept_Name
</cfquery>

<!--- Reformat the generated numbers to show only thousands. --->
<cfloop index = "i" from = "1" to = 
  #DataTable.RecordCount#">
  <cfset DataTable.sumSal[i] = Round(DataTable.sumSal[i]/1000)*1000>
  <cfset DataTable.avgSal[i] = Round(DataTable.avgSal[i]/1000)*1000>
</cfloop>

<table>
<thead>
<tr>
<th>Color name</th>
<th>RGB value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Teal</td>
<td>##008080</td>
</tr>
<tr>
<td>White</td>
<td>##FFFFFF</td>
</tr>
<tr>
<td>Yellow</td>
<td>##FFFF00</td>
</tr>
</tbody>
</table>
Employee Salary Analysis

Bar graph, from Query of Queries

<cfchart format="flash"
xaxistitle="Department"
yaxistitle="Salary Average">
  <cfchartseries type="bar"
query="DataTable"
itemcolumn="Dept_Name"
valuecolumn="avgSal">
    <cfchartdata item="Facilities" value="35000">
  </cfchartseries>
</cfchart>
**cfchartdata**

**Description**

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

**Category**

Data output tags, Extensibility tags

**Syntax**

```xml
<cfchartdata
  item = "text"
  value = "number">
</cfchartdata>
```

**See also**

`cfchart, cfchartseries`; Chapter 31, “Creating Charts and Graphs,” in *ColdFusion MX Developer’s Guide*

ColdFusion MX: Added this tag.

**Attributes**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Req/Opt</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>item</td>
<td>Required</td>
<td></td>
<td>Data point name; string.</td>
</tr>
<tr>
<td>value</td>
<td>Required</td>
<td></td>
<td>Data point value; number or expression.</td>
</tr>
</tbody>
</table>

**Example**

```xml
<!--- The following example analyzes the salary data in the cfdocexamples database and generates a bar chart showing average salary by department. The body of the cfchartseries tag loops over a cfchartdata tag to include data available from the query. --->

<!--- Get the raw data from the database. --->
<cfquery name="GetSalaries" datasource="cfdocexamples">
SELECT Departmt.Dept_Name,
Employee.Dept_ID,
Employee.Salary
FROM Departmt, Employee
WHERE Departmt.Dept_ID = Employee.Dept_ID
</cfquery>

<!--- Use a query of queries to generate a new query with --->
<cfquery dbtype = "query" name = "DataTable">
SELECT
Dept_Name,
AVG(Salary) AS avgSal,
SUM(Salary) AS sumSal
FROM GetSalaries
GROUP BY Dept_Name
</cfquery>
```

74 Chapter 2: ColdFusion Tags
</cfquery>

<!--- Reformat the generated numbers to show only thousands. --->
<cfloop index="i" from="1" to="#DataTable.RecordCount#">
<cfset DataTable.sumSal[i] = Round(DataTable.sumSal[i]/1000)*1000>
<cfset DataTable.avgSal[i] = Round(DataTable.avgSal[i]/1000)*1000>
</cfloop>

<h1>Employee Salary Analysis</h1>

<!--- Bar graph, from Query of Queries. --->
<cfchart format="flash"
  xaxistitle="Department"
yaxistitle="Salary Average">
<cfchartseries type="bar"
  itemcolumn="Dept_Name"
  valuecolumn="avgSal">
<cfloop query="DataTable">
<cfchartdata item="#DataTable.Dept_Name#" value="#DataTable.avgSal#"/>
</cfloop>
</cfchartseries>
</cfchart>
cfchartseries

Description

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

Category

Data output tags, Extensibility tags

Syntax

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

See also

cfchart, cfchartdata; Chapter 31, “Creating Charts and Graphs,” in ColdFusion MX Developer’s Guide

History

ColdFusion MX 7: Added the dataLabelStyle attribute and the horizontalbar chart type.

ColdFusion MX 6.1: Changed interpolation behavior: the tag now interpolates data points on line charts with multiple series.

ColdFusion MX: Added this tag.
### Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Req/Opt</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>colorlist</td>
<td>Optional</td>
<td></td>
<td>Sets colors for each data point. Applies if the <code>cfchartseries</code> type attribute is <code>pie</code>, <code>pyramid</code>, <code>area</code>, <code>horizontalbar</code>, <code>cone</code>, <code>cylinder</code>, or <code>step</code>. Comma-delimited list of hexadecimal values or supported, named web colors; see the name list and information about six- and eight-digit hexadecimal values in the <code>cfchart</code> Usage section. For a hexadecimal value, use the form <code>&quot;##xxxxxx&quot;</code> or <code>&quot;##xxxxxxxx&quot;</code>, where x = 0-9 or A-F; use two number signs or none.</td>
</tr>
<tr>
<td>itemColumn</td>
<td>Required if <code>query</code> attribute is specified</td>
<td></td>
<td>Name of a column in the query specified in the <code>query</code> attribute; contains the item label for a data point to graph.</td>
</tr>
</tbody>
</table>
| markerStyle | Optional | rectangle | Sets the icon that marks a data point for two-dimensional line, curve, and scatter graphs:  
  - rectangle  
  - triangle  
  - diamond  
  - circle  
  - letter  
  - mcross  
  - snow  
  - rcross |
| paintStyle  | Optional | plain    | Sets the paint display style of the data series:  
  - plain: solid color.  
  - raise: the appearance of a button.  
  - shade: gradient fill, darker at the edges.  
  - light: a lighter shade of color; gradient fill. |
| query       | Optional |         | Name of the ColdFusion query from which to get data to graph. |
| seriesColor | Optional |         | Color of the main element (such as the bars) of a chart. For a pie chart, the color of the first slice. Hexadecimal value or supported named color; see the name list and information about six- and eight-digit hexadecimal values in the Usage section for the `cfchart` tag. For a hexadecimal value, use the form `"##xxxxxx"` or `"##xxxxxxxx"`, where x = 0-9 or A-F; use two number signs or none. |
| seriesLabel | Optional |         | Text of the data series label |
Usage

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

- If the `seriesColor` attribute is omitted, ColdFusion automatically determines the colors of the slices.
- If the `seriesColor` attribute is specified, ColdFusion automatically determines the colors of the slices after the first one, starting with the specified color for the first slice.

Example

```cfml
<!--- The following example analyzes the salary data in the cfdocexamples database and generates a bar chart showing average salary by department. --->

<!--- Get the raw data from the database. --->
<cfquery name="GetSalaries" datasource="cfdocexamples">
SELECT Departmt.Dept_Name,
      Employee.Dept_ID,
      Employee.Salary
FROM Departmt, Employee
WHERE Departmt.Dept_ID = Employee.Dept_ID
</cfquery>
```
<!--- Use a query of queries to generate a new query with --->
<!--- statistical data for each department. --->
<!--- AVG and SUM calculate statistics. --->
<!--- GROUP BY generates results for each department. --->
<cfquery dbtype = "query" name = "DataTable">
SELECT
  Dept_Name,
  AVG(Salary) AS avgSal,
  SUM(Salary) AS sumSal
FROM GetSalaries
GROUP BY Dept_Name
</cfquery>

<!--- Reformat the generated numbers to show only thousands. --->
<cfloop index = "i" from = "1" to = "#DataTable.RecordCount#">
  <cfset DataTable.sumSal[i] = Round(DataTable.sumSal[i]/1000)*1000>
  <cfset DataTable.avgSal[i] = Round(DataTable.avgSal[i]/1000)*1000>
</cfloop>

<h1>Employee Salary Analysis</h1>
<!--- Bar graph, from Query of Queries --->
<cfchart format="flash"
xaxistitle="Department"
yaxistitle="Salary Average">
  <cfchartseries type="bar"
query="DataTable"
itemcolumn="Dept_Name"
valuecolumn="avgSal" />
</cfchart>
cfcoll

Description

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

Category

Data output tags

Syntax

```
<cfcol
    header = "column_header_text"
    width = "number_indicating_width_of_column"
    align = "left" or "right" or "center"
    text = "column_text">
```

See also

cfcontent, cfoutput, cftable; "Performing file operations with cfftp" in Chapter 40, "Interacting with Remote Servers," in ColdFusion MX Developer's Guide

History

ColdFusion MX: Added the ability to construct dynamic cfcoll statements.

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

Usage

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

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

<!--- This example shows the use of cfcol and cftable to align information returned from a query. --->
<!--- query selects information from cfdocexamples data source. --->
<cfquery name = "GetEmployees" dataSource = "cfdocexamples">
  SELECT Emp_ID, FirstName, LastName, Email, Phone, Department FROM Employees
</cfquery>
<html>
<body>
<h3>cfcol Example</h3>
<!--- Uses the HTMLTable attribute to display cftable as an HTML table, rather than PRE formatted information --->
<cftable
  query = "GetEmployees"
  startRow = "1" colSpacing = "3"
  HTMLTable colheaders>
  <!--- Each cfcol tag sets the width of a column in the table, the header information, and the text/CFML for the cell. --->
  <cfcol header = "<b>ID</b>"
    align = "Left"
    width = 2
    text = "#Emp_ID#">
  <cfcol header = "<b>Name/Email</b>"
    align = "Left"
    width = 15
    text = "<a href = 'mailto:#Email#'>#FirstName# #LastName#</a>">
  <cfcol header = "<b>Phone Number</b>"
    align = "Center"
    width = 15
    text = "#Phone#">
</cftable>
cfcollection

Description

Creates and administers Verity search engine collections.

Category

Extensibility tags

Syntax

```xml
<cfcollection
  action = "action"
  collection = "collection_name"
  path = "path_to_verity_collection"
  language = "language"
  name = "queryname"
  categories = "yes" or "no">
</cfcollection>
```

See also

cfexecute, cfindex, cfobject, cfreport, cfsearch, cfwddx

History

ColdFusion MX 7:

- Removed reference to external collections.
- Deprecated the map and repair options of the action attribute. They might not work, and might cause an error, in later releases.
- Added categories attribute and categorylist action.
- Added CATEGORIES, SIZE, DOCCOUNT, and LASTMODIFIED to list of variables returned by the list action.
- Marked as obsolete the MAPPED, ONLINE, and REGISTERED variables returned by the list action.

ColdFusion MX:

- Changed the requirements for the action attribute: it is now required.
- Added the action attribute list value. It is the default.
- Changed the requirements for the action attribute value map: it is not necessary to specify the action attribute value map. (ColdFusion detects collections and creates maps collections as required.)
- Changed acceptable collection naming: this tag accepts collection names that include spaces.
- Changed Verity operations behavior: ColdFusion supports Verity operations on Acrobat PDF files.
- Changed thrown exceptions: this tag can throw the SEARCHENGINE exception.
Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Req/Opt</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>action</td>
<td>Required; see Usage</td>
<td>list</td>
<td>• categorylist: retrieves categories from the collection and indicates how many documents are in each one. Returns a structure of structures in which the category representing each substructure is associated with a number of documents. For a category in a category tree, the number of documents is the number at or below that level in the tree. • create: registers the collection with ColdFusion. ▪ If the collection is present: creates a map to it. ▪ If the collection is not present: creates it. • delete: unregisters a collection and deletes its directories. • list: returns a query result set, named from the name attribute value, of the attributes of the collections that are registered by ColdFusion. • map: creates a map to a collection. If the action is create and the collection already exists, ColdFusion also creates a map to the collection. • optimize: optimizes the structure and contents of the collection for searching; recovers space. Causes collection to be taken offline, preventing searches and indexing. • repair: deprecated. Does nothing.</td>
</tr>
<tr>
<td>collection</td>
<td>See Usage</td>
<td>A collection name. The name can include spaces.</td>
<td></td>
</tr>
<tr>
<td>path</td>
<td>See Usage</td>
<td>Absolute path to a Verity collection. To map an existing collection, specify a fully qualified path to the collection (not including the collection name); for example, &quot;C:\MyCollections&quot;.</td>
<td></td>
</tr>
<tr>
<td>language</td>
<td>See Usage</td>
<td>English Although English is the default language, Englishx, a more advanced English locale, is also provided. For a list of options, see Usage. Requires the appropriate (European or Asian) Verity Locales language pack.</td>
<td></td>
</tr>
<tr>
<td>name</td>
<td>See Usage</td>
<td>Name for the query results returned by the list and categorylist actions.</td>
<td></td>
</tr>
<tr>
<td>categories</td>
<td>See Usage</td>
<td>no Used only for creating a collection: ▪ yes: this collection includes support for categories. ▪ no: this collection does not support categories.</td>
<td></td>
</tr>
</tbody>
</table>

Usage

With this tag you can create, register, and administer a Verity collection that was created by ColdFusion or by a Verity application.
The following table shows the dependence relationships among this tag's attribute values:

<table>
<thead>
<tr>
<th>This attribute is required, optional, or unnecessary (blank):</th>
<th>For this action attribute value:</th>
</tr>
</thead>
<tbody>
<tr>
<td>collection</td>
<td>list   create map optimize repair delete categorylist</td>
</tr>
<tr>
<td>path</td>
<td>Required Required Required Required Required Required</td>
</tr>
<tr>
<td>language</td>
<td>Optional Optional</td>
</tr>
<tr>
<td>name</td>
<td>Required</td>
</tr>
<tr>
<td>categories</td>
<td>Required</td>
</tr>
</tbody>
</table>

The following examples illustrate the structures returned by the `categorylist` action:

**CATEGORIES**

- blue: 10
- green: 3
- magenta: 3
- purple: 2

**CATEGORYTREES**

- a/: 10
- a/b: 10
- a/b/c: 10
- a/b/c/subdir: 3

The `list` action returns the following information in a result set that contains one row per collection:

<table>
<thead>
<tr>
<th>Column</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>CATEGORIES</td>
<td>• yes: the collection has category support enabled.</td>
</tr>
<tr>
<td></td>
<td>• no: the collection does not have category support enabled.</td>
</tr>
<tr>
<td>CHARSET</td>
<td>The character set of the collection.</td>
</tr>
<tr>
<td>CREATED</td>
<td>The date and time that the collection was created.</td>
</tr>
<tr>
<td>DOCCOUNT</td>
<td>The number of documents in this collection.</td>
</tr>
<tr>
<td>EXTERNAL</td>
<td>• yes: the collection is external.</td>
</tr>
<tr>
<td></td>
<td>• no: the collection is not external.</td>
</tr>
<tr>
<td></td>
<td>• Not Found: the collection is registered but is not available in the defined path.</td>
</tr>
<tr>
<td>LANGUAGE</td>
<td>The locale setting of the collection.</td>
</tr>
<tr>
<td></td>
<td>This information is not available for K2Server collections.</td>
</tr>
</tbody>
</table>
The ColdFusion MX Administrator Verity > Collections page displays the information that is returned when you use the `list` attribute.

If the Verity Server is not running when the `list` action is executed, the tag throws an error.

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

```coldfusion
<cfcollection action="list" name="myCollections">
<cfquery name="qoq" dbtype="query">
    select * from myCollections
    where myCollections.name = 'myCollectionName'
</cfquery>
<cfif qoq.recordcount GT 0>
    <!--- Collection exists --->
    <cfdump var="#qoq#">
</cfif>
```

To get a result set with values for all the collections that are registered with the Verity server, use code such as the following:

```coldfusion
<cfcollection action="list" name="myCollections">
<cfoutput query="myCollections">
    #name#
</cfoutput>
</cfcollection>
```

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

The `language` attribute of this tag supports the following options:

<table>
<thead>
<tr>
<th>Asian Language Pack</th>
<th>Chinese</th>
<th>Traditional Chinese</th>
</tr>
</thead>
<tbody>
<tr>
<td>Japanese</td>
<td>Korean</td>
<td></td>
</tr>
<tr>
<td>Multilanguage Language Pack</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Unicode</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Western European Language Pack</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bokmal</td>
</tr>
<tr>
<td>Danish</td>
</tr>
<tr>
<td>Dutch</td>
</tr>
</tbody>
</table>

---

**Column** | **Contents**
---|---
LASTMODIFIED | The date and time that the collection was last changed.
MAPPED | Obsolete
NAME | The name of the collection.
ONLINE | Obsolete
PATH | Absolute path to the collection.
REGISTERED | Obsolete
SIZE | The size of the collection, expressed in kilobytes.
The default location of Verity collections is as follows:

- Server configuration:
  - Windows: C:\CFusionMX7\verity\collections
  - UNIX system: /opt/coldfusionmx7/verity/collections

- J2EE configuration: webapp_root/WEB-INF/cfusion/verity/collections

Example

```coldfusion
<!-------------------------------------------------------------------------
(call_actn.cfm)
Check for server platform and use its default Verity Collection directory.
If you did not install ColdFusion MX in the default directory, or if you use
the J2EE configuration, or if your webroot is not C:\CFusionMX7\wwwroot, you
might need to change the path in this example. For example, for JRun4 the path
might be C:\JRun4\Verity\Collections\
------------------------------------------------------------------------->
<cfif Find("Windows", Server.OS.Name)>
  <cfset collPath = "C:\JRun4\Verity\Collections\">
<cfelse>
  <cfset collpath = "/opt/coldfusionmx7/verity/collections/">
</cfif>

<!-------------------------------------------------------------------------
Process form input and do the requested cfcollection operation.
------------------------------------------------------------------------->
<cfif IsDefined("form.CollectionName") AND IsDefined("form.CollectionAction")>
  <cfif form.CollectionName is not ">
    <cfswitch expression="#FORM.CollectionAction#">
      <cfcase value="Create">
        <cfcollection action="CREATE" collection="#FORM.CollectionName#"
        path="#collPath#" categories="yes">
        <h3>Collection created.<br>
        Use CFINDEX to populate it.</h3>
        </cfcollection>
      </cfcase>
      <cfcase value="Repair">
        <cfcollection action="REPAIR" collection="#FORM.CollectionName#">
        <h3>Collection repaired.</h3>
        </cfcollection>
      </cfcase>
      <cfcase value="Optimize">
        <cfcollection action="OPTIMIZE" collection="#FORM.CollectionName#">
        <h3>Collection optimized.</h3>
        </cfcollection>
      </cfcase>
      <cfcase value="Delete">
        <cfcollection action="DELETE" collection="#FORM.CollectionName#">
        <h3>Collection deleted.</h3>
        </cfcollection>
    </cfswitch>
  </cfif>
</cfif>
```

Eastern European/Middle Eastern Language Pack

<table>
<thead>
<tr>
<th>Arabic</th>
<th>Greek</th>
<th>Polish</th>
<th>Turkish</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bulgarian</td>
<td>Hebrew</td>
<td>Russian</td>
<td></td>
</tr>
<tr>
<td>Czech</td>
<td>Hungarian</td>
<td>Russian2</td>
<td></td>
</tr>
</tbody>
</table>

86 Chapter 2: ColdFusion Tags
<cfswitch>
  <cfcase> </cfcase>
  </cfswitch>
  <cfelse>
    <h3>Please enter a name for your collection</h3>
  </cfif>
  </cfif>

<!-------------------------------------------------------------------
(coll_form.cfm)
Form to specify the collection name and action
  coll_form.cfm
-------------------------------------------------------------------->

<form action="coll_actn.cfm" method="POST">
  <select name="CollectionAction">
    <option value="Create">Create this collection</option>
    <option value="Optimize">Optimize this collection</option>
    <option value="Repair">Repair this collection</option>
    <option value="Delete">Delete this collection</option>
  </select>

  <p><strong>Collection on which to act</strong><br>
    Use the default value or enter your own Collection name<br>
    <input type="Text" name="CollectionName" value="My_coll"></p>

  <input type="Submit" name="" value="alter or create my collection">
</form>
**cfcomponent**

**Description**

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

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

A component method is invoked in the following ways:

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

**Category**

*Extensibility tags*

**Syntax**

```cfcomponent
  extends = "anotherComponent"
  output = "yes" or "no"
  style = "rpc" or "document"
  namespace = "default service namespace"
  serviceportname = "port element name"
  porttypename = "porttype element name"
  bindingname = "binding element name"
  wsdlfile = "path to hard-coded wsdl file"
  displayname = "text string"
  hint = "text string"
  variable declarations
  <cffunction ...
  ...
  </cffunction>
  ...
  <cffunction ...
  ...
  </cffunction>
  </cfcomponent>
```

**See also**

History

ColdFusion MX 7:

• Added support for publishing document-literal style web services.
• Added the style, namespace, serviceportname, porttypename, wsdlfile, and bindingname attributes.
• Extended functionality for the hint and displayname attributes when publishing document-literal style web services.

ColdFusion MX: Added this tag.

Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Req/Opt</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>extends</td>
<td>Optional</td>
<td></td>
<td>Name of parent component from which to inherit methods and properties.</td>
</tr>
<tr>
<td>output</td>
<td>Optional</td>
<td></td>
<td>Specifies whether constructor code in the component can generate HTML output; does not affect output in the body of cffunction tags in the component.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Component body displayable text is processed as standard CFML.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>yes: Constructor code is processed as if it were within a cfoutput tag. Variable names surrounded by number signs (#) are automatically replaced with their values.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>no: Constructor code is processed as if it were within a cfsilent tag.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>If you do not specify this attribute, constructor code is processed as standard CFML. Any variables must be in cfoutput tags.</td>
<td></td>
</tr>
<tr>
<td>style</td>
<td>Optional</td>
<td>rpc</td>
<td>Specifies whether a CFC used for web services uses RPC-encoded style or document-literal style:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• rpc: RPC-encoded style</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• document: document-literal style</td>
</tr>
<tr>
<td>namespace</td>
<td>Optional</td>
<td>classname</td>
<td>Specifies the namespace used in the WSDL when using the CFC as a document-literal style web service. If you don’t specify this attribute, ColdFusion MX derives the value from the CFC class name.</td>
</tr>
<tr>
<td>serviceportname</td>
<td>Optional</td>
<td></td>
<td>Specifies the name attribute of the port element in the WSDL. If you don’t specify this attribute, ColdFusion MX derives the value from the CFC class name.</td>
</tr>
</tbody>
</table>

This attribute applies only when style="document".
<cfinvoke component="managerCFC" method="getEmployeeName"
  returnVariable="managerName" EmployeeID=#EmpID#>
</cfinvoke>

This tag requires an end tag.

If you specify style="document", ColdFusion MX publishes the CFC as a document-literal style web service. For more information, see "Publishing document-literal style web services" in Chapter 36, "Using Web Services" in ColdFusion MX Developer's Guide.
Example

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

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

**Description**

Does either or both of the following:

- Sets the MIME content encoding header for the current page; if the encoding information includes a character encoding, sets the character encoding of generated output.
- Sends the contents of a file, or of a variable that contains binary data, as the page output.

To restrict this tag, use the Sandbox Security feature of the ColdFusion MX Administrator. For more information, see the Administrator online Help.

**Category**

Data output tags

**Syntax**

```plaintext
<cfcontent
    type = "file_type"
    deleteFile = "yes" or "no"
    file = "filename"
    variable = "variablename"
    reset = "yes" or "no">
```

**See also**

`cfcol, cfheader, cfhttp, cfoutput, cftable`

**History**

ColdFusion MX 7: Added the `variable` attribute.
### Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Req/Opt</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
</table>
| **type**  | Optional|         | The MIME content type of the page, optionally followed by a semicolon and the character encoding. By default, ColdFusion sends pages as text/html content type in the UTF-8 character encoding. The content type determines how the browser or client interprets the page contents. The following are some of the content type values you can use:  
• text/html  
• text/plain  
• application/x-shockwave-flash  
• application/msword  
• image/jpeg  
The following list includes commonly used character encoding values:  
• utf-8  
• iso-8859-1  
• windows-1252  
• us-ascii  
• shift_jis  
• iso-2022-jp  
• euc-jp  
• euc-kr  
• big5  
• euc-cn  
• utf-16  
For example:  
type = "text/html"  
type = "text/html; charset=ISO-8859-1" |
| **deleteFile** | Optional | no | Applies only if you specify a file with the file attribute.  
• yes: deletes the file on the server after sending its contents to the client.  
• no: leaves the file on the server. |
| **file** | Optional | | Name of file whose contents will be the page output. The file name must start with a drive letter and a colon, or a forward or backward slash. When using ColdFusion in a distributed configuration, the file attribute must refer to a path on the system on which the web server runs. When you use this attribute, any other output on the current CFML page is ignored; only the contents of the file are sent to the client. |
Usage

To set the character encoding (character set) of generated output, including the page HTML, use code such as the following:

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

When ColdFusion processes an HTTP request, it determines the character encoding to use for the data it returns in the HTTP response. By default, ColdFusion returns character data using the Unicode UTF-8 format, regardless of the value of an HTML meta tag in the page. You can use the `cfcontent` tag to override the default character encoding of the response. For example, to tell ColdFusion MX to return the page using Japanese EUC character encoding, use the `type` attribute, as follows:

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

If you call the `cfcontent` tag from a custom tag, and you do not want the tag to discard the current page when it is called from another application or custom tag, set `reset = "no"`. If a file delete operation is unsuccessful, ColdFusion throws an error.

Do not use this tag after the `cfflush` tag on a page, it will have no effect or ColdFusion will throw an error.

The following tag can force most browsers to display a dialog box that asks users whether they want to save the contents of the file specified by the `cfcontent` tag using the filename specified by the `filename` value. If the user selects to open the file, most browsers open the file in the related application, not the browser window.

```cfheader name="Content-Disposition" value="attachment; filename=filename.ext">
```

Some file types, such as PDF documents, do not use executable code and can display directly in most browsers. To request the browser to display the file directly, use a `cfheader` tag similar to the following:

```cfheader name="Content-Disposition" value="inline; filename=filename.ext">
```

You can use any value for the `filename` part of the `filename` attribute, but the `ext` part must be the standard Windows extension for the file type.
For file types that might contain executable code, such as Microsoft Excel documents, most browsers always ask before opening the document. For these file types, the inline content disposition specification requests the browser to display the file directly if the user selects to open the file.

For more information on character encodings, see the following web pages:

• [www.w3.org/International/O-charset.html](http://www.w3.org/International/O-charset.html) provides general information on character encodings and the web, and has several useful links.

• [www.iana.org/assignments/character-sets](http://www.iana.org/assignments/character-sets) is a complete list of character sets names used on the Internet, maintained by the Internet Assigned Numbers Authority.

• [java.sun.com/j2se/1.3/docs/guide/intl/encoding.doc.html](http://java.sun.com/j2se/1.3/docs/guide/intl/encoding.doc.html) lists the character encodings that Java, and therefore ColdFusion, can interpret. This list uses Java internal names, not the IANA character encoding names that you use in the `SetEncoding charset` parameter and other ColdFusion attributes and parameters. ColdFusion MX 6.0 Updater 3 uses JDK 1.3. CFMX 6.1 uses JDK 1.4.2; for encoding support, see [http://java.sun.com/j2se/1.4.2/docs/guide/intl/encoding.doc.html](http://java.sun.com/j2se/1.4.2/docs/guide/intl/encoding.doc.html).

For a complete list of media types used on the Internet, see [www.iana.org/assignments/media-types/](http://www.iana.org/assignments/media-types/).

Example

<!--- CFCONTENT Example 1
This example shows the use of cfcontent to return the contents of the CF Documentation page dynamically to the browser. You might need to change the path and/or drive letter depending on how ColdFusion is installed on your system. Notice that the graphics do not display and the hyperlinks do not work, because the html page uses relative filename references. The root of the reference is the ColdFusion page, not the location of the html page. --->

```<cfcontent type = "text/html"
    file = "C:\CFusionMX7\wwwroot\cfdocs\dochome.htm"
    deleteFile = "no">
```

<!--- CFCONTENT Example 2
This example shows how the Reset attribute changes text output. Notice how the first text section ("This example shows how the Reset attribute changes output for text reset = "Yes":123) does NOT print out to the screen. --->

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

<!--- CFCONTENT Example 3
This example triggers a download of an Excel file. The user will be prompted with an option to save the file or open it in the browser. --->

```<cfheader name="Content-Disposition" value="inline; filename=acmesales03.xls">
<cfcontent type="application/vnd.ms-excel" file="c:\temp\acmesales03.xls">
```
<!--- CFCONTENT EXAMPLE 4
This example triggers a download of a Word document then deletes the original from the "temp" directory. The user will be prompted with an option to save the file or open it in the browser. --->
<cfheader name="Content-Disposition" value="inline; filename=temp.doc">
<cfcontent type="application/msword" file="c:\temp\Cable.doc" deletefile="yes">

<!--- CFCONTENT EXAMPLE 5
This example causes the browser to treat the HTML table as Excel data. Excel interprets the table format. Because Excel can include executable code, the browser prompts the user whether to save the file or open it in a browser. --->
<cfheader name="Content-Disposition" value="inline; filename=acmesalesQ1.xls">
<cfcontent type="application/vnd.msexcel">
<table border="2">
<tr><td>Month</td><td>Quantity</td><td>$ Sales</td></tr>
<tr><td>January</td><td>80</td><td>$245</td></tr>
<tr><td>February</td><td>100</td><td>$699</td></tr>
<tr><td>March</td><td>230</td><td>$2036</td></tr>
<tr><td>Total</td><td>=Sum(B2..B4)</td><td>=Sum(C2..C4)</td></tr>
</table>
**cfcookie**

**Description**

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

**Category**

Forms tags, Variable manipulation tags

**Syntax**

```html
cfcookie
   name = "cookie_name"
   value = "text"
   expires = "period"
   secure = "yes" or "no"
   path = "url"
   domain = ".domain"
</cfcookie>
```

**See also**

cfdump, cfparam, cfregistry, cfsavecontent, cfschedule, cfset

**History**

ColdFusion MX 6.1:

- Changed the `expires` attribute: it now accepts a date time object.
- Cookie names can include all ASCII characters except commas, semicolons, or whitespace characters.

**Attributes**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Req/Opt</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Required</td>
<td></td>
<td>Name of cookie variable. ColdFusion converts cookie names to all-uppercase. Cookie names set using this tag can include any printable ASCII characters except commas, semicolons or white space characters.</td>
</tr>
<tr>
<td>value</td>
<td>Optional</td>
<td></td>
<td>Value to assign to cookie variable. Must be a string or variable that can be stored as a string.</td>
</tr>
<tr>
<td>expires</td>
<td>Optional</td>
<td></td>
<td>Expiration of cookie variable.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• The default: the cookie expires when the user closes the browser, that is, the cookie is &quot;session only&quot;.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• A date or date/time object (for example, 10/09/97)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• A number of days (for example, 10, or 100)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• now: deletes cookie from client cookie.txt file (but does not delete the corresponding variable the Cookie scope of the active page).</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• never: The cookie expires in 30 years from the time it was created (effectively never in web years).</td>
</tr>
</tbody>
</table>
If this tag specifies that a cookie is to be saved beyond the current browser session, the client
browser writes or updates the cookie in its local cookies file. Until the browser is closed, the
cookie resides in browser memory. If the `expires` attribute is not specified, the cookie is not
written to the browser cookies file.

If you use this tag after the `cfflush` tag on a page, ColdFusion does not send the cookie to the
browser; however, the value you set is available to ColdFusion in the Cookie scope during the
browser session.

**Note:** You can also create a cookie that expires when the current browser session expires by using
the `cfset` tag or a CFScript assignment statement to set a variable in the Cookie scope, as in `<cfset
Cookie.mycookie="sugar">`. To get a cookie’s value, refer to the cookie name in the Cookie scope, as
in `<cfif Cookie.mycookie is "oatmeal">`.

You can use dots in cookie names, as the following examples show:

```
<cfcookie name="person.name" value="wilson, john">
<cfset cookie.person.lastname="Santiago">
```

To access cookies, including cookies that you set and all cookies that are sent by the client, use the
Cookie scope. For example, to display the value of the `person.name` cookie set in the preceding
code, use the following line:

```
<cfoutput>#{cookie.person.name}#</cfoutput>
```
Example

<!--- This example shows how to set/delete a cfcookie variable. --->
<!--- Select users who have entered comments into a sample database. --->
<cfquery name = "GetAolUser" dataSource = "cfdocexamples">
  SELECT EMail, FromUser, Subject, Posted
  FROM Comments
</cfquery>
<html>
<body>
<h3>cfcookie Example</h3>
<!--- If the URL variable delcookie exists, set cookie expiration date to NOW --->
<cfif IsDefined("url.delcookie") is True>
  <cfcookie name = "TimeVisited"
    value = "#Now()#"
    expires = "NOW">  
<cfelse>
<!--- Otherwise, loop through list of visitors; stop when you match the string aol.com in a visitor's e-mail address. --->
<cfloop query = "GetAolUser">
  <cfif FindNoCase("aol.com", Email, 1) is not 0>
    <cfcookie name = "LastAOLVisitor"
      value = "#Email#
      expires = "NOW" >
  </cfif>
</cfloop>
<!--- If the timeVisited cookie is not set, set a value. --->
<cfif IsDefined("Cookie.TimeVisited") is False>
  <cfcookie name = "TimeVisited"
    value = "#Now()#"
    expires = "10">  
</cfif>
</cfif>
<!--- Show the most recent cookie set. --->
<cfif IsDefined("Cookie.LastAOLVisitor") is "True">
  <p>The last AOL visitor to view this site was <cfoutput>"#Cookie.LastAOLVisitor"</cfoutput>, on <cfoutput>"#DateFormate(COOKIE.TimeVisited)"</cfoutput>
</cfif>
<!--- Use this link to reset the cookies. --->
<p><a href = "cfcookie.cfm?delcookie = yes">Hide my tracks</a>
<cfelse>
<p>No AOL Visitors have viewed the site lately.
</cfif>
**cfdefaultcase**

**Description**

Used only inside the `cfswitch` tag body. Contains code to execute when the expression specified in the `cfswitch` tag does not match a of the value specified by a `cfcase` tag.

**Category**

Flow-control tags

**Syntax**

```xml
<cfdefaultcase>
```

**See also**


**History**

ColdFusion MX: Changed placement requirements: this tag does not have to follow all `cfcase` tags in the `cfswitch` tag body.

**Usage**

The contents of the `cfdefaultcase` tag body is executes if the expression attribute of the `cfswitch` tag does not match any of the values specified by the `cfcase` tags in the `cfswitch` tag body. The contents of the `cfdefaultcase` tag body can include HTML and text, and CFML tags, functions, variables, and expressions.

You can specify only one `cfdefaultcase` tag within a `cfswitch` tag. You can put the `cfdefaultcase` tag at any position within a `cfswitch` statement; it is not required to be the last item, but it is good programming practice to put it last.

**Example**

<!--- The following example displays a grade based on a 1-10 score. Several of the cfcase tags match more than one score. For simplicity, the example sets the score to 7. --->

```xml
<cfset score="7">
<cfswitch expression="#score#">
  <cfcase value="10">
    <cfset grade="A">
  </cfcase>
  <cfcase value="9;8" delimiters=";">
    <cfset grade="B">
  </cfcase>
  <cfcase value="7;6" delimiters=";">
    <cfset grade="C">
  </cfcase>
  <cfcase value="5;4;" delimiters=";">
    <cfset grade="D">
  </cfcase>
  <cfdefaultcase>
    <cfset grade="F">
  </cfdefaultcase>
</cfswitch>
```

---

100 Chapter 2: ColdFusion Tags
</cfdefaultcase>
</cfswitch>
<cfoutput>
  Your grade is #grade#
</cfoutput>
cfdirectory

Description
Manages interactions with directories.

Category
File management tags

Syntax
```xml
<cfdirectory
    action = "directory action"
    directory = "directory name"
    name = "query name"
    filter = "list filter"
    mode = "permission"
    sort = "sort specification"
    newDirectory = "new directory name"
    recurse = "yes" or "no">
</cfdirectory>
```

See also
cffile

History
ColdFusion MX 7: Added the recurse attribute (named recursive in Alpha 1) and directory result-set column.

ColdFusion MX:
- Changed behavior for action = "list":
  - On Windows, cfdirectory action = "list" no longer returns the directory entries "." (dot) or "..." (dot dot), which represent "the current directory" and "the parent directory."
  - On Windows, cfdirectory action = "list" no longer returns the values of the Archive and System attributes.
  - On UNIX and Linux, cfdirectory action = "list" does not return any information in the mode column.
## Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Req/Opt</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
</table>
| action      | Optional | List    | • list: returns a query record set of the files in the specified directory. The directory entries "." (dot) and ".." (dot dot), which represent the current directory and the parent directory, are not returned.  
• create  
• delete  
• rename |
| directory   | Required | Absolute pathname of directory against which to perform action.  
You can use an IP address, as in the following example:  
```<cfdirectory directory="//12.3.123.123/c_drive/" name="dirQuery" action="LIST">``` |
| name        | Required if action = "list" | Name for output record set. |
| filter      | Optional if action = "list" | File extension filter applied to returned names; for example, ".cfm. One filter can be applied. |
| mode        | Optional | Used with action = "create". Permissions. Applies only to UNIX and Linux. Octal values of chmod command. Assigned to owner, group, and other, respectively. For example:  
• 644: assigns read/write permission to owner; read permission to group and other.  
• 777: assigns read/write/execute permission to all. |
| sort        | Optional; used if action = "list" | ASC | Query column(s) by which to sort directory listing. Delimited list of columns from query output.  
To qualify a column, use:  
• asc: ascending (a to z) sort order.  
• desc: descending (z to a) sort order.  
For example:  
```sort = "directory ASC, size DESC, datelastmodified"``` |
| newDirectory | Required if action = "rename" | New name for directory. |
| recurse     | Optional | no      | Whether ColdFusion performs the action on subdirectories:  
• yes  
• no  
Valid for action="list" and action="delete". |
Usage

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

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

- **name** | Directory entry name. The entries "." and ".." are not returned.
- **directory** | Directory containing the entry.
- **size** | Directory entry size.
- **type** | File type: File, for a file; Dir, for a directory.
- **dateLastModified** | The date that an entry was last modified.
- **attributes** | File attributes, if applicable.
- **mode** | Empty column; retained for backward compatibility with ColdFusion 5 applications on UNIX.

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

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

**Note:** If the `cfdirectory` tag does not appear to work, for example, if a `list` operation returns an empty result set, make sure that you have correct permissions to access the directory. For example, if you run ColdFusion as a service on Windows, it operates by default as System, and cannot access directories on a remote system or mapped drive; to resolve this issue, do not run ColdFusion using the local system account.

The `filter` attribute specifies a pattern of one or more characters. All names that match that pattern are included in the list. On Windows systems, pattern matching ignores text case, on UNIX and Linux, pattern matches are case-sensitive.

The following two characters have special meaning in the pattern and are called metacharacters:

- **matches any zero or more characters**
- **matches any single character**
The following table shows examples of patterns and file names that they match:

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Matches</th>
</tr>
</thead>
<tbody>
<tr>
<td>foo.*</td>
<td>Any file called foo with any extension; for example, foo.html, foo.cfm, and foo.xml.</td>
</tr>
<tr>
<td>*.html</td>
<td>All files with the suffix .html, but not files with the suffix .htm.</td>
</tr>
<tr>
<td>??</td>
<td>All files with two-character names.</td>
</tr>
</tbody>
</table>

Example

<!--- EXAMPLE 1: Creating and Renaming  --->
Check that the directory exists to avoid getting a ColdFusion error message. --->
<cfset newDirectory = "otherNewDir">
<cfset currentDirectory = GetDirectoryFromPath(GetTemplatePath()) & "newDir">
<!--- Check to see if the Directory exists. --->
<cfif DirectoryExists(currentDirectory)>
<!--- If TRUE then rename the directory. --->
<cfdirectory action = "rename" directory = "#currentDirectory#" newDirectory = "#newDirectory#" >
<cfoutput>
<p>The directory existed and the name has been changed to: #newDirectory#</p>
</cfoutput>
<cfelse>
<!--- If FALSE, create the directory. --->
<cfdirectory action = "create" directory = "#currentDirectory#" >
<cfoutput><p>Your directory has been created.</p></cfoutput>
</cfif>

<!--- EXAMPLE 2: Deleting a directory  --->
Check that the directory exists and that files are not in the directory to avoid getting ColdFusion error messages. --->
<cfset currentDirectory = GetDirectoryFromPath(GetTemplatePath()) & "otherNewDir">
<!--- Check to see if the Directory exists. --->
<cfif DirectoryExists(currentDirectory)>
<!--- If TRUE, check to see if there are files in the directory before deleting. --->
<cfdirectory action="list" directory="#currentDirectory#" name="myDirectory">
<cfif myDirectory.recordcount gt 0>
<!--- If TRUE, delete the files from the directory. --->
<cfoutput>
<p>Files exist in this directory. Either delete the files or code something to do so.</p>
</cfoutput>
<cfelse>
<!--- Directory is empty - just delete the directory. --->
<cfdirectory action = "delete" directory = "#currentDirectory#" >
<cfoutput>
<p>The directory existed and has been deleted.</p>
</cfoutput>
</cfif>
</cfdirectory>
<cfif>
  <cfelse>
    <!--- If FALSE, post message or do some other function. --->
    <cfoutput><p>The directory did NOT exist.</p></cfoutput>
  </cfif>
</cfif>

<!---EXAMPLE 3: List directories
The following example creates both an array of directory names and a query that contains entries for the directories only. --->

<cfdirectory directory="C:/temp" name="dirQuery" action="LIST">
  <!--- Get an array of directory names. --->
  <cfset dirsArray=arraynew(1)>
  <cfset i=1>
  <cfloop query="dirQuery">
    <cfif dirQuery.type IS "dir">
      <cfset dirsArray[i]=dirQuery.name>
      <cfset i = i + 1>
    </cfif>
  </cfloop>
  <cfdump var="#dirsArray#">

  <!--- Get all directory information in a query of queries.--->
  <cfquery dbtype="query" name="dirsOnly">
    SELECT * FROM dirQuery
    WHERE TYPE='Dir'
  </cfquery>
  <cfdump var="#dirsOnly#"
cfdocument

Description

Creates PDF or FlashPaper output from a text block containing CFML and HTML.

Category

Data output tags

Syntax

```xml
<cfdocument
    format = "PDF" or "FlashPaper"
    filename = "filename"
    overwrite = "yes" or "no"
    name = "output variable name"
    pagetype = "page type"
    pageheight = "page height in inches"
    pagewidth = "page width in inches"
    orientation = "portrait/landscape"
    margin = "number"
    marginbottom = "number"
    marginleft = "number"
    marginright = "number"
    unit = "in" or "cm"
    encryption = "128-bit" or "40-bit" or "none"
    ownerpassword = "password"
    userpassword = "password"
    permissions = "permission list"
    fontembed = "yes" or "no"
    backgroundvisible = "yes" or "no"
    scale = "percentage less than 100">
    HTML and CFML code
</cfdocument>
```

See also

cfreport, cfdocumentitem, cfdocumentsection

History

ColdFusion MX 7: Added this tag.

Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Req/Opt</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>format</td>
<td>Required</td>
<td></td>
<td>Specifies the report format:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• PDF</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• FlashPaper</td>
</tr>
<tr>
<td>filename</td>
<td>Optional</td>
<td></td>
<td>Specifies the name of a file to contain the PDF or FlashPaper output.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>If you omit the filename attribute, ColdFusion MX streams output to the browser.</td>
</tr>
<tr>
<td>Attribute</td>
<td>Req/Opt</td>
<td>Default</td>
<td>Description</td>
</tr>
<tr>
<td>------------</td>
<td>---------</td>
<td>---------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>overwrite</td>
<td>Optional</td>
<td>no</td>
<td>Specifies whether ColdFusion MX overwrites an existing file. Used in conjunction with the filename attribute.</td>
</tr>
<tr>
<td>name</td>
<td>Optional</td>
<td></td>
<td>Specifies the name of an existing variable into which the tag stores the PDF or FlashPaper output.</td>
</tr>
<tr>
<td>pagetype</td>
<td>Optional</td>
<td>A4</td>
<td>Specifies the page size into which ColdFusion generates the report:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• legal: 8.5 inches x 14 inches.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• letter: 8.5 inches x 11 inches.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• A4: 8.27 inches x 11.69 inches.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• A5: 5.81 inches x 8.25 inches.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• B5: 9.81 inches x 13.88 inches.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Custom: custom height and width. If you specify custom, you must also specify the pageheight and pagewidth attributes, can optionally specify margin attributes, and can optionally specify whether the units are inches or centimeters.</td>
</tr>
<tr>
<td>pageheight</td>
<td>Optional</td>
<td></td>
<td>Specifies the page height in inches (default) or centimeters. This attribute is only valid if pagetype=custom. To specify page height in centimeters, include the unit=cm attribute.</td>
</tr>
<tr>
<td>pagewidth</td>
<td>Optional</td>
<td></td>
<td>Specifies the page width in inches (default) or centimeters. This attribute is only valid if pagetype=custom. To specify page width in centimeters, include the unit=cm attribute.</td>
</tr>
<tr>
<td>orientation</td>
<td>Optional</td>
<td>portrait</td>
<td>Specifies the page orientation:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• portrait</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• landscape</td>
</tr>
<tr>
<td>margintop</td>
<td>Optional</td>
<td></td>
<td>Specifies the top margin in inches (default) or centimeters. To specify the top margin in centimeters, include the unit=cm attribute.</td>
</tr>
<tr>
<td>marginbottom</td>
<td>Optional</td>
<td></td>
<td>Specifies the bottom margin in inches (default) or centimeters. To specify the bottom margin in centimeters, include the unit=cm attribute.</td>
</tr>
<tr>
<td>marginleft</td>
<td>Optional</td>
<td></td>
<td>Specifies the left margin in inches (default) or centimeters. To specify the left margin in centimeters, include the unit=cm attribute.</td>
</tr>
<tr>
<td>marginright</td>
<td>Optional</td>
<td></td>
<td>Specifies the right margin in inches (default) or centimeters. To specify the right margin in centimeters, include the unit=cm attribute.</td>
</tr>
<tr>
<td>unit</td>
<td>Optional</td>
<td>in</td>
<td>Specifies the default unit for the pageheight, pagewidth, and margin attributes:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• in: inches.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• cm: centimeters.</td>
</tr>
</tbody>
</table>
Usage

Use the `<cfdocument>` tag to render HTML and CFML output into PDF or FlashPaper format. ColdFusion MX does not return HTML and CFML outside of the `<cfdocument>` pair.

The `<cfdocument>` tag can render HTML that supports the following standards:

- HTML 4.01
- XML 1.0
- DOM Level 1 and 2
- CSS1 and CSS2

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Req/Opt</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
</table>
| encryption      | Optional| None      | (format="PDF" only) Specifies whether the output is encrypted:  
- 128-bit  
- 40-bit  
- none |
| ownerpassword   | Optional|           | (format="PDF" only) Specifies an owner password. |
| userpassword    | Optional|           | (format="PDF" only) Specifies a user password. |
| permissions     | Optional|           | (format="PDF" only) Specifies one or more of the following permissions:  
- AllowPrinting  
- AllowModifyContents  
- AllowCopy  
- AllowModifyAnnotations  
- AllowFillIn  
- AllowScreenReaders  
- AllowAssembly  
- AllowDegradedPrinting  
Separate multiple permissions with a comma. |
| fontembed       | Optional| yes       | Specifies whether ColdFusion embeds fonts in the output:  
- yes: embed fonts.  
- no: do not embed fonts.  
Selective: embed all fonts except Java fonts and core fonts. For more information, see Usage. |
| backgroundvisible | Optional| no        | Specifies whether the background prints when the user prints the document:  
- yes: include the background when printing.  
- no: do not include the background when printing. |
| scale           | Optional| Calculated by ColdFusion | Specifies a scale factor as a percentage. Use this option to reduce the size of the HTML output so that it fits on that paper. Specify a number less than 100. |
The `cfdocument` tag does not support the Internet Explorer-specific HTML generated by Microsoft Word.

The PDF or FlashPaper document returned by the `cfdocument` tag overwrites any previous HTML in the input stream and ignores any HTML after the `</cfdocument>` tag.

You cannot embed a `cfreport` tag in a `cfdocument` tag.

When you use the `cfdocument` tag, ColdFusion MX creates a new scope named `cfdocument`. This scope contains the following variables:

- `currentpagenumber`
- `totalpagecount`

**Note:** The `cfdocument` scope variables are reserved for page number rendering. Do not use them in ColdFusion expressions. For example, the following code does not work:

```cfml
<cfif cfdocument.currentpagenumber gt 1>
  <cfoutput>#cfdocument.currentpagenumber-1#</cfoutput>
</cfif>
```

**Example**

```cfml
<cfdocument format="flashpaper">
<p>This is a document rendered by the cfdocument tag.</p>
<table width="50%" border="2" cellspacing="2" cellpadding="2">
  <tr>
    <td><strong>Name</strong></td>
    <td><strong>Role</strong></td>
  </tr>
  <tr>
    <td>Bill</td>
    <td>Lead</td>
  </tr>
  <tr>
    <td>Susan</td>
    <td>Principal Writer</td>
  </tr>
  <tr>
    <td>Adelaide</td>
    <td>Part Time Senior Writer</td>
  </tr>
  <tr>
    <td>Thomas</td>
    <td>Full Time for 6 months</td>
  </tr>
  <tr>
    <td>Michael</td>
    <td>Full Time for 4 months</td>
  </tr>
</table>
</cfdocument>
```
**cfdocumentitem**

**Description**

Specifies action items for a PDF or FlashPaper document created by the `cfdocument` tag. Actions include the following:

- Page break
- Header
- Footer

**Category**

Data output tags

**Syntax**

```xml
<cfdocument ...
Syntax 1
<cfdocumentitem type = "pagebreak"/>

Syntax 2
<cfdocumentitem
    type = "header" or "footer">
header/footer text
</cfdocumentitem>
</cfdocument>
```

**See also**

cfreport, cfdocument, cfdocumentsection

**History**

ColdFusion MX 7: Added this tag.

**Attributes**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Req/Opt</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>Required</td>
<td></td>
<td>Specifies the action:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- pagebreak: start a new page at the location of the tag.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- header: use the text between the <code>&lt;cfdocumentitem&gt;</code> and <code>&lt;/cfdocumentitem&gt;</code> tags as the running header.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- footer: use the text between the <code>&lt;cfdocumentitem&gt;</code> and <code>&lt;/cfdocumentitem&gt;</code> tags as the running footer.</td>
</tr>
</tbody>
</table>

**Usage**

Use the `cfdocumentitem` tag to control the formatting of a PDF or FlashPaper report. This tag must be wrapped inside a `<cfdocument>` `</cfdocument>` pair.

Write code for one `cfdocumentitem` tag for each pagebreak, running header, or running footer.
You can use the cfdocument scope variable, cfdocument.currentpagenumber, to display the current page number in a header or footer. You can also use cfdocument.totalpagecount to display the total number of pages. For example:

```
...<cfdocumentItem type="footer">
  #cfdocument.currentpagenumber# of #cfdocument.totalpagecount#
</cfdocumentItem>
```

You can use cfdocumentitem tags with or without the cfdocumentsection tag, as follows:

**Without cfdocumentsection**  The cfdocumentitem attribute applies to the entire document, as follows:

- If the tag is at the top of the document, it applies to the entire document.
- If the tag is in the middle of the document, it applies to the rest of the document.
- If the tag is at the end of the document, it has no affect.

**With cfdocumentsection tags**  The cfdocumentitem attribute applies only to the section and overrides previously specified header and footer specifications.

**Example**

```
<cfquery datasource="cfdocexamples" name="parksQuery">
SELECT parkname, suptmgr from parks
</cfquery>

<cfdocument format="PDF">
  <cfdocumentitem type="header">National Parks Report</cfdocumentitem>
  <!--- Use a footer with current page of totalpages format --->
  <cfdocumentitem type="footer">
    <cfoutput>Page #cfdocument.currentpagenumber# of #cfdocument.totalpagecount#</cfoutput>
  </cfdocumentitem>
  <h1>Park list</h1>
  <table width="95%" border="2" cellspacing="2" cellpadding="2">
    <tr>
      <th>Park</th>
      <th>Manager</th>
    </tr>
    <cfoutput query="parksQuery">
      <tr>
        <td><font size="-1">#parkname#</font></td>
        <td><font size="-1">#suptmgr#</font></td>
      </tr>
    </cfoutput>
  </table>
</cfdocument>
```
**cfdocumentsection**

**Description**

Divides a PDF or FlashPaper document into sections. By using this tag in conjunction with a `cfdocumentitem` tag, each section can have unique headers, footers, and page numbers.

**Category**

*Data output tags*

**Syntax**

```html
<cfdocument ...

<cfdocumentsection
    margintop = "number"
    marginbottom = "number"
    marginleft = "number"
    marginright = "number">

HTML, CFML, and cfdocumentitem tags
</cfdocumentsection>

</cfdocument>
```

**See also**

*cfreport, cfdocument, cfdocumentitem*

**History**

*ColdFusion MX 7: Added this tag and the margintop, marginbottom, marginleft, marginright attributes.*

**Attributes**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Req/Opt</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>margintop</td>
<td>Optional</td>
<td></td>
<td>Specifies the top margin in inches (default) or centimeters. To specify the top margin in centimeters, include the unit=&quot;cm&quot; attribute in the parent cfdocument tag.</td>
</tr>
<tr>
<td>marginbottom</td>
<td>Optional</td>
<td></td>
<td>Specifies the bottom margin in inches (default) or centimeters. To specify the bottom margin in centimeters, include the unit=&quot;cm&quot; attribute in the parent cfdocument tag.</td>
</tr>
<tr>
<td>marginleft</td>
<td>Optional</td>
<td></td>
<td>Specifies the left margin in inches (default) or centimeters. To specify the left margin in centimeters, include the unit=&quot;cm&quot; attribute in the parent cfdocument tag.</td>
</tr>
<tr>
<td>marginright</td>
<td>Optional</td>
<td></td>
<td>Specifies the right margin in inches (default) or centimeters. To specify the right margin in centimeters, include the unit=&quot;cm&quot; attribute in the parent cfdocument tag.</td>
</tr>
</tbody>
</table>
Usage

Use the `cfdocumentsection` tag to divide a report into sections. Within each `cfdocumentsection` tag, you can use one or more `cfdocumentitem` tags to specify unique headers and footers for each section.

When using `cfdocumentsection`, ColdFusion MX ignores HTML and CFML not enclosed within `cfdocumentsection` tags.

The margin attributes override margins specified in previous sections or in the parent `cfdocument` tag. If you specify margin attributes, the units are controlled by the unit attribute of the parent `cfdocument` tag; the unit attribute has a default value of inches.

Example

```cfquery datasource="cfdocexamples" name="empSalary">
SELECT Emp_ID, firstname, lastname, e.dept_id, salary, d.dept_name
FROM employee e, departmt d
WHERE e.dept_id = d.dept_id
ORDER BY d.dept_name
</cfquery>

<cfdocument format="PDF">
<cfoutput query="empSalary" group="dept_id">
<cfdocumentsection>
<cfdocumentitem type="header">
<font size="-3"><i>Salary Report</i></font>
</cfdocumentitem>
<cfdocumentitem type="footer">
<font size="-3">Page #cfdocument.currentpagenumber#</font>
</cfdocumentitem>
<h2>#dept_name#</h2>
<table width="95%" border="2" cellspacing="2" cellpadding="2" >
<tr>
<th>Employee</th>
<th>Salary</th>
</tr>
<cfset deptTotal = 0 >
<!--- inner cfoutput --->
<cfoutput>
<tr>
<td><font size="-1">
#empSalary.lastname#, #empSalary.firstname#</font></td>
<td align="right"><font size="-1">
$#DollarFormat(empSalary.salary)#</font></td>
</tr>
<cfset deptTotal = deptTotal + empSalary.salary>
</cfoutput>
<tr>
<td align="right"><font size="-1">Total</font></td>
<td align="right"><font size="-1">$#DollarFormat(deptTotal)#</font></td>
</tr>
<cfset deptTotal = 0>
</table>
```
**cfdump**

**Description**

Use the `cfdump` tag to get the elements, variables, and values of most kinds of ColdFusion objects. Useful for debugging. You can display the contents of simple and complex variables, objects, components, user-defined functions, and other elements.

**Category**

Debugging tags, Variable manipulation tags

**Syntax**

```xml
<cfdump
    var = "#variable#"
    expand = "yes" or "no"
    label = "text"
    top = "number of rows or levels">
```

**See also**

cfcookie, cfparam, cfsavecontent, cfschedule, cfset, cftimer, cfwddx

**History**

- ColdFusion MX 7: Added the `top` attribute.
- ColdFusion MX 6.1: Added the ability to dump COM objects; it displays the methods and Get and Put properties typeinfo information for the object.

**Attributes**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Req/Opt</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>var</td>
<td>Required</td>
<td></td>
<td>Variable to display. Enclose a variable name in number signs. These kinds of variables yield meaningful <code>cfdump</code> displays: array, CFC, COM object, Java object, simple, query, structure, UDF, wddx, xml</td>
</tr>
<tr>
<td>expand</td>
<td>Optional</td>
<td>yes</td>
<td>• yes: in Internet Explorer and Mozilla, expands views. • no: contracts expanded views.</td>
</tr>
<tr>
<td>label</td>
<td>Optional</td>
<td></td>
<td>A string; header for the dump output.</td>
</tr>
<tr>
<td>top</td>
<td>Optional</td>
<td>9999</td>
<td>The number of rows to display. For a structure, this is the number of nested levels to display.</td>
</tr>
</tbody>
</table>
**Usage**

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

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

```cfml
cif IsXmlDoc(mydoc) is "yes";
cfdump var="#mydoc#"
cif/
```

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

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

**Example**

```cfml
<!--- This example shows how to use this tag to display the CGI scope as a structure: --->

cfdump var="#cgi#"
```
**cfelse**

**Description**

Used as the last control block in a `cfif` tag block to handle any case not identified by the `cfif` tag or a `cfelseif` tag.

**Category**

Flow-control tags

**Syntax**

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

**See also**

`cfif`, `cfelseif`, `cfabort`, `cfbreak`, `cfexecute`, `cfexit`, `cflocation`, `cfloop`, `cfswitch`, `cfstream`, `cfttry`

**Usage**

If the values of the expressions in the containing `cfif` tag and all `cfelseif` tags are false, ColdFusion processes the code between this tag and the `cfif` end tag. This tag must be inside a `cfif` tag block. It does not require an end tag.

For more information and an example, see `cfif` on page 227.
cfelseif

Description

Used as a control block in a cfif tag block to handle any case not identified by the cfif tag or a cfelseif tag.

Category

Flow-control tags

Syntax

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

See also

cfif, cfelse, cfabort, cfbreak, cfexecute, cfexit, cflocation, cfloop, cfswitch, cftthrow, cftry

Usage

If the value of the expression in this tag is true, and the values of the expressions in the containing cfi tag and preceding cfelseif tags are false, ColdFusion processes the code between this tag and a following cfelseif or cfelse tag, or the cfif end tag and then skips to the code following the cfif end tag. Otherwise, ColdFusion skips the code.

This tag must be inside a cfi tag block. It does not require an end tag.

For more information and an example, see cfi on page 227.
cferror

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

Category
Exception handling tags, Extensibility tags, Application framework tags

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

See also
cfthrow, cfthrow, cftry, Chapter 14, “Handling Errors” in ColdFusion MX Developer’s Guide.

History
ColdFusion MX: Deprecated the monitor option of the exception attribute. It might not work, and might cause an error, in later releases.

Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Req/Opt</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>Required</td>
<td></td>
<td>Type of error that the custom error page handles. The type also determines how ColdFusion handles the error page. For more information, see “Specifying a custom error page” in Chapter 14, “Specifying a custom error page,” in ColdFusion MX Developer’s Guide.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• exception: an exception of the type specified by the exception attribute.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• validation: errors recognized by server-side type validation.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• request: any encountered error.</td>
</tr>
<tr>
<td>template</td>
<td>Required</td>
<td></td>
<td>Relative path to the custom error page. (A ColdFusion page was formerly called a template.)</td>
</tr>
</tbody>
</table>
Usage

Use this tag to provide custom error messages for pages in an application. This lets you maintain a consistent look and feel within the application, even when errors occur.

You generally embed this tag in your Application CFC or Application.cfm file to specify error-handling responsibilities for an entire application. You must put it in one of these files if you specify type="validation"; ColdFusion ignores it on any other page.

The cftry and cfcatch tags provide a more interactive way to handle ColdFusion errors within a ColdFusion page than the cferror tag, but the cferror tag is a good safeguard against general errors.

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

Page types

The following table describes the types of errors you can specify and code you can use on the pages that handle these error type:

<table>
<thead>
<tr>
<th>Page type</th>
<th>Description</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exception</td>
<td>Dynamically invoked by the CFML language processor when it detects an unhandled exception condition. Uses the full range of CFML tags. Error variables must be in cfoutput tags.</td>
<td>Can handle specific exception types or display general information for exceptions.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Req/Opt</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mailTo</td>
<td>Optional</td>
<td></td>
<td>An e-mail address. This attribute is available on the error page as the variable error.mailto. ColdFusion does not automatically send anything to this address.</td>
</tr>
<tr>
<td>exception</td>
<td>Optional</td>
<td>Any</td>
<td>Type of exception that the tag handles:  • application: application exceptions.  • database: database exceptions.  • template: ColdFusion page exceptions.  • security: security exceptions.  • object: object exceptions.  • missingInclude: missing include file exceptions.  • expression: expression exceptions.  • lock: lock exceptions.  • custom_type: developer-defined exceptions, defined in the cfthrow tag.  • any: all exception types.  For more information on exception types, see cftry.</td>
</tr>
</tbody>
</table>
### Error variables

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

The following table lists error variables:

<table>
<thead>
<tr>
<th>Page type</th>
<th>Error variable</th>
<th>Description</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>Request</td>
<td></td>
<td>Includes the error variables described in the Error Variables section. Cannot include CFML tags, but you can display values of the error variables by enclosing them in number signs (#), as in <code>#error.MailTo#</code>.</td>
<td>Use as a backup error handler to other error handling methods, including exception type.</td>
</tr>
<tr>
<td>Validation</td>
<td></td>
<td>Handles data input validation errors that occur when submitting a form that uses hidden form-field validation or onSubmit validation. Cannot include CFML tags, but you can display values of the error variables by enclosing them in number signs (#), as in <code>#Error.InvalidFields#</code>. You must specify the validation error handler in the Application.cfc or Application.cfm file.</td>
<td>Handles hidden form-field or onSubmit format validation errors only.</td>
</tr>
</tbody>
</table>

### Error variables

<table>
<thead>
<tr>
<th>Page type</th>
<th>Error variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Validation only</td>
<td>error.validationHeader</td>
<td>Validation message header text.</td>
</tr>
<tr>
<td></td>
<td>error.invalidFields</td>
<td>Unordered list of validation errors.</td>
</tr>
<tr>
<td>Request and Exception</td>
<td>error.validationFooter</td>
<td>Validation message footer text.</td>
</tr>
<tr>
<td></td>
<td>error.diagnostics</td>
<td>Detailed error diagnostics from ColdFusion MX.</td>
</tr>
<tr>
<td></td>
<td>error.mailTo</td>
<td>E-mail address (same as value in <code>cferror.MailTo</code>).</td>
</tr>
<tr>
<td></td>
<td>error.dateTime</td>
<td>Date and time when error occurred.</td>
</tr>
<tr>
<td></td>
<td>error.browser</td>
<td>Browser that was running when error occurred.</td>
</tr>
<tr>
<td></td>
<td>error.remoteAddress</td>
<td>IP address of remote client.</td>
</tr>
<tr>
<td></td>
<td>error.HTTPReferer</td>
<td>Page from which client accessed link to page where error occurred.</td>
</tr>
<tr>
<td></td>
<td>error.template</td>
<td>Page executing when error occurred.</td>
</tr>
<tr>
<td></td>
<td>error.generatedContent</td>
<td>The content generated by the page up to the point where the error occurred.</td>
</tr>
<tr>
<td></td>
<td>error.queryString</td>
<td>URL query string of client’s request.</td>
</tr>
</tbody>
</table>
Note: If type = "exception", you can substitute the prefix cferror for Error; for example, cferror.diagnostics, cferror.mailTo, or cferror.dateTime.

<table>
<thead>
<tr>
<th>Page type</th>
<th>Error variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exception only</td>
<td>error.message</td>
<td>Error message associated with the exception.</td>
</tr>
<tr>
<td></td>
<td>error.rootCause</td>
<td>The root cause of the exception. This structure contains the information that is returned by a cfcatch tag. For example, for a database exception, the SQL statement that caused the error is in the error.RootCause.Sql variable. For Java exceptions, this variable contains the Java servlet exception reported by the JVM as the cause of the &quot;root cause&quot; of the exception.</td>
</tr>
<tr>
<td></td>
<td>error.tagContext</td>
<td>Array of structures containing information for each tag in the tag stack. The tag stack consists of each tag that is currently open.</td>
</tr>
<tr>
<td></td>
<td>error.type</td>
<td>Exception type.</td>
</tr>
</tbody>
</table>

Page type Error variable Description
---
---
---
---
---
---
---

The root cause of the exception. This structure contains the information that is returned by a cfcatch tag. For example, for a database exception, the SQL statement that caused the error is in the error.RootCause.Sql variable. For Java exceptions, this variable contains the Java servlet exception reported by the JVM as the cause of the "root cause" of the exception.

Array of structures containing information for each tag in the tag stack. The tag stack consists of each tag that is currently open.

Exception type.

Example

```cfc
<h3>cferror Example</h3>

<!--- Example of cferror call within a page. NOTE: If you use cferror type="VALIDATION" you MUST put it in Application.cfc or Application.cfm --->
<cferror type = "REQUEST"
template = "request_err.cfm"
mailTo = "admin@mywebsite.com">

<!--- This query calls a non-existent datasource, triggering an error to be handled. --->
<cfquery name="testQuery" datasource="doesNotExist">
    select * from nothing
</cfquery>

<!--- Example of the page (request_err.cfm) to handle this error. --->
<html>
<head>
<title>We're sorry -- An Error Occurred</title>
</head>
<body>
<h2>We're sorry -- An Error Occurred</h2>
<p>
If you continue to have this problem, please contact #error.mailTo# with the following information:
</p>
<p>
</p>
<ul>
<li>Your Location: #error.remoteAddress#</li>
<li>Your Browser: #error.browser#</li>
<li>Date and Time the Error Occurred: #error.dateTime#</li>
<li>Page You Came From: #error.HTTPReferer#</li>
<li>Message Content: #error.diagnostics#</li>
</ul>
```
**cfexecute**

**Description**

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

**Category**

Extensibility tags, Flow-control tags

**Syntax**

```cfexecute
<cfexecute
  name = "application name"
  arguments = "command line arguments"
  outputFile = "output filename"
  variable = "variable name"
  timeout = "timeout interval">
  ...
</cfexecute>
```

**See also**

cfcollection, cfindex, cfobject, cfreport, cfsearch, cfwddx

**History**

ColdFusion MX 6.1:

- Added the `variable` attribute.
- Changed file path behavior for the `outputFile` attribute: if you do not specify an absolute file path in the `outputFile` attribute, the path is relative to the ColdFusion temporary directory.

**Attributes**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Req/Opt</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Required</td>
<td>Absolute path of the application to execute. On Windows, you must specify an extension; for example, C:\myapp.exe.</td>
<td></td>
</tr>
</tbody>
</table>
| arguments | Optional | Command-line variables passed to application. If specified as string, it is processed as follows:  
- Windows: passed to process control subsystem for parsing.  
- UNIX: tokenized into an array of arguments. The default token separator is a space; you can delimit arguments that have embedded spaces with double-quotation marks.  
If passed as array, it is processed as follows:  
- Windows: Elements are concatenated into a string of tokens, separated by spaces. Passed to process control subsystem for parsing.  
- UNIX: Elements are copied into an array of exec() arguments. |
Usage

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

Exceptions

Throws the following exceptions:

- If the application name is not found: java.io.IOException
- If the effective user of the ColdFusion executing thread does not have permissions to execute the process: a security exception

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

Example

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

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

**Description**

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

**Category**

Debugging tags, Flow-control tags

**Syntax**

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

**See also**

cfabort, cfbreak, cffexecute, cfif, cflocation, cfloop, cfswitch, cfthrow, cftry;

**Attributes**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Req/Opt</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
</table>
| method    | Optional| exitTag | - exitTag: aborts processing of currently executing tag.  
- exitTemplate: exits page of currently executing tag.  
- loop: re-executes body of currently executing tag. |

**Usage**

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

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

<table>
<thead>
<tr>
<th>Method value</th>
<th>Location of cfexit call</th>
<th>Behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td>exitTag</td>
<td>Base page</td>
<td>Terminate processing</td>
</tr>
<tr>
<td></td>
<td>Execution mode = Start</td>
<td>Continue after end tag</td>
</tr>
<tr>
<td></td>
<td>Execution mode = End</td>
<td>Continue after end tag</td>
</tr>
<tr>
<td>exitTemplate</td>
<td>Base page</td>
<td>Terminate processing</td>
</tr>
<tr>
<td></td>
<td>Execution mode = Start</td>
<td>Continue from first child in body</td>
</tr>
<tr>
<td></td>
<td>Execution mode = End</td>
<td>Continue after end tag</td>
</tr>
<tr>
<td>loop</td>
<td>Base page</td>
<td>Error</td>
</tr>
<tr>
<td></td>
<td>Execution mode = Start</td>
<td>Error</td>
</tr>
<tr>
<td></td>
<td>Execution mode = End</td>
<td>Continue from first child in body</td>
</tr>
</tbody>
</table>
Example

Example

Example

Example

Example

Example

Example

Example

Example

cfile

Description

Manages interactions with server files.

The following sections describe the actions of the cfile tag:

- cfile action = "append" on page 131
- cfile action = "copy" on page 133
- cfile action = "delete" on page 135
- cfile action = "move" on page 136
- cfile action = "read" on page 138
- cfile action = "readBinary" on page 140
- cfile action = "rename" on page 141
- cfile action = "upload" on page 143
- cfile action = "write" on page 146

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

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

Category

File management tags

Syntax

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

See also

cfdirectory

History

ColdFusion MX 7: Added the result attribute, which allows you to specify an alternate variable in which to receive result parameters. Used for action = "upload" action.

ColdFusion MX 6.1:

- Changed file path requirements: if you do not specify an absolute file path, the path is relative to the ColdFusion temporary directory, which is returned by the GetTempDirectory function.
- Changed behavior for action="read": if the file starts with a byte order mark (BOM) ColdFusion uses it to determine the character encoding.
- Changed behavior for action="upload" nameConflict="MakeUnique" ColdFusion now makes filenames unique by appending a incrementing number, 1 for the first file, 2 for the second and so on, to the name. In ColdFusion MX filenames were made unique by appending an additional "1" for each file, as in 1, 11, 111, and so on.
ColdFusion MX:

- Changed use of slashes in paths: you can use forward (/) or backward (\) slashes in paths on both UNIX and Windows systems.
- Changed file hierarchy requirements: ColdFusion does not require that you put files and directories that you manipulate with this tag below the root of the web server document directory.
- Changed directory path requirements for the destination attribute: a directory path that you specify in the destination attribute does not require a trailing slash.
- Deprecated the system value of the attributes attribute.
- Deprecated the temporary value of the attributes attribute. In ColdFusion MX, it is a synonym for normal. It might not work in later releases.
- Changed the action attribute options read, write, append and move: they support a new attribute, charset.
- The archive value of the attributes attribute is obsolete and has no effect.

Example

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

<cfparam name="fileExists" default="no">
<cfparam name="readText" default="">
```
<!-- First, check whether canned file exists. -->
<cfif FileExists("#GetTempDirectory()#foobar.txt") is "Yes">
  <cfset fileExists="yes">
</cfif>

<!-- Now, make the form that runs the example. -->
<form action="index.cfm" method="POST">
<h4>Type in some text to include in your file:</h4>
<p>
  <cfif fileExists is "yes">
    A file exists (foobar.txt, in <cfoutput>#GetTempDirectory()#</cfoutput>). You may add to it, read from it, or delete it. </p>
  </cfif>
</form>

<!-- If reading from a form, let that information display in textarea. -->
<textarea name="the_text" cols="40" rows="5">
  <cfif readText is not "">
    <cfoutput>#readText#</cfoutput>
  </cfif></textarea>

<!-- Select from the actions depending on whether the file exists. -->
<select name="action">
  <cfif fileExists is "no">
    <option value="new">Make new file</option>
  </cfif>
  <cfif fileExists is "yes">
    <option value="add">Add to existing file</option>
    <option value="delete">Delete file</option>
    <option value="read">Read existing file</option>
  </cfif>
</select>

<input type="Hidden" name="formsubmit" value="yes">
<input type="Submit" name="" value="make my changes">
</form>
cfile action = "append"

Description
Appends text to a text file on the server.

Syntax
```cfile
action = "append"
file = "full_path_name"
output = "string"
addNewLine = "yes" or "no"
attributes = "file_attributes_list"
mode = "mode"
charset = "charset_option"
```

See also
cfdirectory

History
See the History section of the main cfile tag page.

Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Req/Opt</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>action</td>
<td>Required</td>
<td></td>
<td>Type of file manipulation that the tag performs.</td>
</tr>
<tr>
<td>file</td>
<td>Required</td>
<td></td>
<td>Pathname of the file to which to append content of output attribute.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>If not an absolute path (starting a with a drive letter and a colon, or a forward or backward slash), it is relative to the ColdFusion temporary directory, which is returned by the GetTempDirectory function.</td>
</tr>
<tr>
<td>mode</td>
<td>Optional</td>
<td></td>
<td>Applies only to UNIX and Linux. Permissions. Octal values of UNIX chmod command. Assigned to owner, group, and other, respectively. For example:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• 644: assigns read/write permission to owner; read permission to group and other.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• 777: assigns read/write/execute permission to all.</td>
</tr>
<tr>
<td>output</td>
<td>Required</td>
<td></td>
<td>String to append to the file.</td>
</tr>
<tr>
<td>addNewLine</td>
<td>Optional</td>
<td>yes</td>
<td>• yes: appends newline character to text written to file.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• no</td>
</tr>
</tbody>
</table>
Example

<!--The first example creates the file \temp\foo on a windows system and sets attributes to normal. --->

<cffile action = "write" file = "\temp\foo" attributes = normal output = "some
text">

<!--- The second example appends to the file. --->

<cffile action = "append" file = "\temp\foo" attributes = normal output = "Is
this a test?"
**cfile action = "copy"**

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

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

**See also**
cfdirectory

**History**
See the History section of the main cfile tag page.

**Attributes**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Req/Opt</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>action</td>
<td>Required</td>
<td></td>
<td>Type of file manipulation that the tag performs.</td>
</tr>
<tr>
<td>source</td>
<td>Required</td>
<td></td>
<td>Pathname of the file to copy. If not an absolute path (starting a with a drive letter and a colon, or a forward or backward slash), it is relative to the ColdFusion temporary directory, which is returned by the GetTempDirectory function.</td>
</tr>
<tr>
<td>destination</td>
<td>Required</td>
<td></td>
<td>Pathname of a directory or file on web server where the file will be copied. If you specify a filename without a directory path, ColdFusion copies it relative to the source directory.</td>
</tr>
</tbody>
</table>
| mode      | Optional|         | Applies only to UNIX and Linux. Permissions. Octal values of UNIX chmod command. Assigned to owner, group, and other, respectively. For example:  
  • 644: assigns read/write permission to owner; read permission to group and other.  
  • 777: assigns read/write/execute permission to all. |
| attributes| Optional|         | Applies to Windows. A comma-delimited list of attributes to set on the file. If omitted, the file's attributes are maintained. Each value must be specified explicitly. For example, if you specify attributes = "readOnly", all other attributes are overwritten.  
  • readOnly  
  • hidden  
  • normal |
Example

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

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

**Description**

Deletes a file on the server.

**Syntax**

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

**See also**

`cfdirectory`

**Attributes**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Req/Opt</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>action</td>
<td>Required</td>
<td></td>
<td>Type of file manipulation that the tag performs.</td>
</tr>
<tr>
<td>file</td>
<td>Required</td>
<td></td>
<td>Pathname of the file to delete. If not an absolute path (starting a with a drive letter and a colon, or a forward or backward slash), it is relative to the ColdFusion temporary directory, which is returned by the GetTempDirectory function.</td>
</tr>
</tbody>
</table>

**Example**

The following example deletes the specified file:

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

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

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

**See also**
`cfdirectory`

**History**
See the History section of the main cffile tag page.

**Attributes**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Req/Opt</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>action</td>
<td>Required</td>
<td></td>
<td>Type of file manipulation that the tag performs.</td>
</tr>
<tr>
<td>source</td>
<td>Required</td>
<td></td>
<td>Pathname of the file to move. If not an absolute path (starting a with a drive letter and a colon, or a forward or backward slash), it is relative to the ColdFusion temporary directory, which is returned by the <code>GetTempDirectory</code> function.</td>
</tr>
<tr>
<td>destination</td>
<td>Required</td>
<td></td>
<td>Pathname of the destination directory or file. If not an absolute path, it is relative to the source directory.</td>
</tr>
</tbody>
</table>
| mode      | Optional|         | Applies only to UNIX and Linux. Permissions. Octal values of UNIX chmod command. Assigned to owner, group, and other, respectively. For example:  
- 644: assigns read/write permission to owner; read permission to group and other.  
- 777: assigns read/write/execute permission to all. |

---

136 Chapter 2: ColdFusion Tags
Example

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

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

In this example, the destination directory is “memo.”

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Req/Opt</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>attributes</td>
<td>Optional</td>
<td></td>
<td>Applies to Windows. A comma-delimited list of attributes to set on the file. If omitted, the file’s attributes are maintained. Each value must be specified explicitly. For example, if you specify attributes = &quot;readOnly&quot;, all other attributes are overwritten.</td>
</tr>
<tr>
<td>charset</td>
<td>Optional</td>
<td></td>
<td>The character encoding in which the file contents is encoded. The following list includes commonly used values: utf-8, iso-8859-1, windows-1252, us-ascii, shift_jis, iso-2022-jp, euc-jp, euc-kr, big5, euc-cn, utf-16. For more information character encodings, see <a href="http://www.w3.org/International/O-charset.html">www.w3.org/International/O-charset.html</a>.</td>
</tr>
</tbody>
</table>

For more information on the attributes, see [cfile](http://).
cfile action = "read"

Description
Reads a text file on the server. The file is read into a dynamic, local variable that you can use in
the page. For example:
• Read a text file; insert the file's contents into a database
• Read a text file; use the find and replace function to modify the file's contents

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

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

See also
cfdirectory

History
See the History section of the main cfile tag page.

Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Req/Opt</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>action</td>
<td>Required</td>
<td></td>
<td>Type of file manipulation that the tag performs.</td>
</tr>
<tr>
<td>file</td>
<td>Required</td>
<td></td>
<td>Pathname of the file to read. If not an absolute path (starting a with a drive letter and a colon, or a forward or backward slash), it is relative to the ColdFusion temporary directory, which is returned by the GetTempDirectory function.</td>
</tr>
</tbody>
</table>
The following example creates a variable named Message for the contents of the file message.txt:

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

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

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

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

**Note:** If you use this tag to read a file that is encoded using the Windows Cp1252 (windows-1252) encoding of the Latin-1 character set on a system whose default character encoding is Cp1252, and the file has characters encoded in the Hex 8x or 9x range, you must specify `charset="windows-1252"` attribute, even though this is the default encoding. Otherwise, some characters in the Hex8x and 9x ranges that do not map correctly and display incorrectly.

---

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Req/Opt</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>variable</td>
<td>Required</td>
<td></td>
<td>Name of variable to contain contents of text file.</td>
</tr>
<tr>
<td>charset</td>
<td>Optional</td>
<td></td>
<td>The character encoding in which the file contents is encoded. The following list includes commonly used values: utf-8, iso-8859-1, windows-1252, us-ascii, shift_jis, iso-2022-jp, euc-jp, euc-kr, big5, euc-cn, utf-16. If the file starts with a byte order mark and you set this attribute to a conflicting character encoding, ColdFusion generates an error. For more information character encodings, see <a href="http://www.w3.org/International/O-charset.html">www.w3.org/International/O-charset.html</a>.</td>
</tr>
</tbody>
</table>

---

Usage

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

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

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

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

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

**Note:** If you use this tag to read a file that is encoded using the Windows Cp1252 (windows-1252) encoding of the Latin-1 character set on a system whose default character encoding is Cp1252, and the file has characters encoded in the Hex 8x or 9x range, you must specify `charset="windows-1252"` attribute, even though this is the default encoding. Otherwise, some characters in the Hex8x and 9x ranges that do not map correctly and display incorrectly.
cfile action = "readBinary"

Description

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

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

Syntax

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

See also

cdirectory

Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Req/Opt</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>action</td>
<td>Required</td>
<td></td>
<td>Type of file manipulation that the tag performs.</td>
</tr>
<tr>
<td>file</td>
<td>Required</td>
<td></td>
<td>Pathname of a binary file to read.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>If not an absolute path (starting a with a drive letter and a colon, or a forward or backward slash), it is relative to the ColdFusion temporary directory, which is returned by the GetTempDirectory function.</td>
</tr>
<tr>
<td>variable</td>
<td>Required</td>
<td></td>
<td>Name of variable to contain contents of binary file.</td>
</tr>
</tbody>
</table>

Usage

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

Example

The following example reads the binary file somewhere.jpg, writes it to a different folder as somewhereB.jpg, and then displays the new file:

```<cfile action = "readBinary" file = "C:\inetpub\wwwroot\cfdocs\getting_started\photos\somewhere.jpg" variable = "aBinaryObj">

<!--- Output binary object to JPEG format for viewing. --->
<cfile action="write" file = "c:\files\updates\somewhereB.jpg" output = "#toBinary(aBinaryObj)#">

<!--- HTML to view image. --->
<img src="C:\files\updates\somewhereB.jpg">
```
cfile action = "rename"

Description
Renames or moves a file on the server.

Syntax
<cfcontent
    action = "rename"
    source = "full_path_name"
    destination = "path_name"
    mode = "mode"
    attributes = "file_attributes_list">

See also
cfdirectory

History
See the History section of the main cffile tag page.

Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Req/Opt</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>action</td>
<td>Required</td>
<td></td>
<td>Type of file manipulation that the tag performs.</td>
</tr>
<tr>
<td>source</td>
<td>Required</td>
<td></td>
<td>Pathname of file to rename.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>If not an absolute path (starting a with a drive letter and a colon, or a forward or backward slash), it is relative to the ColdFusion temporary directory, which is returned by the GetTempDirectory function.</td>
</tr>
<tr>
<td>destination</td>
<td>Required</td>
<td></td>
<td>Destination file or directory. If not an absolute path, it is relative to the source directory.</td>
</tr>
<tr>
<td>mode</td>
<td>Optional</td>
<td></td>
<td>Applies only to UNIX and Linux. Permissions. Octal values of UNIX chmod command. Assigned to owner, group, and other. For example:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• 644: assigns read/write permission to owner; read permission to group and other.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• 777: assigns read/write/execute permission to all.</td>
</tr>
<tr>
<td>attributes</td>
<td>Optional</td>
<td></td>
<td>Applies to Windows. A comma-delimited list of attributes to set on the file.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>If omitted, the file’s attributes are maintained.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Each value must be specified explicitly. For example, if attributes = &quot;readOnly&quot;, all other attributes are overwritten.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• hidden</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• normal</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• readOnly</td>
</tr>
</tbody>
</table>

Usage
The rename action renames or move a file. The destination attribute must be a pathname, not just a new name for the file. If the destination is a directory, the file is moved and not renamed.
Example

Windows Example:

<!--- Source Document is read-only but when renamed it becomes normal (not hidden or read-only). -->
<cffile action = "rename" source = "c:\files\memo\readonlymemo.doc"
       destination = "c:\files\memo\normalmemo.doc" attributes="normal">

Unix Example:

<cffile action = " rename " source = "#myWR#/memo/sample.txt" destination =
           "#myWR#/memo/other_sample.txt" mode="666">
cfile action = "upload"

Description
Copies a file to a directory on the server.

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

See also
cfdirectory

History
See the History section of the main cffile tag page.

Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Req/Opt</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>action</td>
<td>Required</td>
<td></td>
<td>Type of file manipulation that the tag performs.</td>
</tr>
<tr>
<td>fileField</td>
<td>Required</td>
<td></td>
<td>Name of form field used to select the file.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Do not use number signs (#) to specify the field name.</td>
</tr>
<tr>
<td>destination</td>
<td>Required</td>
<td></td>
<td>Pathname of directory in which to upload the file.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>If not an absolute path (starting a with a drive letter and a colon, or a</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>forward or backward slash), it is relative to the ColdFusion temporary</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>directory, which is returned by the GetTempDirectory function.</td>
</tr>
<tr>
<td>nameConflict</td>
<td>Optional</td>
<td>Error</td>
<td>Action to take if filename is the same as that of a file in the directory.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Error: file is not saved. ColdFusion stops processing the page and</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>returns an error.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Skip: file is not saved. This option permits custom behavior based on</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>file properties.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Overwrite: replaces file.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• MakeUnique: forms a unique filename for the upload; name is stored in</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>the file object variable serverFile.</td>
</tr>
<tr>
<td>accept</td>
<td>Optional</td>
<td></td>
<td>Limits the MIME types to accept. Comma-delimited list. For example, to</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>permit JPEG and Microsoft Word file uploads:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>accept = &quot;image/jpg, application/msword&quot;</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>The browser uses file extension to determine file type.</td>
</tr>
</tbody>
</table>
After a file upload is completed, you can get status information using file upload parameters. To refer to parameters, use either the `cffile` prefix or, if you specified an alternate name in the `result` attribute, the name you specified there.

For example, if you did not specify a name in the `result` attribute, access the `fileExisted` parameter as `#cffile.fileExisted#`. If you set the `result` attribute to `myResult`, however, access `fileExisted` as `#myResult.fileExisted#`.

Status parameters can be used anywhere that other ColdFusion parameters can be used.

**Tip:** The `result` attribute allows functions or CFCs that get called from multiple pages at the same time to avoid overwriting the results of one call with another.

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

**Tip:** If your page is uploading a file that was selected on a form or was otherwise sent to your page via a multipart/form-data HTTP message, you can determine the approximate size of the file by checking the value of the `CGI.content_length` variable. This variable includes the file length plus the length of any other request content.

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>attemptedServerFile</code></td>
<td>Initial name ColdFusion used when attempting to save a file</td>
</tr>
<tr>
<td><code>clientDirectory</code></td>
<td>Directory location of the file uploaded from the client’s system</td>
</tr>
<tr>
<td><code>clientFile</code></td>
<td>Name of the file uploaded from the client’s system</td>
</tr>
<tr>
<td><code>clientFileExt</code></td>
<td>Extension of the uploaded file on the client system (without a period)</td>
</tr>
</tbody>
</table>
Note: File status parameters are read-only. They are set to the results of the most recent `cffile` operation. If two `cffile` tags execute, the results of the second overwrite the first, unless you have specified a different result variable in the `result` attribute.

Example

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

```<cfif isDefined("Form.FileContents")>
   <!- - If TRUE, upload the file. --->
   <cffile action = "upload"
       fileField = "FileContents"
       destination = "c:\files\upload\"
       accept = "text/html"
       nameConflict = "MakeUnique">
</cfif>
```

Parameter Description

- **clientFileName**: Name of the uploaded file on the client system (without an extension)
- **contentSubType**: MIME content subtype of the saved file
- **contentType**: MIME content type of the saved file
- **dateLastAccessed**: Date and time the uploaded file was last accessed
- **fileExisted**: Whether the file already existed with the same path (yes or no)
- **fileSize**: Size of the uploaded file
- **fileWasAppended**: Whether ColdFusion appended uploaded file to a file (yes or no)
- **fileWasOverwritten**: Whether ColdFusion overwrote a file (yes or no)
- **fileWasRenamed**: Whether uploaded file renamed to avoid a name conflict (yes or no)
- **fileWasSaved**: Whether ColdFusion saves a file (yes or no)
- **oldFileSize**: Size of a file that was overwritten in the file upload operation
- **serverDirectory**: Directory of the file saved on the server
- **serverFile**: Filename of the file saved on the server
- **serverFileExt**: Extension of the uploaded file on the server (without a period)
- **serverFileName**: Name of the uploaded file on the server (without an extension)
- **timeCreated**: Time the uploaded file was created
- **timeLastModified**: Date and time of the last modification to the uploaded file
**cfﬁle action = "write"**

**Description**

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

**Syntax**

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

**See also**

**cfdirectory**

**History**

See the History section of the main `cfﬁle` tag page.

**Attributes**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Req/Opt</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>action</td>
<td>Required</td>
<td></td>
<td>Type of ﬁle manipulation that the tag performs.</td>
</tr>
<tr>
<td>file</td>
<td>Required</td>
<td></td>
<td>Pathname of the ﬁle to write.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>If not an absolute path (starting a with a drive letter and a colon, or a forward or backward slash), it is relative to the ColdFusion temporary directory, which is returned by the <code>GetTempDirectory</code> function.</td>
</tr>
<tr>
<td>output</td>
<td>Required</td>
<td></td>
<td>Content of the ﬁle to be created.</td>
</tr>
<tr>
<td>mode</td>
<td>Optional</td>
<td></td>
<td>Applies only to UNIX and Linux. Permissions. Octal values of UNIX chmod command. Assigned to owner, group, and other, respectively. For example:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• 644: assigns read/write permission to owner; read permission to group and other.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• 777: assigns read/write/execute permission to all.</td>
</tr>
<tr>
<td>addNewLine</td>
<td>Optional</td>
<td>yes</td>
<td>• yes: appends newline character to text written to ﬁle.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• no</td>
</tr>
</tbody>
</table>
Example

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

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

If the user submitted a form with the following:

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

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

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

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Req/Opt</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>attributes</td>
<td>Optional</td>
<td></td>
<td>Applies to Windows. A comma-delimited list of attributes to set on the file. If omitted, the file's attributes are maintained. Each value must be specified explicitly. For example, if you specify attributes = &quot;readOnly&quot;, all other attributes are overwritten. • readOnly • hidden • normal</td>
</tr>
<tr>
<td>charset</td>
<td>Optional</td>
<td>JVM default file character set.</td>
<td>The character encoding in which the file contents is encoded. The following list includes commonly used values: • utf-8 • iso-8859-1 • windows-1252 • us-ascii • shift_jis • iso-2022-jp • euc-jp • euc-kr • big5 • euc-cn • utf-16 For more information character encodings, see <a href="http://www.w3.org/International/O-charset.html">www.w3.org/International/O-charset.html</a>.</td>
</tr>
<tr>
<td>fixnewline</td>
<td>Optional</td>
<td>No</td>
<td>• Yes: changes embedded line-ending characters in string variables to operating-system specific line endings. • No: (default) do not change embedded line-ending characters in string variables.</td>
</tr>
</tbody>
</table>
This example shows the use of the `mode` attribute for UNIX. It creates the file `/tmp/foo` with permissions `rw-r--r--` (owner = read/write, group = read, other = read):

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

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

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

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

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

This example uses the `fixnewline` attribute to changes embedded line-ending characters in `xmlString`, which is derived from `xmlData`, to operating-system specific line endings.

```cfc
<cfxml variable="xmlData">
    <docroot>
        <payload type="string">This is some plain text</payload>
    </docroot>
</cfxml>
<cfset xmlString = toString(xmlData)>
<cfset key = createUUID()>
<cfset encString=encrypt(xmlString, key)>
<cffile action="write" addnewline="yes"
    file="C:\CFusionMX7\wwwroot\test\store.dat" output="#encString#"
    fixnewline="yes">
<cffile action="read" file="C:\CFusionMX7\wwwroot\test\store.dat"
    variable="retrievedString">
<cfset decString=decrypt(retrievedString, key)>
<cfset newXML = xmlParse(decString)>
<cfset newXML = xmlParse(decString)>
<cfset newXML = xmlParse(decString)>
```
**cflush**

**Description**
Flushes currently available data to the client.

**Category**
Data output tags, Page processing tags

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

**See also**
cache, cfheader, cfinclude, cfsetting, csilent

**Attributes**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Req/Opt</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>interval</td>
<td>Optional</td>
<td>Integer</td>
<td>Fl ushes output each time this number of bytes becomes available. HTML headers, and data that is already available when the tag is executed, are omitted from the count.</td>
</tr>
</tbody>
</table>

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

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

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

**Caution:** Because the cflush tag sends data to the browser when it executes, it has several limitations, including the following: Using any of the following tags or functions on a page anywhere after the cflush tag can cause errors or unexpected results: cfcontent, cfcookie, cfform, cfheader, cfhtmlhead, cflocation, and SetLocale. (These tags and functions normally modify the HTML header, but cannot do so after a cflush tag, because the cflush sends the header.) Using the cfsavecookie tag to set a cookie anywhere on a page that has a cflush tag does not set the cookie in the browser. Using the cflush tag within the body of several tags, including cfsavecontent, cfquery, and custom tags, cause errors. If you save Client variables as cookies, any client variables that you set after a cflush tag are not saved in the browser.

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

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

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

<cfflush>
<cfloop index="randomindex" from="1" to="200000" step="1">
    <cfset random=rand()>
</cfloop>

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

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

Description
Builds a form with CFML custom control tags; these provide more functionality than standard HTML form input elements. You can include the resulting form on the client page as HTML or Flash content, and generate the form using XML and XSLT.

Category
Forms tags

Syntax
<cfform
  name = "name"
  action = "form_action"
  method = "POST" or "GET"
  format = "HTML" or "Flash" or "XML"
  skin = "Flash or XSL skin"
  style = "style specification"
  preserveData = "yes" or "no"
  onSubmit = "javascript"
  scriptSrc = "path"
  codeBase = "URL"
  archive = "URL"
  The following attributes are supported in Flash and XML only
  width = "pixels or percent"
  height = "pixels or percent"
  The following attributes are supported in Flash only
  onError = "ActionScript code"
  wMode = "window" or "transparent" or "opaque"
  accessible = "yes" or "no"
  preloader = "yes" or "no"
  timeout = "seconds"
  The following attributes are supported in HTML and XML only
  class = "form class"
  enctype = "Internet media type"
  id = "HTML id"
  onload = "load event script"
  onreset = "reset event script"
  target = "target window or frame">
  ...
</cfform>

See also
cfapplet, cfcalendar, cfformgroup, cfformitem, cfgrid, cfinput, cfselect, cfslider, cftextarea, cftree; Part V, “Requesting and Presenting Information” in ColdFusion MX
Developer’s Guide

History
ColdFusion MX 7:
- Added ability to set the default value of the scriptSrc attribute in the ColdFusion MX Administrator.
• Deprecated the `passthrough` attribute. The tag now supports all HTML `form` tag attributes directly.

• Added the `method` attribute and support for the GET method.

• Added support for Flash and XML output, including the `format`, `height`, `width`, `preloader`, `timeout`, `wMode`, `accessible`, and `skin` attributes.

• Added `cfformgroup`, `cfformitem`, and `cftextarea` child tags.

ColdFusion MX:

• Deprecated the `enableCAB` attribute. It might not work, and might cause an error, in later releases.

• Changed the `name` and `action` attributes to optional.

• Changed integer validation to require an integer value. In previous releases it would convert a floating point value to an integer.

**Attributes**

The following table lists attributes that ColdFusion uses directly. For HTML format forms, this tag also supports the standard HTML `form` tag attributes that are not on this list, and passes them directly to the browser. ColdFusion also includes all supported HTML attributes in the XML.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Applies to</th>
<th>Req/Opt</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>HTML, Flash, XML</td>
<td>Opt</td>
<td>CFForm_n</td>
<td>A name for the form. In HTML format, if you omit this attribute and specify an <code>id</code> attribute, ColdFusion does not include a <code>name</code> attribute in the HTML sent to the browser; this behavior lets you use the <code>cfform</code> tag to create X+HTML-compliant forms. If you omit the <code>name</code> attribute and the <code>id</code> attribute, ColdFusion generates a name of the form CFForm_n where n is a number that assigned serially to the forms on a page.</td>
</tr>
<tr>
<td>action</td>
<td>HTML, Flash, XML</td>
<td>Opt</td>
<td>See Description</td>
<td>Name of ColdFusion page to execute when the form is submitted for processing. If you omit this attribute, the form posts to the page identified by the CGI.SCRIPT_NAME variable, the requested page that resulted in displaying the form.</td>
</tr>
<tr>
<td>method</td>
<td>HTML, Flash, XML</td>
<td>Opt</td>
<td>post</td>
<td>The method the browser uses to send the form data to the server: • post: sends the data using the HTTP post method. This method sends the data in a separate message to the server. • get: sends the data using the HTTP get method, which puts the form field contents in the URL query string.</td>
</tr>
<tr>
<td>Attribute</td>
<td>Applies to</td>
<td>Req/Opt</td>
<td>Default</td>
<td>Description</td>
</tr>
<tr>
<td>-----------</td>
<td>------------</td>
<td>---------</td>
<td>---------</td>
<td>-------------</td>
</tr>
</tbody>
</table>
| format    | HTML, Flash, XML | Opt | HTML | • HTML: generates an HTML form and send it to the client. cfgrid and cftree child controls can be in Flash or applet format.  
• Flash: generates a Flash form and send it to the client. All controls are in Flash format.  
• XML: generates XForms-compliant XML and save the results in a variable specified by the name attribute. By default, ColdFusion also applies an XSL skin and displays the result. For more information, see the skin attribute. |
| skin      | Flash, XML | Opt | Flash: haloGreen XML: default.xsl | Flash: Use a Macromedia halo color to stylize the output. The skin determines the color used for highlighted and selected elements.  
• haloSilver  
• haloBlue  
• haloGreen  
• haloOrange  
XML: Specifies whether to apply an XSL skin and display the resulting HTML to the client. Can be any of the following:  
• ColdFusion MX skin name: applies the specified skin.  
• XSL file name: applies the skin located in the specified path.  
• "none": does not apply an XSL skin. Your CFML page must process the XML that ColdFusion saves in the variable specified by the name attribute and display any results.  
• (omitted) or "default": uses the ColdFusion MX default skin.  
You can specify the following ColdFusion MX skins (located in the cf_webroot\CFIDE\scripts\xsl directory):  
• basic  
• basiccss  
• beige  
• blue  
• bluegray  
• lightgray  
• red  
• silver  
A filename can be any of the following:  
• absolute URL  
• URL relative to the web root  
• absolute file path  
• name of a file in the scripts folder or a subdirectory of the cf_webroot\CFIDE\scripts directory. In this case, do not specify the .xsl suffix. |
<table>
<thead>
<tr>
<th>Attribute</th>
<th>Applies to</th>
<th>Req/Opt</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>style</td>
<td>HTML, Flash, XML</td>
<td>Opt</td>
<td></td>
<td>Styles to apply to the form. In HTML or XML format, ColdFusion passes the style attribute to the browser or XML. In Flash format, must be a style specification in CSS format. For detailed information on specifying Flash styles, see Chapter 29, “Creating Forms in Macromedia Flash” in ColdFusion MX Developer’s Guide.</td>
</tr>
</tbody>
</table>
| preserveData | HTML, XML   | Opt     | no      | When the cfform action attribute posts back to the page that contains the form, this determines whether to override the control values with the submitted values.  
• false: uses values specified in the control tag attributes.  
• true: uses corresponding submitted values.  
Applies to these controls:  
• cfinput, cfslider, cftextinput: overrides the value attribute value.  
• cfselect controls that are populated from queries:Overrides the selected attribute. See cfselect.  
• cftree controls: overrides the cftreeitem expand attribute. If true, expands previously-selected elements. The cftree completePath attribute must be set to yes.  
• cfgrid controls: has no effect. (This avoids confusion as to whether data has been resubmitted to the database by the control.) |
<p>| onLoad      | HTML, XML   | Opt     |         | JavaScript to execute when the form loads. |
| onReset     | HTML, XML   | Opt     |         | JavaScript to execute when the user clicks a reset button. |
| onSubmit    | HTML, Flash, XML | Opt     |         | JavaScript or ActionScript function to execute to preprocess data before form is submitted. See ColdFusion MX Developer’s Guide. If any child tags specify onSubmit field validation, ColdFusion does the validation before executing this JavaScript. |</p>
<table>
<thead>
<tr>
<th>Attribute</th>
<th>Applies to</th>
<th>Req/Opt</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>scriptSrc</td>
<td>HTML, Flash, XML</td>
<td>Opt</td>
<td>See Description</td>
<td>Specifies the URL, relative to the web root, of the directory that contains the cfform.js file with the client-side JavaScript used by this tag and its child tags. For XML format forms, this directory is also the default directory for XSLT skins. This attribute is useful if the file is not in the default location. This attribute may be required in some hosting environments and configurations that block access to the /CFIDE directory. The default location is set in the ColdFusion MX Administrator; by default, it is /CFIDE/scripts/cfform.js. If the Administrator has an empty default value, ColdFusion looks for the script in the directory that contains the ColdFusion page.</td>
</tr>
<tr>
<td>codeBase</td>
<td>applets in HTML and XML</td>
<td>Opt</td>
<td>See Description</td>
<td>URL of downloadable JRE plug-in for Internet Explorer; used for cfform, cfgrid, cfslider, and cftree Java applet controls. Default: /CFIDE/classes/cf-j2re-win.cab</td>
</tr>
<tr>
<td>archive</td>
<td>applets in HTML and XML</td>
<td>Opt</td>
<td>See Description</td>
<td>URL of downloadable Java classes for cfform, cfgrid, cfslider, and cftree applet controls. Default: /CFIDE/classes/cfapplets.jar</td>
</tr>
<tr>
<td>height</td>
<td>Flash XML</td>
<td>Opt</td>
<td>100%</td>
<td>The height of the form. Use a number to specify pixels. In Flash, you can use a percentage value, such as &quot;height=60%&quot; to specify a percentage of the available width. The displayed height might be less than the specified size.</td>
</tr>
<tr>
<td>width</td>
<td>Flash XML</td>
<td>Opt</td>
<td>100%</td>
<td>The width of the form. Use a number to specify pixels. In Flash, you can use a percentage value, such as &quot;width=60%&quot; to specify a percentage of the available width..</td>
</tr>
<tr>
<td>onError</td>
<td>Flash</td>
<td>Opt</td>
<td></td>
<td>Applies only for onSubmit or onBlur validation; has no effect for onServer validation. An ActionScript expression or expressions to execute if the user submits a form with one or more validation errors.</td>
</tr>
</tbody>
</table>
### Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Applies to</th>
<th>Req/Opt</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>wMode</td>
<td>Flash</td>
<td>Opt</td>
<td>Window</td>
<td>Specifies how the Flash form appears relative to other displayable content that occupies the same space on an HTML page.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>- window: the Flash form is the topmost layer on the page and obscures anything that would share the space, such as drop-down dynamic HTML lists.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>- transparent: the Flash form honors the z-index of dhtml so you can float items above it. If the Flash form is above any item, transparent regions in the form show the content that is below it.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>- opaque: the Flash form honors the z-index of dhtml so you can float items above it. If the Flash form is above any item, it blocks any content that is below it.</td>
</tr>
<tr>
<td>accessible</td>
<td>Flash</td>
<td>Opt</td>
<td>No</td>
<td>Specifies whether to include support screen readers in the Flash form. Screen reader support adds approximately 80KB to the SWF file sent to the client.</td>
</tr>
<tr>
<td>preloader</td>
<td>Flash</td>
<td>Opt</td>
<td>true</td>
<td>Specifies whether to display a progress bar when loading the Flash form.</td>
</tr>
<tr>
<td>timeout</td>
<td>Flash</td>
<td>Opt</td>
<td>0</td>
<td>Integer number of seconds for which to keep the form data in the Flash cache on the server. A value of 0 prevents the data from being cached. For more information, see “Caching data in Flash forms” in ColdFusion MX Developer’s Guide.</td>
</tr>
</tbody>
</table>

**Note:** Attributes that are not marked as supported in XML are not handled by the skins provided with ColdFusion MX. They are, however, included in the generated XML as html namespace attributes to the form tag.

### Usage

This tag requires an end tag.

You can use the following ColdFusion form control tags within the cfform tag:

- **cfapplet** Used in HTML and XML format only; embeds a registered Java applet.
- **cfformgroup** Used in Flash and XML format only; groups and arranges child controls.
- **cfformitem** Used in Flash and XML format only; adds horizontal rules, vertical rules, and text to the form.
- **cgrid** Creates a grid control to display tabular data.
- **cfinput** Creates and an input element.
- **cfselect** Creates a drop-down list box.
- **cslider** Used in HTML and XML format only; creates a slider control.
- **cftextarea** Creates a multi-line text input box.
- **cftree** Creates a tree control.
In HTML format, all tags, and in Flash format the cftree and cfgrid tags, require JavaScript support on the browser. The cfapplet tag and applet format cfgrid, cfslider, and cftree tags require the client to download a Java applet.

If you specify Flash format in the cfform tag, ColdFusion ignores any HTML in the form body. You must use ColdFusion tags, such as cfinput, for all form controls. You can include individual Flash format cfgrid and cftree controls in an HTML format cfform tag.

In Flash format, if your forms do not request sensitive data (such as credit card numbers), consider setting the timeout attribute. This can prevent users from getting “The form data has expired, Please reload this page in your browser” errors if they use the browser back button to return to the form. For more information, see “Caching data in Flash forms” in Chapter 29, “Caching data in Flash forms,” in *ColdFusion MX Developer’s Guide*.

**Note:** In Flash format, if you do not specify height and width attributes, Flash reserves browser space equal to the area of the browser window. If any other output follows the form, users must scroll to see it. Therefore, if you follow a Flash form with additional output, specify the height and width values.

If attribute value text must include quotation marks, escape them by doubling them.

### Using the onError attribute in Flash forms

If you use onSubmit or onBlur validation, the onError attribute lets you specify ActionScript code to execute if the user tries to submit a Flash form with validation errors, as follows:

- If you specify one or more valid Flash expressions, Flash executes the expressions.
- If you omit the attribute, Flash displays a dialog box with all applicable error messages.
- If you specify onError="" (an empty string) Flash does not display any message, but does not submit the form.

Your ActionScript can use the errors variable to determine the fields and errors. The errors object has the following fields:

<table>
<thead>
<tr>
<th>Field</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>The name attribute of the control’s CFML tag.</td>
</tr>
<tr>
<td>field</td>
<td>The internal name used by Flash for the field. name (for example, _level0.field1)</td>
</tr>
<tr>
<td>value</td>
<td>The value in the field.</td>
</tr>
<tr>
<td>message</td>
<td>The message attribute of the control’s CFML tag.</td>
</tr>
</tbody>
</table>

The following example shows cfform tags with an onError attribute that selects the tab in an accordion or tabnavigator that contains a lastName field with an invalid entry:

```xml
<cfform name="form1" format="flash" width="800" height="500"
onError="if (errors['lastName'] != undefined)
{tabA.selectedIndex=0; _root.lastName.setFocus();}"
>```
Incorporating HTML form tags and attributes

In HTML format, the `cfform` tag lets you incorporate the following standard HTML elements. They are not available in Flash format:

- Standard HTML `form` tag attributes and values. The attributes and values are included in the `form` tag that `cfform` outputs in the page. For example, you can use `form` tag attributes like `target` or `onMouseOver` with `cfform`.
- HTML tags that can ordinarily be put within the HTML `form` tag. For example, you can use the HTML `input` tag to create a submit button in a `cfform`, without the other features of `cfinput`:

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

Examples

```html
<h3>cfform Example</h3>
<!--- If Form.oncethrough exists, the form has been submitted. --->
<cfif IsDefined("Form.oncethrough")>
  <cfif IsDefined("Form.testVal1")>
    <h3>Results of Radio Button Test</h3>
    <cfif Form.testVal1> Your radio button answer was yes </cfif>
    <cfelse> Your radio button answer was no </cfif>
  </cfif>
</cfif>

<h3>Results of Checkbox Test</h3>
<cfif IsDefined("Form.chkTest2")>
  Your checkbox answer was yes
<cfelse>
  Your checkbox answer was no
</cfif>

<cfif IsDefined("Form.textSample") AND Form.textSample is not "">
  <h3>Results of Credit Card Input</h3>
  Your credit card number, <cfoutput>#Form.textSample#</cfoutput>, was valid under the MOD 10 algorithm.
</cfif>

<cfif IsDefined("Form.sampleSlider")>
  <cfoutput>
    <h3>You gave this page a rating of #Form.sampleSlider#</h3>
  </cfoutput>
</cfif>

<hr noshade="True">
```

<!--- Begin by calling the cfform tag. --->
<cfform name="cfformexample">

<h4>This example displays radio button input type for cfinput.</h4>
Yes <cfinput type = "Radio" name = "TestVal1" value = "Yes" checked>
No <cfinput type = "Radio" name = "TestVal1" value = "No">

<h4>This example displays checkbox input type for cfinput.</h4>
<cfinput type = "Checkbox" name = "chkTest2" value = "Yes">

<h4>This shows client-side validation for cfinput text boxes.</h4>
<i>This item is optional</i>
```


158   Chapter 2: ColdFusion Tags
Please enter a credit card number:
<cfinput type = "Text" name = "TextSample"
        message = "Please enter a Credit Card Number"
        validate = "creditcard" required = "No">
<h4>This example shows the use of the cfslider tag.</h4>
Rate your approval of this example from 1 to 10 by sliding control.<br>
<cfslider name = "sampleSlider" width="100"
        label = "Page Value: " range = "1,10"
        message = "Please enter a value from 1 to 10"> 10
<p><cfinput type = "submit" name = "submit" value = "show me the result">
<cfinput type = "hidden" name = "oncethrough" value = "Yes"></p>
</cfform>

**A Simple XML form**

The following example shows a simple XML-format form. It uses the default.xsl transform that is supplied with ColdFusion to generate the HTML output for display:

<cfform name="testXForm" format="XML" skin="basic">
<!--- Use cfformgroup to put the first and last names on a single line. --->
<cfformgroup type="horizontal">
<cfinput type="text" name="firstname" label="First Name:" value="Robert">
<cfinput type="text" name="lastname" label="Last Name:" value="Smith">
</cfformgroup>
<cfinput type="password" name="password" label="Password:" value="">
<cfinput type="hidden" name="hidden" label="hidden:" value="">
<cfselect name="state" style="width:200" label="State">
<option>California</option>
<option selected>Utah</option>
<option>Iowa</option>
<option selected>New York</option>
</cfselect>
<cftextarea name="description" label="Description:" rows="5" cols="40">this is sample text.</cftextarea>
</cfform>
cformgroup

Description

Creates a container control for multiple form controls. Used in the cfform tag body of Macromedia Flash and XML forms. Ignored in HTML forms.

Category

Forms tags

Syntax
cfformgroup

- type = "group type"
- label = "label"
- style = "style specification"
- selectedIndex = "page number"
- width = "pixels"
- height = "pixels"
- enabled = "Yes" or "No"
- visible = "Yes" or "No"
- OnChange = "ActionScript expression"
- toolTip = "Tip text"

...ColdFusion forms controls
</cfformgroup>

or

cfformgroup

- type = "repeater"
- query = "query object"
- startrow = "row number"
- maxrows = "integer"

...ColdFusion forms controls
</cfformgroup>

See also


History

ColdFusion MX 7: Added this tag.

Attributes

The following table lists the attributes and their behavior in Flash forms. For XML, if not otherwise noted, the attribute is passed to the XML but is not interpreted by the basic XSL style sheet provided with ColdFusion MX.
**Note:** Attributes that are not marked as supported in XML are not handled by the skins provided with ColdFusion MX. They are, however, included in the generated XML.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Req/Opt: Formats</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
</table>
| type      | Required; Flash and XML | XML: Can be any XForms group type defined in the XSLT. The XSL skins provided with ColdFusion MX support the following types:  
  - horizontal: align child tags horizontally within a form and put this tag’s label attribute to the left of the children.  
  - vertical: align child tags vertically within a form and put this tag’s label attribute to the left of the children.  
  - fieldset: corresponds to the HTML fieldset tag, which groups its children, typically by drawing a box around them and replacing part of the top line with legend text. To specify the legend, use the label attribute. To specify the box dimensions, use the style attribute with height and width values. You must explicitly use cfformgroup type="vertical" inside this formgroup to align its child tags vertically.  
Flash: Must be one of the following:  
  - repeater: dynamically creates an instance of the cfformgroup's child tag or tags for each row of a query object, without requiring ColdFusion to recompile the Flash SWF file when the number of rows changes.  
  - horizontal: aligns child tags horizontally within a form and put this tag’s label attribute to the left of the children. Use this tag to arrange individual controls horizontally.  
  - vertical: aligns child tags vertically within a form and puts this tag’s label attribute to the left of the children. Use this tag to arrange individual controls vertically.  
  - hbox: aligns children horizontally. Use this type to arrange groups of controls horizontally. Do not use this attribute to align individual controls horizontally, because the child controls do not align properly; use the horizontal type instead.  
  - vbox: aligns children vertically. Use this type to arrange groups of controls vertically. Do not use this attribute to align individual controls vertically, as the child controls will not align properly; use the vertical type instead. |
• **hdividedbox**: aligns children horizontally. Each child is in a box with a border, and there are dividers between the boxes that users can move to change the relative sizes of the children. Use a tag with this attribute to arrange groups of form controls horizontally. You cannot use this attribute to align individual controls horizontally.

• **vdividedbox**: aligns children vertically. Each child is in a box with a border, and there are dividers between the boxes that users can move to change the relative sizes of the children. Use this type to group form controls, for example as a unit in an hbox form group. Do not use this attribute to align individual tags vertically.

• **panel**: a container consisting of a title bar containing the `label` attribute text, a border, and a content area with vertically arranged children.

• **tile**: places the children in a rectangular grid.

• **accordion**: places each child in a pleat of an expanding and contracting accordion. Define each pleat using a `cformgroup type="page"` tag.

• **tabnavigator**: places the children in a tabbed dialog box. Define each tab using a `cformgroup type="page"` tag.

• **page**: places the children tags, aligned vertically, in a single tab of a parent tabnavigator or pleat of an accordion container.

---

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Req/Opt: Formats</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>type</strong></td>
<td>(continued)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>query</td>
<td>Required for <code>type=repeaters</code></td>
<td></td>
<td>The query to use with the repeater. Flash creates an instance of each of the <code>cformgroup</code> tag’s child tags for each row in the query. You can use the <code>bind</code> attribute in the child tags to use data from the query row for the instance.</td>
</tr>
<tr>
<td>startrow</td>
<td>Optional; Flash</td>
<td>0</td>
<td>Used only for the <code>repeater</code> type; ignored otherwise. Specifies the row number of the first row of the query to use in the Flash form repeater. This attribute is zero-based: the first row is row 0, not row 1 (as in most ColdFusion tags).</td>
</tr>
<tr>
<td>maxrows</td>
<td>Optional; Flash</td>
<td></td>
<td>Used only for the <code>repeater</code> type; ignored otherwise. Specifies the maximum number of query rows to use in the Flash form repeater. If the query has more rows than the sum of the <code>startrow</code> attribute and this value, the repeater does not use the remaining rows.</td>
</tr>
<tr>
<td>Attribute</td>
<td>Req/Opt: Formats</td>
<td>Default</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>------------------</td>
<td>---------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>label</td>
<td>Optional; Flash and XML</td>
<td></td>
<td>Label to apply to the form group. In Flash, does the following:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• For a page or panel form group, determines the label to put on the</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>corresponding accordion pleat, the tabnavigator tab, or the panel title bar.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• For a Flash horizontal or vertical form group, specifies the label to</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>put to the left of the group.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Ignored in Flash for repeater, hbox, hdividedbox, vbox,</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>vdividedbox, tile, accordion, and tabnavigator types.</td>
</tr>
<tr>
<td>style</td>
<td>Optional; Flash and XML</td>
<td></td>
<td><strong>Flash</strong>: A Flash style specification in CSS format. For detailed</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>information on specifying Flash styles, see Chapter 29, &quot;Creating Forms in</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Macromedia Flash&quot; in ColdFusion MX Developer’s Guide.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><strong>XML</strong>: An inline CSS style specification.</td>
</tr>
<tr>
<td>selectedIndex</td>
<td>Optional; Flash only</td>
<td></td>
<td>Used only for accordion and tabnavigator types; ignored otherwise.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Specifies the page control to display as open, where 0 (not 1) specifies</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>the first page control defined in the group.</td>
</tr>
<tr>
<td>width</td>
<td>Optional; Flash and XML</td>
<td></td>
<td>Width of the group container, in pixels. If you omit this attribute, Flash</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>automatically sizes the container width. Ignored for Flash repeater type.</td>
</tr>
<tr>
<td>height</td>
<td>Optional; Flash</td>
<td></td>
<td>Height of the group container, in pixels. If you omit this attribute, Flash</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>automatically sizes the container height. Ignored for Flash repeater type.</td>
</tr>
<tr>
<td>enabled</td>
<td>Optional; Flash</td>
<td>Yes</td>
<td>Boolean value specifying whether the controls in the form group are enabled.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Disabled controls appear in light gray.</td>
</tr>
<tr>
<td>visible</td>
<td>Optional; Flash</td>
<td>Yes</td>
<td>Boolean value specifying whether the controls in the form group are visible.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>If the controls are invisible, the space that would be occupied by visible</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>controls is blank.</td>
</tr>
<tr>
<td>onChange</td>
<td>Optional; Flash</td>
<td></td>
<td>Tabnavigator and accordion types only: ActionScript expression or expressions</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>to execute when a new tab or accordion page is selected.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><strong>Note</strong>: The onChange event occurs when the form first appears.</td>
</tr>
<tr>
<td>tooltip</td>
<td>Optional; Flash</td>
<td></td>
<td>Text to display when the mouse pointer hovers in the form group area. If a</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>control in the form group also specifies a tooltip, Flash displays the</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>control’s tooltip when the mouse pointer hovers over the control.</td>
</tr>
</tbody>
</table>

**Usage**

This tag requires an end tag. This tag is ignored if the cfform type is HTML; any tag body’s contents are interpreted as if the surrounding cfformgroup does not exist.
In Flash format forms, this tag organizes the contents of the form. It groups and arranges child tags. The body of this tag can contain the following tags; all other tags and text are ignored:

- cfformgroup
- cfformitem
- cfcalendar
- cfgrid
- cfinput
- cfselect
- cftextarea
- cftree

For more information on using this tag in Flash forms, see Chapter 29, “Creating Forms in Macromedia Flash” in ColdFusion MX Developer’s Guide.

In XML format, ColdFusion passes the tag and its attributes to the XML; it is the responsibility of the skin XSLT to handle the XML. The ColdFusion basic skin supports the horizontal, vertical, and dualselectlist styles only. For more information on using this tag in XML forms, see Chapter 29, “Creating Forms in Macromedia Flash” in ColdFusion MX Developer’s Guide.

Example

For a simple example of an XML form that uses a single cfformgroup tag, see cfform.

The following example shows how to use the cfformgroup tag to arrange elements on a Flash form. It creates an hdividedbox container that has a vbox container on each side. The left box has heading text and two radio buttons. The right box has heading text and three check boxes.

```xml
<h3>Simple cfformgroup Example</h3>
<cfform name="myform" height="450" width="500" format="Flash">
  <cfformgroup type="hdividedbox">
    <cfformgroup type="vbox">
      <cfformitem type="text" height="20">
        Pets:
      </cfformitem>
      <cfinput type="Radio" name="pets" label="Dogs" value="Dogs" checked>
      <cfinput type="Radio" name="pets" label="Cats" value="Cats">
    </cfformgroup>
    <cfformgroup type="vbox">
      <cfformitem type="text" height="20">
        Fruits:
      </cfformitem>
      <cfinput type="Checkbox" name="chk1" label="Apples" value="Apples">
      <cfinput type="Checkbox" name="chk2" label="Bananas" value="Bananas">
      <cfinput type="Checkbox" name="chk3" label="Pears" value="Pears">
    </cfformgroup>
  </cfformgroup>
</cfform>
```

---

164  Chapter 2: ColdFusion Tags
The following more complex example shows more fully how you can use `cfformgroup` tags to arrange controls in a Flash form. It also shows many of the text formatting features that you can use in a text `cfformgroup` body. When you submit the form, the page dumps the contents of the Forms scope, to show you the submitted data.

```<h2>cfformgroup Example</h2>
<cfif IsDefined("form.oncethrough")>
  <h3>The form submitted the following information to ColdFusion MX:</h3>
  <cfdump var="#form#"></br><br>
</cfif>
<h3>A Flash form using cfformgroup tags</h3>
<cfform name="myform" height="450" width="500" format="Flash">
  <!--- The following formgroup shows how you can present formatted text. --->
  <cfformitem type="html">
    <b><font color="#FF0000" size="+4" face="serif">
      This form has two tabs, asking for the following:
    </font></b><br>
    <li>contact information</li>
    <li>preferences</li>
    <b>Try entering information on both tabs</b><br>
    Submit the form and see what ColdFusion gets in the Forms scope.<br>
    <a href="http://www.macromedia.com/" target="_blank">
      This link displays the Macromedia home page in a new browser window
    </a>

    <hr>
    &nbsp;<br>
  </cfformitem>
  <!--- Use a tabnavigator with two tabs for user input. --->
  <cfformgroup type="tabnavigator" height="220">
    <cfformgroup type="page" label="Contact Information">
      <!--- Align the first and last name fields horizontally --->
      <cfformgroup type="horizontal" label="Your Name">
        <cfinput type="text" required="Yes" name="firstName" label="First" value="" width="100"/>
        <cfinput type="text" required="Yes" name="lastName" label="Last" value="" width="100"/>
      </cfformgroup>
    </cfformgroup>
    <cfformitem type="html">Flash fills the email field in automatically.<br>
    You can replace any of the text.<br>
    </cfformitem>
  </cfformgroup>
  <!--- The bind attribute gets the field contents from the firstname
  and lastName fields as they get filled in. --->
  <cfinput type="text" name="email" label="email" bind="[firstName.text].(lastName.text)@mm.com">
  <cfinput type="text" name="phone" validate="telephone" required="Yes" label="Phone Number">
</cfformgroup>
```
<cfformgroup type="page" label="Preferences">
  <cfformitem type="text" height="30">
    <b>Tell us your preferences</b>
  </cfformitem>
  <!--- Put the pet selectors to the left of the fruit selectors. --->
  <cfformgroup type="hbox">
    <!--- Group the pet selector box contents, aligned vertically. --->
    <cfformgroup type="vbox">
      <cfformitem type="text" height="20">
        Pets:
      </cfformitem>
      <cfformgroup type="vertical">
        <cffinput type="Radio" name="pets" label="Dogs" value="Dogs" checked>
        <cffinput type="Radio" name="pets" label="Cats" value="Cats">
      </cfformgroup>
    </cfformgroup>
  </cfformgroup>
  <!--- Group the fruit selector box contents, aligned vertically. --->
  <cfformgroup type="vbox">
    <cfformitem type="text" height="20">
      Fruits:
    </cfformitem>
    <cfformgroup type="tile" width="200" label="Tile box">
      <!--- Flash requires unique names for all controls --->
      <cffinput type="Checkbox" name="chk1" Label="Apples" value="Apples">
      <cffinput type="Checkbox" name="chk2" Label="Bananas" value="Bananas">
      <cffinput type="Checkbox" name="chk3" Label="Pears" value="Pears">
      <cffinput type="Checkbox" name="chk4" Label="Oranges" value="Oranges">
      <cffinput type="Checkbox" name="chk5" Label="Grapes" value="Grapes">
      <cffinput type="Checkbox" name="chk6" Label="Cumquats" value="Cumquats">
    </cfformgroup>
  </cfformgroup>
</cfformgroup>
</cfformgroup>
</cfformgroup>
</cfformgroup>

<!--- Flash requires unique names for all controls --->
<cffinput type = "submit" name="submit" width="100" value = "Show Results">
<cffinput type = "reset" name="reset" width="100" value = "Reset Fields">
<cffinput type = "hidden" name="oncethrough" value = "Yes">
</cfformgroup>
</cfform>
**cformitem**

**Description**

Inserts a horizontal line, a vertical line, a spacer, or text in a Flash form. Used in the `cform` or `cformgroup` tag body for Flash and XML forms. Ignored in HTML forms.

**Category**

Forms tags

**Syntax**

```xml
<cfformitem
    type = "hrule, vrule, or spacer"
    style = "style specification"
    width = "pixels"
    height = "pixels"
    visible = "Yes" or "No"/>

or

<cfformitem
    type = "html or text"
    style = "style specification"
    width = "pixels"
    height = "pixels"
    visible = "Yes" or "No"
    enabled = "Yes" or "No"
    tooltip = "Tip text"
    bind = "bind expression">
    ...
    text
</cfformitem>
```

**See also**

`cfapplet`, `cform`, `cformgroup`, `cfgrid`, `cfinput`, `cfselect`, `cfslider`, `cftextarea`, `cftree`, “Adding text, images, rules, and space with the cformitem tag” in Chapter 29, “Adding text, images, rules, and space with the cformitem tag,” in ColdFusion MX Developer’s Guide

**History**

ColdFusion MX 7: Added tag

**Attributes**

The following table lists the attributes and their behavior in Flash forms. For XML format, if not otherwise noted, the attribute is passed to the XML but is not interpreted by the basic XSL style sheet provided with ColdFusion MX.
**Note:** Attributes that are marked as Flash only are not handled by the skins provided with ColdFusion MX. They are, however, included in the generated XML in all controls except text and html types.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Req/Opt; Formats</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
</table>
| type      | Required; Flash and XML | Flash: | - Place the text in the body of this tag on the form. For Flash forms, you can use the following text formatting tags, most of which correspond to HTML tags, in the text: `a`, `b`, `br`, `font`, `i`, `img`, `li`, `p`, `textformat`, and `u`. For details on using these formatting tags, see the Flash documentation. The `style` attribute has no effect on the format of the text in `type`.  
- Place the text in the body of this tag on the form verbatim, without interpreting any markup. You can control the overall appearance of the text by using the `style` attribute.  
- Places an invisible spacer of the specified height and width on the form. Used to place space between form controls. This tag must not have any children.  
- Places a horizontal rule on the form. This tag must not have any children.  
- Places a vertical rule on the form. This tag must not have any children.  
XML:  
- Puts the CFML tag’s body text in a CDATA section in an XML `xf:output` element.  
- Formats (escapes characters such as `i`) the CFML tag’s body text and puts it in a CDATA section in an XML `xf:output` element.  
- Puts an `hr` tag in the output. Use the `style` attribute to specify all rule characteristics, including height and width. This tag must not have any children.  
- Any other string generates an XML `xf:group` element with the type name as the `appearance` attribute. The CFML tag body is put in a CDATA section in a `cf:attribute name="body"` element. The XSL transforms provided with ColdFusion MX ignore these elements. |
| style     | Optional; Flash and XML | Flash: | Must be a style specification in CSS format. Ignored if the `type` attribute is `html` or `text`. For detailed information on specifying Flash styles, see Chapter 29, “Creating Forms in Macromedia Flash” in ColdFusion MX Developer’s Guide. Not used with the `spacer` type.  
XML:  
ColdFusion passes the `style` attribute to the XML. ColdFusion skins include the `style` attribute in the generated HTML. |
### Attribute Table

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Req/Opt; Formats</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>width</td>
<td>Optional; Flash</td>
<td></td>
<td>Width of the item, in pixels. If you omit this attribute, Flash automatically sizes the width. In ColdFusion XSL skins, use the style attribute, instead.</td>
</tr>
<tr>
<td>height</td>
<td>Optional; Flash</td>
<td></td>
<td>Height of the item, in pixels. If you omit this attribute, Flash automatically sizes the width. In ColdFusion XSL skins, use the style attribute, instead.</td>
</tr>
<tr>
<td>enabled</td>
<td>Optional; Flash</td>
<td>Yes</td>
<td>Boolean value specifying whether the control is enabled. Disabled text appear in light gray. Has no effect on spacers and rules.</td>
</tr>
<tr>
<td>visible</td>
<td>Optional; Flash</td>
<td>Yes</td>
<td>Boolean value specifying whether to show the control. Space that would be occupied by an invisible control is blank. Has no effect on spacers.</td>
</tr>
<tr>
<td>tooltip</td>
<td>Optional; Flash</td>
<td></td>
<td>Text to display when the mouse pointer hovers over the control. Has no effect on spacers.</td>
</tr>
<tr>
<td>bind</td>
<td>Optional; Flash</td>
<td></td>
<td>A Flash bind expression that populates the field with information from other form fields. If you use this attribute, ColdFusion MX ignores any text that you specify in the body of the cftextitem tag. This attribute can be useful if the cfformitem tag is in a cfformgroup type=&quot;repeater&quot; tag. See Usage. For details, see &quot;Flash form data binding&quot; in the cfinput tag description.</td>
</tr>
</tbody>
</table>

**Note:** Attributes that are marked as Flash only are not handled by the skins provided with ColdFusion MX. They are, however, included in the generated XML in all controls except text and html types.

### Usage

This tag requires an end tag or a slash before the closing end character of the opening tag, as the following example shows:

```xml
<cfformitem type="hrule" />
```

For more information on using this tag in Flash forms, see Chapter 29, “Creating Forms in Macromedia Flash” in ColdFusion MX Developer’s Guide.

### Example

The following example shows a simple Flash form using horizontal rules and text:

```xml
<h3>cfformitem Example</h3>
<cfform name="myform" height="450" width="500" format="Flash" >
  <cfformitem type="hrule" />
  <cfformitem type="text">
    This simple form has two hrule cfformitem tags around the cfformitem tag that contains this text.
  </cfformitem>
  <cfformitem type="hrule" />
</cfform>
```

For a more complex form, see cfformgroup.
cftp

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

Category
File management tags, Internet Protocol tags

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

- “cftp: Opening and closing FTP server connections” on page 171
- “cftp: Connection: File and directory operations” on page 174
- “cftp action = "listDir"” on page 178

See also
cfhttp, cfldap, cfmail, cfpop; “Performing file operations with cftp” in Chapter 40,
“Interacting with Remote Servers,” in ColdFusion MX Developer’s Guide

History
ColdFusion MX 7: Added the result attribute for file and directory operations.

ColdFusion MX: Deprecated the agentname attribute. It might not work, and might cause an error, in later releases.

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

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

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

Security settings
ColdFusion MX security settings can prevent the cftp tag from executing. If you run
ColdFusion applications on a server that is used by multiple customers, consider the security of
the files that the customer can move. For more information, see the “Administering Security”
section of Configuring and Administering ColdFusion MX.
cftp: Opening and closing FTP server connections

Description

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

Syntax

```cftp
action = "action"
username = "name"
password = "password"
server = "server"
timeout = "timeout in seconds"
port = "port"
connection = "name"
proxyServer = "proxy server"
retryCount = "number"
stopOnError = "yes" or "no"
passive = "yes" or "no"
```

See also

cfhttp, cfldap, cfmail, cfpop

Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Req/Opt</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>action</td>
<td>Required</td>
<td></td>
<td>FTP operation to perform.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• open: creates an FTP connection</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• close: terminates an FTP connection</td>
</tr>
<tr>
<td>username</td>
<td>Required</td>
<td></td>
<td>User name to pass in the FTP operation.</td>
</tr>
<tr>
<td>password</td>
<td>Required</td>
<td></td>
<td>Password to log in the user.</td>
</tr>
<tr>
<td>server</td>
<td>Required</td>
<td></td>
<td>FTP server to which to connect; for example,</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><code>ftp.myserver.com</code></td>
</tr>
<tr>
<td>timeout</td>
<td>Optional</td>
<td>30</td>
<td>Value in seconds for the timeout of all operations, including</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>individual data request operations.</td>
</tr>
<tr>
<td>port</td>
<td>Optional</td>
<td>21</td>
<td>Remote port to which to connect.</td>
</tr>
<tr>
<td>connection</td>
<td>Optional, but always used with open or close</td>
<td></td>
<td>Name of the FTP connection. If you specify the <code>username</code>, <code>password</code>, and <code>server</code> attributes, and if no connection exists for them, ColdFusion creates one. Calls to <code>cftp</code> with the same connection name reuse the connection.</td>
</tr>
<tr>
<td>proxyServer</td>
<td>Optional</td>
<td></td>
<td>String. Name of proxy server (or servers) to use, if proxy access is specified.</td>
</tr>
</tbody>
</table>
When you establish a connection with `cfftp action="open"` and specify a name in the `connection` attribute, ColdFusion caches the connection so that you can reuse it to perform additional FTP operations. When you use a cached connection for subsequent FTP operations, you do not have to specify the `username`, `password`, or `server` connection attributes. The FTP operations that use the same `connection` name automatically use the information stored in the cached connection. Using a cached connection helps save connection time and improves file transfer performance.

You do not need to open a connection for single, simple, FTP operations, such as `GetFile` or `PutFile`.

To keep a connection open throughout a session or longer, put the connection name in the `Session` or `Application` scope; for example, specify `connection="Session.FTPConnection"`. However, if you do this, you must specify the full variable name in all FTP operations, and you must use the `close` action when you are finished. Keeping a connection open prevents others from using the FTP server; so close a connection as soon as possible. If you do not assign the connection name to `Session` or `Application` variable, the connection remains open for the current page only, and you do not have to close it manually.

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

### Example

```xml
<p>cfftp lets users implement File Transfer Protocol operations. By default, cfftp caches an open connection to an FTP server. cfftp operations are usually of two types: </p>
<ul>
  <li>Establishing a connection</li>
  <li>Performing file and directory operations</li>
</ul>
<p>This example opens and verifies a connection, lists the files in a directory, and closes the connection. </p>
<p>Open a connection</p>
<p>cfftp action = "open"
</p>
```

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Req/Opt</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>retryCount</td>
<td>Optional</td>
<td>1</td>
<td>Number of retries until failure is reported.</td>
</tr>
<tr>
<td>stopOnError</td>
<td>Optional</td>
<td>no</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• yes: halts processing, displays an appropriate error.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• no: populates these variables:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• <code>cfftp.succeeded</code> - yes or no.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• <code>cfftp.errorText</code> - Message text. For conditional operations, use <code>cfftp.errorCode</code>. Do not use <code>cfftp.errorText</code> for this purpose.</td>
</tr>
<tr>
<td>passive</td>
<td>Optional</td>
<td>no</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• yes: enables passive mode.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• no</td>
</tr>
</tbody>
</table>

Usage

When you establish a connection with `cfftp action="open"` and specify a name in the `connection` attribute, ColdFusion caches the connection so that you can reuse it to perform additional FTP operations. When you use a cached connection for subsequent FTP operations, you do not have to specify the `username`, `password`, or `server` connection attributes. The FTP operations that use the same `connection` name automatically use the information stored in the cached connection. Using a cached connection helps save connection time and improves file transfer performance.

You do not need to open a connection for single, simple, FTP operations, such as `GetFile` or `PutFile`.

To keep a connection open throughout a session or longer, put the connection name in the `Session` or `Application` scope; for example, specify `connection="Session.FTPConnection"`. However, if you do this, you must specify the full variable name in all FTP operations, and you must use the `close` action when you are finished. Keeping a connection open prevents others from using the FTP server; so close a connection as soon as possible. If you do not assign the connection name to `Session` or `Application` variable, the connection remains open for the current page only, and you do not have to close it manually.

Changes to a cached connection, such as changing `retryCount` or `timeout` values, might require reestablishing the connection.
username = "anonymous"
connection = "My_query"
password = "youremail@email.com"
server = "ftp.tucows.com"
stopOnError = "Yes">
<p>Did it succeed? <cfoutput>#cfftp.succeeded#</cfoutput>

<p>List the files in a directory:
<cfftp action = "LISTDIR"
    stopOnError = "Yes"
    name = "ListFiles"
    directory = "/"
    connection = "my_query">
<cfoutput query = "ListFiles">
 #name#<br>
</cfoutput>
</cfftp>

<p>Close the connection:
<cfftp action = "close"
    connection = "My_query"
    stopOnError = "Yes">
<p>Did it succeed? <cfoutput>#cfftp.succeeded#</cfoutput>
**cfftp: Connection: File and directory operations**

**Description**

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

**Syntax**

```
<cfftp
    action = "action"
    username = "name"
    password = "password"
    name = "query_name"
    server = "server"
    ASCIIExtensionList = "extensions"
    transferMode = "mode"
    failIfExists = "yes" or "no"
    directory = "directory name"
    localFile = "filename"
    remoteFile = "filename"
    item = "directory or file"
    existing = "file or directory name"
    new = "file or directory name"
    proxyServer = "proxy server"
    passive = "yes" or "no"
>
    result = "result_name"
</cfftp>
```

**See also**

`cfhttp`, `cfldap`, `cfmail`, `cfpop`

**Attributes**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Req/Opt</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
</table>
| action      | Required if connection is not cached | FTP operation to perform:  
• changedir  
• createDir  
• listDir  
• removeDir  
• getFile  
• putFile  
• rename  
• remove  
• getCurrentDir  
• getCurrentURL  
• existsDir  
• existsFile  
• exists | |
| username    | Required if connection is not cached | User name to pass in the FTP operation. |
| password    | Required if action = "open" | Password to log in the user. |
**Attribute** | **Req/Opt** | **Default** | **Description**  
--- | --- | --- | ---  
name | Required if `action` = "listDir" |  | Query name of directory listing.  
server | Required if FTP connection is not cached | FTP server to which to connect; for example, ftp.myserver.com. |  
ASCIIExtension List | Optional | txt;htm;html; cfm;cfml; shtm;shhtml; css;asp;asa | Delimited list of file extensions that force ASCII transfer mode, if `transferMode` = "auto".  
transferMode | Optional | Auto | • ASCII FTP transfer mode  
null | Binary FTP transfer mode  
null | Auto FTP transfer mode  
failIfExists | Optional | yes | • yes: if a local file with same name exists, the `getFile` action will fail.  
null | • no  
directory | Required if `action` = "changedir", "createDir", "listDir", or "existsDir" | Directory on which to perform an operation.  
localFile | Required if `action` = "getFile" or "putFile" | Name of the file on the local file system.  
remoteFile | Required if `action` = "getFile", "putFile", or "existsFile" | Name of the file on the FTP server file system.  
item | Required if `action` = "exists" or "remove" | Object of these actions: file or directory.  
existing | Required if `action` = "rename" | Current name of the file or directory on the remote server.  
new | Required if `action` = "rename" | New name of file or directory on the remote server.  
proxyServer | Optional | String. Name of the proxy server (s) to use, if proxy access is specified.  
passive | Optional | no | • yes: enables passive mode.  
null | • no  
result | Optional | Specifies a name for the structure in which cfftp stores the `returnValue` variable. If set, this value replaces `cfftp` as the prefix to use when accessing `returnVariable`. See the Usage section for more information.  

*cfftp: Connection: File and directory operations*
Usage

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

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

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

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

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

- getCurrentDir
- getCurrentURL
- existsDir
- existsFile
- exists

For more information, see ColdFusion MX Developer’s Guide.

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

Action (cfftp.ReturnValue variable)

The results of an action determine the value of the returnValue variable, as the following table shows:

<table>
<thead>
<tr>
<th>cfftp action</th>
<th>Value of cfftp.returnValue</th>
</tr>
</thead>
<tbody>
<tr>
<td>getCurrentDir</td>
<td>String. Current directory.</td>
</tr>
<tr>
<td>getCurrentURL</td>
<td>String. Current URL.</td>
</tr>
<tr>
<td>existsDir</td>
<td>yes or no.</td>
</tr>
<tr>
<td>existsFile</td>
<td>yes or no.</td>
</tr>
<tr>
<td>exists</td>
<td>yes or no.</td>
</tr>
</tbody>
</table>

To access the returnValue variable, you must prefix it with either cfftp or the value specified by the result attribute, if it is set. The result attribute provides a way for cfftp calls from multiple pages, possibly at the same time, to avoid overwriting the results of one with another. If you set the result attribute to myResult, for example, you would access the returnVariable variable as myResult.returnValue. Otherwise, you would access it as cfftp.returnValue.
Example

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

```cupdate connection = "myConnection"
    username = "myUserName"
    password = "myUserName@allaire.com"
    server = "ftp.allaire.com"
    action = "open"
    stopOnError = "Yes"
</cupdate>
<p>Did it succeed?  <cfoutput>#cfftp.succeeded#</cfoutput>
</p>
<cupdate connection = "myConnection"
    action = "LISTDIR"
    stopOnError = "Yes"
    name = "ListDirs"
    directory = "/">
</cupdate>
</p>
<cfoutput>
FTP Directory Listing:<br>
<cftable query = "ListDirs" HTMLTable = "Yes" colHeaders = "Yes">
    <cfcol header = "<b>Name</b>" text = "#name#">
    <cfcol header = "<b>Path</b>" text = "#path#">
    <cfcol header = "<b>URL</b>" text = "#url#">
    <cfcol header = "<b>Length</b>" text = "#length#">
    <cfcol header = "<b>LastModified</b>" text = "#DateFormat(lastmodified)#">
    <cfcol header = "<b>IsDirectory</b>" text = "#isdirectory#">
</cftable>
</p>
<p>Move Image File to Remote Server:<br></p>
<!--- The image will be put into the root directory of the FTP server unless otherwise noted. 
1.e. remoteFile = "somewhere_put.jpg" vs remoteFile = "/support/somewhere_put.jpg" --->
<cupdate connection = "myConnection"
    action = "putFile"
    name = "uploadFile"
    transferMode = "binary"
    localFile = "C:\files\upload\somewhere.jpg"
    remoteFile = "somewhere_put.jpg"
</cupdate>
<p>Did it succeed?  <cfoutput>#cfftp.succeeded#</cfoutput>
</p>
<p>Close the connection:
<cupdate connection = "myConnection"
    action = "close"
    stopOnError = "Yes"
</cupdate>
<p>Did it succeed?  <cfoutput>#cfftp.succeeded#</cfoutput>
</p>
```
**cfftp action = "listDir"**

**Description**

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

**Usage**

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

<table>
<thead>
<tr>
<th>cfftp query object column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Filename of the current element.</td>
</tr>
<tr>
<td>Path</td>
<td>File path (without drive designation) of the current element.</td>
</tr>
<tr>
<td>URL</td>
<td>Complete URL for the current element (file or directory).</td>
</tr>
<tr>
<td>Length</td>
<td>File size of the current element.</td>
</tr>
<tr>
<td>LastModified</td>
<td>Unformatted date/time value of the current element.</td>
</tr>
<tr>
<td>Attributes</td>
<td>String. Attributes of the current element: normal or Directory.</td>
</tr>
<tr>
<td>IsDirectory</td>
<td>Boolean. Whether object is a file or directory.</td>
</tr>
<tr>
<td>Mode</td>
<td>Applies only to UNIX and Linux. Permissions. Octal string.</td>
</tr>
</tbody>
</table>

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

Description
Defines a function that you can call in CFML. Required to define ColdFusion component methods.

History
ColdFusion MX 7: Added the XML value to the returnType attribute.
ColdFusion MX: Added this tag.

Category
Extensibility tags

Syntax
<cffunction
  name = "methodName"
  returnType = "dataType"
  roles = "securityRoles"
  access = "methodAccess"
  description = "function description"
  output = "yes" or "no"
  displayName = "name"
  Hint = "hint text">

See also
 cfargument, cfcomponent, cfinvoke, cfinvokeargument, cfobject, cfproperty, cfreturn
## Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Req/Opt</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Required</td>
<td>A string; a component method that is used within the <code>cfcomponent</code> tag.</td>
<td></td>
</tr>
</tbody>
</table>
| returnType | Required for a web service; Optional, otherwise. | any String; a type name; data type of the function return value:  
• any  
• array  
• binary  
• boolean  
• date  
• guid - The argument must be a UUID or GUID of the form `xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx` where each x is a character representing a hexadecimal number (0-9A-F).  
• numeric  
• query  
• string  
• struct  
• uuid: the argument must be a ColdFusion UUID of the form `xxxxxxxx-xxxx-xxxx-xxxxxxxxxxxxxxxx` where each x is a character representing a hexadecimal number (0-9A-F).  
• variableName: a string formatted according to ColdFusion variable naming conventions.  
• void: does not return a value  
• xml: allows web service functions to return CFML XML objects and XML strings.  
• a component name: if the type attribute value is not one of the preceding items, ColdFusion treats it as the name of a ColdFusion component. When the function executes, it generates an error if the argument that is passed in is not a CFC with the specified name. |
| roles     | Optional | "" (empty) | A comma-delimited list of ColdFusion security roles that can invoke the method. Only users who are logged in with the specified roles can execute the function. If this attribute is omitted, all users can invoke the method. |
The \texttt{cffunction} tag can define a function that you call in the same manner as a ColdFusion built-in function.

To define a ColdFusion component (CFC) method, you must use a \texttt{cffunction} tag.

The following example shows \texttt{cffunction} tag attributes for a simple CFC method that returns a ColdFusion Query object.

\begin{verbatim}
<cffunction
    name="getEmployees"
    access="remote"
    returnType="query"
    hint="This query returns all records in the employee database. It can drill-down or narrow the search, based on optional input parameters."
>
\end{verbatim}
For information on using the `cffunction` tag for ColdFusion components, see Chapter 10, “Building and Using ColdFusion Components” in *ColdFusion MX Developer’s Guide*.

If you specify a `roles` attribute, the function executes only if a user is logged in and belongs to one of the specified roles.

If you specify `variableName` for the `returnType` attribute, the function must return a string that is in ColdFusion variable name format; that is, the function must return a string that starts with a letter, underscore, or Unicode currency symbol, and consist of letters, numbers, and underscores (_), periods, and Unicode currency symbols, only. ColdFusion does not check whether the value corresponds to an existing ColdFusion variable.

**Example**

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

**Description**

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

Displays data graphically.

**History**

ColdFusion MX: Deprecated this tag. It works differently than it did in ColdFusion 5, and it might not work in later releases.

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

<table>
<thead>
<tr>
<th>cfgraph tag attribute</th>
<th>ColdFusion MX functionality</th>
</tr>
</thead>
<tbody>
<tr>
<td>Title</td>
<td>Ignored.</td>
</tr>
<tr>
<td>Titlefont</td>
<td>Ignored.</td>
</tr>
<tr>
<td>Barspacing</td>
<td>Ignored.</td>
</tr>
<tr>
<td>Bordercolor</td>
<td>Color used for border, gridlines, and text displays.</td>
</tr>
<tr>
<td>Colorlist</td>
<td>List of colors to use for each data point for bar, pyramid, area, horizontalbar, cone, cylinder, step, and pie charts.</td>
</tr>
<tr>
<td>Valuelabelfont</td>
<td>Sets value label text font. If the <code>Valuelabelfont</code>, <code>Itemlabelfont</code>, and <code>Legendfont</code> values differ, ColdFusion uses the last value that you specify in the tag. Arial is not supported; it is mapped to Dialog.</td>
</tr>
<tr>
<td>Itemlabelfont</td>
<td>Sets item label text font. If the <code>Valuelabelfont</code>, <code>Itemlabelfont</code>, and <code>Legendfont</code> values differ, ColdFusion uses the last value that you specify in the tag. Arial is not supported; it is mapped to Dialog.</td>
</tr>
<tr>
<td>Legendfont</td>
<td>Sets legend text font. If the <code>Valuelabelfont</code>, <code>Itemlabelfont</code>, and <code>Legendfont</code> values differ, ColdFusion uses the last value that you specify in the tag. Arial is not supported; it is mapped to Dialog.</td>
</tr>
</tbody>
</table>
| ShowLegend           | • above, below, left, right: these options cause the legend to display, but have no effect on its location.  
• none: prevents display of a legend. |
| Valuelabelsize       | Sets value label text size. If the `Valuelabelsize` and `Itemlabelsize` values differ, ColdFusion uses the last value that you specify in the tag. |
| Itemlabelsize        | Sets item label text size. |
| Itemlabelorientation | Ignored. ColdFusion calculates best orientation based on label and graph size. |
| Borderwidth          | • a nonzero number: default-width border, regardless of number value.  
• 0: no border. |
<table>
<thead>
<tr>
<th>cfgraph tag attribute</th>
<th>ColdFusion MX functionality</th>
</tr>
</thead>
</table>
| Depth                  | • 0: displays graph with two-dimensional appearance.  
                           • any other value: displays graph with three-dimensional appearance. |
| Linewidth              | Ignored.                  |
| Showvaluelabel         | • yes: displays values on mouse-click.  
                           • no: suppresses value displays.  
                           • rollover: displays values on mouse-over. |
| Valuelocation          | Ignored.                  |
| url                    | URL of page to open if any item in the graph is clicked.  
The following variables may be used within the URL; they are substituted with real values before the URL is accessed:  
• "$value$": selected row/column value or an empty string.  
• "$itemlabel$": selected item (column) value or an empty string.  
• "$serieslabel$": selected series (row) value or an empty string.  
• "javascript:...": executes client side scripts. |
| Urlcolumn              | Ignored.                  |
| Type="HorizontalBar"  | The (0,0) coordinate is located at the lower-left. |
| ScaleFrom              | If the smallest value in the data is less than scaleFrom or the largest value in the data is greater than scaleTo, the respective data value is used as the minimum or maximum on the Y scale. Therefore, regardless of the scaleFrom or scaleTo value, all data values display. |
**cfgraphdata**

**Description**

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

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

**History**

ColdFusion MX: Deprecated this tag. It works differently than in ColdFusion 5 and might not work in later releases.
cfgrid

Description

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

Category

Forms tags

Syntax

```xml
<cfgrid
    name = "name"
    format = "applet" or "Flash" or "xml"
    height = "integer"
    width = "integer"
    query = "query_name"
    selectMode = "mode"
    insert = "yes" or "no"
    delete = "yes" or "no"
    font = "column_font"
    fontSize = "size"
    italic = "yes" or "no"
    bold = "yes" or "no"
    textColor = "web color"
    gridLines = "yes" or "no"
    rowsHeight = "pixels"
    colHeaders = "yes" or "no"
    colHeaderFont = "font_name"
    colHeaderFontSize = "size"
    colHeaderItalic = "yes" or "no"
    colHeaderBold = "yes" or "no"
    colHeaderTextColor = "web color"
    bgColor = "web color"
    maxRows = "number"
    style = "style specification"
    enabled = "Yes" or "No"
    visible = "Yes" or "No"
    toolTip = "Tip text"
    onChange = "ActionScript"
    autoWidth = "yes" or "no"
    vSpace = "integer"
    hSpace = "integer"
    align = "value"
    sort = "yes" or "no"
    href = "URL"
    hrefKey = "column_name"
    target = "URL_target"
    appendKey = "yes" or "no"
    highlightHref = "yes" or "no"
```
onValidate = "javascript_function"
onError = "text"
gridDataAlign = "position"
rowHeaders = "yes" or "no"
rowHeaderAlign = "position"
rowHeaderFont = "font_name"
rowHeaderFontSize = "size"
rowHeaderItalic = "yes" or "no"
rowHeaderBold = "yes" or "no"
rowHeaderTextColor = "web color"
colHeaderAlign = "position"
selectColor = "web color"
notSupported = "text"
pictureBar = "yes" or "no"
insertButton = "text"
deleteButton = "text"
sortAscendingButton = "text"
sortDescendingButton = "text"

zero or more cfgridcolumn and cfgridrow tags
</cfgrid>

See also
cfapplet, cfcalendar, cfgridcolumn, cfgridrow, cfgridupdate, cfform, cfformgroup, cfformitem, cfgrid, cfinput, cfselect, cfslider, cftextarea, cftree

History

ColdFusion MX 7:

- Added the format attribute and support for Flash and XML output.
- Added enabled, onChange, style, tooltip, and visible attributes (Flash format only).

ColdFusion MX: Changed the rowHeaderWidth attribute: ColdFusion does not use the rowHeaderWidth attribute. You can omit it.
Attributes

*Note:* In XML format, ColdFusion MX passes all attributes to the XML. The supplied XSLT skins do not handle or display XML format grids, but do display applet and Flash format grids.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Req/Opt: Formats</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Required; All</td>
<td></td>
<td>Name of the grid control.</td>
</tr>
</tbody>
</table>
| format    | Optional; All   | applet  | • applet: generates a Java applet.  
|           |                 |         | • Flash: generates a Flash grid control.  
|           |                 |         | • xml: generates an XML representation of the grid.  
|           |                 |         | In XML format forms, includes the generated XML in the form.  
|           |                 |         | In HTML format forms, puts the XML in a string variable with the name specified by the name attribute. |
| height    | Optional; All   | 300 (applet only) | Height of the control, in pixels.  
|           |                 |         | If you omit the attribute in Flash format, the grid sizes automatically. |
| width     | Optional; All   | 300 (applet only) | Width of the control, in pixels.  
|           |                 |         | If you omit the attribute in Flash format, the grid sizes automatically. |
| query     | Optional; All   |         | Name of the query associated with the control. |
| selectMode| Optional; All   | Applet  | Selection mode for items in the control.  
|           |                 | format:  | • Edit: user can edit grid data. Selecting a cell opens the editor for the cell type.  
|           |                 | Browse;  | • Row: user selections automatically extend to the row that contains selected cell.  
|           |                 | Flash   | The following are used in applet format only. Flash interprets these as Row:  
|           |                 | format:  | • Single: user selections are limited to selected cell.  
|           |                 | Row     | • Column: user selections automatically extend to column that contains selected cell.  
|           |                 |         | • Browse: user can only browse grid data. |
| font      | Optional; All   |         | Font of text. |
| fontSize  | Optional; All   |         | Size of text, in points. |
| italic    | Optional; All   | no      | • yes: displays text in italics.  
|           |                 |         | • no |
| bold      | Optional; All   | no      | • yes: displays text in bold.  
<p>|           |                 |         | • no |</p>
<table>
<thead>
<tr>
<th>Attribute</th>
<th>Req/Opt; Formats</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>textColor</td>
<td>Optional; All</td>
<td>Black</td>
<td>Color of text. Can be a hexadecimal value or a named color.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>For a hexadecimal value, use the form &quot;##xxxxxx&quot;, where x = 0-9 or A-F; use two</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>number signs or none.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>For a list of the supported named colors, see cfchart.</td>
</tr>
<tr>
<td>selectColor</td>
<td>Optional; All</td>
<td></td>
<td>Background color for a selected item.</td>
</tr>
<tr>
<td>gridLines</td>
<td>Optional; All</td>
<td>yes</td>
<td>• yes: enables row and column rules.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• no</td>
</tr>
<tr>
<td>rowHeight</td>
<td>Optional; All</td>
<td>yes</td>
<td>Minimum row height, in pixels. Used with cfgridcolumn type = &quot;Image&quot;; defines</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>space for graphics to display in row.</td>
</tr>
<tr>
<td>colHeaders</td>
<td>Optional; All</td>
<td>yes</td>
<td>• yes: displays column headers.</td>
</tr>
<tr>
<td>colHeaderFont</td>
<td>Optional; All</td>
<td></td>
<td>• no</td>
</tr>
<tr>
<td>colHeaderFontSize</td>
<td>Optional; All</td>
<td></td>
<td>Font of column header.</td>
</tr>
<tr>
<td>colHeaderItalic</td>
<td>Optional; All</td>
<td>no</td>
<td>• yes: displays column headers in italics.</td>
</tr>
<tr>
<td>colHeaderBold</td>
<td>Optional; All</td>
<td>no</td>
<td>• yes: displays column headers in bold.</td>
</tr>
<tr>
<td>colHeaderTextColor</td>
<td>Optional; All</td>
<td></td>
<td>• no</td>
</tr>
<tr>
<td>bgColor</td>
<td>Optional; All</td>
<td></td>
<td>Color of column headers.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Options: same as for textColor attribute.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Options: for applet format, same as for textColor attribute; for Flash format,</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>must be a hexadecimal value.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Flash format only; to specify background colors for alternating rows,</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>separate the two colors with a comma.</td>
</tr>
<tr>
<td>maxRows</td>
<td>Optional; All</td>
<td></td>
<td>Maximum number of rows to display in the grid.</td>
</tr>
<tr>
<td>style</td>
<td>Optional; Flash</td>
<td></td>
<td>Must be a style specification in CSS format.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Ignored for type=&quot;text&quot;.</td>
</tr>
<tr>
<td>enabled</td>
<td>Optional; Flash</td>
<td>Yes</td>
<td>Flash format only. Boolean value specifying whether the control is enabled.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>A disabled control appears in light gray.</td>
</tr>
<tr>
<td>Attribute</td>
<td>Req/Opt; Formats</td>
<td>Default</td>
<td>Description</td>
</tr>
<tr>
<td>-----------</td>
<td>-----------------</td>
<td>---------</td>
<td>-------------</td>
</tr>
<tr>
<td>visible</td>
<td>Optional; Flash</td>
<td>Yes</td>
<td>Flash format only: Boolean value specifying whether to show the control. Space that would be occupied by an invisible control is blank.</td>
</tr>
<tr>
<td>tooltip</td>
<td>Optional; Flash</td>
<td></td>
<td>Flash format only: text to display when the mouse pointer hovers over the control.</td>
</tr>
<tr>
<td>onChange</td>
<td>Optional; Flash</td>
<td></td>
<td>ActionScript to run when the control changes due to user action in the control.</td>
</tr>
<tr>
<td>autoWidth</td>
<td>Optional; applet</td>
<td>no</td>
<td>• yes: sets column widths so that all columns display within the grid width. Widths are equal or the proportions are determined by the relative cfgridcolumnwidth attribute values. Horizontal scroll bars are not available. • no: sets columns to equal widths or the values specified in the cfgridcolumnwidth attributes.</td>
</tr>
<tr>
<td>vSpace</td>
<td>Optional; applet</td>
<td></td>
<td>Vertical space above and below the control, in pixels.</td>
</tr>
<tr>
<td>hSpace</td>
<td>Optional; applet</td>
<td></td>
<td>Horizontal space to the left and right of the control, in pixels.</td>
</tr>
<tr>
<td>align</td>
<td>Optional; applet</td>
<td></td>
<td>Alignment of the grid cell contents: • Top • Left • Bottom • Baseline • Texttop • Absbottom • Middle • Absmiddle • Right</td>
</tr>
<tr>
<td>insert</td>
<td>Optional; applet</td>
<td>no</td>
<td>• yes: users can insert row data in the grid; takes effect only if selectmode=&quot;edit&quot;. • no</td>
</tr>
<tr>
<td>delete</td>
<td>Optional; applet</td>
<td>no</td>
<td>• yes: users can delete row data from the grid; takes effect only if selectmode=&quot;edit&quot;. • no</td>
</tr>
<tr>
<td>sort</td>
<td>Optional; applet</td>
<td>no</td>
<td>Adds sort buttons to perform simple text sorts on a user-selected column: • yes: put sort buttons on the grid control. • no Independent of this setting, users can sort columns by clicking the column head. If selectMode=&quot;browse&quot;, the table cannot be sorted.</td>
</tr>
<tr>
<td>Attribute</td>
<td>Req/Opt; Formats</td>
<td>Default</td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>-----------------</td>
<td>---------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>href</td>
<td>Optional; applet</td>
<td></td>
<td>URL or name of a query column that contains URLs to hyperlink each grid cell with.</td>
</tr>
<tr>
<td>target</td>
<td>Optional; applet</td>
<td></td>
<td>The target frame or window in which to display the href URL; for example, &quot;_blank&quot;.</td>
</tr>
<tr>
<td>appendKey</td>
<td>Optional; applet</td>
<td>yes</td>
<td>• yes: when used with href, appends &quot;CFGRIDKEY=&quot; and information about the selected items. For details see &quot;Using the href attribute&quot;.</td>
</tr>
<tr>
<td>hrefKey</td>
<td>Optional; applet</td>
<td></td>
<td>A query column to use for the value appended to the href URL of each cell, if appendKey=&quot;True&quot;. If you use cfgridcolumn tags, the column must be specified in one of these tags.</td>
</tr>
<tr>
<td>highlightHref</td>
<td>Optional; applet</td>
<td>yes</td>
<td>• yes: highlights links associated with an href attribute value.</td>
</tr>
<tr>
<td>onValidate</td>
<td>Optional; applet</td>
<td></td>
<td>A JavaScript function to validate user input. The form object, input object, and input object value are passed to the function, which must return True if validation succeeds; False otherwise.</td>
</tr>
<tr>
<td>onError</td>
<td>Optional; applet</td>
<td></td>
<td>A JavaScript function to execute if validation fails.</td>
</tr>
<tr>
<td>gridDataAlign</td>
<td>Optional; applet</td>
<td>Left</td>
<td>• Left: left-aligns data within the column.</td>
</tr>
<tr>
<td>rowHeaders</td>
<td>Optional; applet</td>
<td>yes</td>
<td>• yes: displays a column of numeric row labels.</td>
</tr>
<tr>
<td>rowHeaderAlign</td>
<td>Optional; applet</td>
<td>Left</td>
<td>• Left: left-aligns the row header text.</td>
</tr>
<tr>
<td>rowHeaderFont</td>
<td>Optional; applet</td>
<td></td>
<td>Font for the row labels.</td>
</tr>
<tr>
<td>rowHeaderFontSize</td>
<td>Optional; applet</td>
<td></td>
<td>Text size of the row labels, in points.</td>
</tr>
<tr>
<td>rowHeaderItalic</td>
<td>Optional; applet</td>
<td>no</td>
<td>• yes: displays row label text in italics.</td>
</tr>
<tr>
<td>rowHeaderBold</td>
<td>Optional; applet</td>
<td>no</td>
<td>• yes: displays row label text in bold.</td>
</tr>
<tr>
<td>rowHeaderTextColor</td>
<td>Optional; applet</td>
<td>Black</td>
<td>• Options: same as for the textColor attribute.</td>
</tr>
</tbody>
</table>
Usage

This tag must be in a **cfform** tag block.

An applet format grid requires the client to download a Java applet. Also, if the client does not have an up-to-date Java plugin installed, the system might also have to download an updated Java plugin to display the an applet format grid. A Flash format grid generates a Flash control, and can be embedded in an HTML format **cfform** tag. For this tag to work properly in either Flash or applet format, the browser must also be JavaScript-enabled.

**Note:** If you specify Flash format for this tag in an HTML format form, and you do not specify **height** and **width** attributes, Flash takes up more than the remaining visible area on the screen. If any other output follows the grid, including any form controls, users must scroll to see it. Therefore, if you follow a Flash grid in an HTML format form with additional output, specify **height** and **width** values.

You can populate a **cfgrid** with data from a **cfquery**. If you do not specify any **cfgridcolumn** tags in the **cfgrid** body, ColdFusion generates a grid with the following:

- A column for each column in the query.
- A default header for each column, created by replacing hyphen or underscore characters in the table column name with spaces. The first character, and any character after a space, are changed to uppercase; all other characters are lowercase.

This tag requires an end tag.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Req/Opt; Formats</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>colHeaderAlign</td>
<td>Optional; applet</td>
<td>Left</td>
<td>• Left: left-aligns the column header text.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Right: right-aligns the column header text.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Center: centers the column header text.</td>
</tr>
</tbody>
</table>
| notSupported        | Optional; applet | (See Description) | Text to display if the browser does not support Java or has Java support disabled. Default: "&lt;b&gt; Browser must support Java to view ColdFusion Java Applets&lt;/b&gt;"
| pictureBar          | Optional; applet | no      | • yes: puts images (and no text) on the Insert, Delete, and Sort buttons.   |
|                     |                  |         | • no: puts text (and no images) on the Insert, Delete, and Sort buttons.    |
| insertButton        | Optional; applet | Insert  | Insert button text; takes effect only if selectmode="edit".                |
| deleteButton        | Optional; applet | Delete  | Delete button text; takes effect only if selectmode="edit".                |
| sortAscendingButton | Optional; applet | A -> Z  | Sort button text.                                                           |
| sortDescendingButton| Optional; applet | Z -> A  | Sort button text.                                                           |
Note: Clicking the submit button while editing a grid cell occasionally causes the cell changes to be lost. To ensure that changes are submitted properly, Macromedia recommends that after user updates data in a cell, they click another cell before submitting the form.

How data is returned from cfgrid

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

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

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

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

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

form.#GridName#.ColumnName#

Each SelectMode returns these form variable(s):

• SelectMode="single"
  form.#GridName#.ColumnName# = "SelectedCellValue"
• SelectMode="column"
  form.#GridName#.ColumnName# = "ValueOfCellRow1, ValueOfCellRow2, ValueOfCellRowN"
• SelectMode="row"
  form.#GridName#.ColumnName# = "ValueOfCellInSelectedRow"
  form.#GridName#.ColumnName# = "ValueOfCellInSelectedRow"
  form.#GridName#.ColumnName# = "ValueOfCellInSelectedRow"

Complex update data (SelectMode = Edit)

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

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

form.#GridName#.ColumnName#
form.#GridName#.original.ColumnName#
form.#GridName#.RowStatus.Action
Each table row that contains an update, insert, or deletion has a parallel entry in each of these arrays. To view all the information for all the changes, you can traverse the arrays, as in this example. To make it work with a cfgrid on a submitted cfform, set the GridName variable to the name of the grid and the ColNameList to a list of the grid columns.

```cfs```
<cfloop index="ColName" list="#ColNameList#">
  <cfif IsDefined("form.#GridName#.#ColName#")>
    <cfoutput>
      form.#GridName#.#ColName#:<br>
      <cfset Array_New = form[#GridName#][#ColName#]>
      <cfset Array_Orig = form[#GridName#]['original'][#ColName#]>
      <cfset Array_Action = form[#GridName#]RowStatus.Action>

      <cfif NOT IsArray(Array_New)>
        <b>The form variable is not an array!</b><br>
      <cfelse>
        <cfset size = ArrayLen(Array_New)>
        <cfoutput>
          Result Array Size is #size#. <br>
          Contents:<br>
        </cfoutput>
      </cfif>
      <cfif size IS 0>
        <b>The array is empty.</b><br>
      <cfelse>
        <table BORDER="yes">
          <tr>
            <th>Loop Index</TH>
            <th>Action</TH>
            <th>Old Value</TH>
            <th>New Value</TH>
          </tr>
          <cfloop index="LoopCount" from="1" to=#size#>
            <cfset Val_Orig= Array_Orig[#LoopCount#]>
            <cfset Val_New = Array_New[#LoopCount#]>
            <cfset Val_Action= Array_Action[#LoopCount#]>
            <cfoutput>
              <tr>
                <td>#LoopCount#</td>
                <td>#Val_Action#</td>
                <td>#Val_Orig#</td>
                <td>#Val_New#</td>
              </tr>
            </cfoutput>
          </cfloop>
        </table>
      </cfif>
    </cfif>
  </cfoutput>
</cfif>
</cfloop>
```cfs```
Using the href attribute

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

http://myserver.com?cfgridkey=selection

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

The value of selection is determined by the value of the selectMode and attribute:

- If you specify a hrefKey attribute, selection is the field value of the column specified by the attribute. Otherwise, it is one of the following:
  - If selectMode="Single", selection is the value of the column clicked.
  - If selectMode="Row", selection is a comma-delimited list of column values in the clicked row, beginning with the value of the first cell in the row.
  - If selectMode="Column", selection is a comma-delimited list of row values in the clicked column, beginning with the value of the first cell in the column.

Example

The following example creates a Flash form that displays a set of available courses from the CourseList table in the cfdocexamples database. For more complex examples that use the cfgrid tag, see cfgridcolumn, cfgridrow, and cfgridupdate.

<!--- Query the database to fill up the grid. --->
<cfquery name = "GetCourses" dataSource = "cfdocexamples">
SELECT Course_ID, Dept_ID, CorNumber, CorName, CorLevel
FROM CourseList
ORDER by Dept_ID ASC, CorNumber ASC
</cfquery>

<h3>cfgrid Example</h3>
<i>Currently available courses</i>
<!--- cfgrid must be inside a cfform tag. --->
<cfform>
  <cfgrid name = "FirstGrid" format="Flash" height="320" width="580"
    font="Tahoma" fontsize="12"
    query = "GetCourses">
  </cfgrid>
</cfform>
cfgridcolumn

Description
Used with the \texttt{cfgrid} tag in a \texttt{cfform}. Formats a column and optionally populates the column from a query. The font and alignment attributes used in \texttt{cfgridcolumn} override global font or alignment settings defined in \texttt{cfgrid}.

Category
Forms tags

Syntax
\begin{verbatim}
<cfgridcolumn
    name = "column_name"
    header = "header"
    width = "column_width"
    type = "type"
    display = "yes" or "no"
    select = "yes" or "no"
    font = "column_font"
    fontSize = "size"
    italic = "yes" or "no"
    bold = "yes" or "no"
    textColor = "web color" or "expression"
    bgColor = "web color" or "expression"
    dataAlign = "position"
    The following attribute applies to Flash format only
    mask = "format mask"
    The following attribute applies to applet format only
    href = "URL"
    hrefKey = "column_name"
    target = "URL_target"
    headerFont = "font_name"
    headerFontSize = "size"
    headerItalic = "yes" or "no"
    headerBold = "yes" or "no"
    headerTextColor = "web color"
    headerAlign = "position"
    numberFormat = "format"
    values = "Comma-separated strings and/or numeric range"
    valuesDisplay = "Comma-separated strings and/or numeric range"
    valuesDelimiter = "delimiter character">
\end{verbatim}

See also
\texttt{cfgrid, cfgridrow, cfgridupdate, cfform, cfapplet, cfinput, cfselect, cfslider, cftextarea, cftree}

History
ColdFusion MX 7: Added the \texttt{mask} attribute, and the currency \texttt{type} attribute value.
ColdFusion MX: Changed behavior if select="no": a user cannot select and edit the cell data, regardless of the cfgrid selectmode attribute value. When clicked, the cell border (and, depending on the selectColor value, the cell background) changes color, but the cell data cannot be edited.

Attributes

**Note:** In XML format, ColdFusion MX passes all attributes to the XML. The supplied XSLT skins do not handle or display XML format grids, but do display applet and Flash format grids.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Req/Opt; Formats</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Required; All</td>
<td></td>
<td>Name of the grid column element. If the grid uses a query, this attribute must be the name of the query column that populates the grid column.</td>
</tr>
<tr>
<td>header</td>
<td>Optional; All</td>
<td>yes</td>
<td>Column header text. Used only if the cfgrid colHeaders attribute is True (the default).</td>
</tr>
<tr>
<td>width</td>
<td>Optional; All</td>
<td>Column head width</td>
<td>Column width, in pixels.</td>
</tr>
<tr>
<td>Attribute</td>
<td>Req/Opt; Formats</td>
<td>Default</td>
<td>Description</td>
</tr>
<tr>
<td>-----------</td>
<td>------------------</td>
<td>-------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>type</td>
<td>Optional; All</td>
<td></td>
<td>You can specify the following values in all formats:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• image: grid displays the image specified by the URL in the column. If you</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>use a relative URL, the image must be in the CFIDE\classes directory or a</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>subdirectory. If image is larger than column cell, it is clipped to fit.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Flash images must be JPEG files. Applet images can be JPEG or GIF files.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• boolean: column displays as check box; if cell is editable, user can</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>change the check mark.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• numeric: user can sort grid data numerically.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• string_noCase: user can sort grid data as case-insensitive text.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>You can specify the following attribute in applet format; it does not work</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>in Flash grids.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• image: you can use the following built-in ColdFusion image names, in</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>addition to paths to image files, in the column values:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- cd</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- computer</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- document</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- element</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- folder</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- floppy</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- fixed</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- remote</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>You can specify the following attribute in Flash format; it does not work</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>in applet grids.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• currency: formats the column data as currency, aligning it around the</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>decimal point. If users sort the grid using this column, it will sort</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>correctly for the currency. Use the mask attribute to specify a currency</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>symbol, which defaults to the dollar sign ($).</td>
</tr>
<tr>
<td>display</td>
<td>Optional; All</td>
<td>yes</td>
<td>• yes</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• no: hides the column.</td>
</tr>
<tr>
<td>select</td>
<td>Optional; All</td>
<td>yes</td>
<td>Determines selection behavior if the cfgrid selectmode attribute value is</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>column, edit, or single; ignored for row or browse values.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• yes: users can select the column or select or edit cells in the column,</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>as specified by the selectmode attribute.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• no: users cannot select the column or select or edit cells in the column.</td>
</tr>
<tr>
<td>font</td>
<td>Optional; All</td>
<td>As specified by cfgrid</td>
<td>Font of data in column.</td>
</tr>
<tr>
<td>Attribute</td>
<td>Req/Opt; Formats</td>
<td>Default</td>
<td>Description</td>
</tr>
<tr>
<td>-------------</td>
<td>------------------</td>
<td>---------</td>
<td>-------------</td>
</tr>
<tr>
<td>fontSize</td>
<td>Optional; All</td>
<td>As specified by cfgrid</td>
<td>Size of text in column.</td>
</tr>
</tbody>
</table>
| italic      | Optional; All     | As specified by cfgrid | • yes: displays grid control text in italics.  
|             |                   |         | • no         |
| bold        | Optional; All     | As specified by cfgrid | • yes: displays grid control text in bold.  
|             |                   |         | • no         |
| textColor   | Optional; All     | All     | Color of grid element text in column as a hexadecimal number or text name.  
|             |                   |         | To enter a hexadecimal value, use the form "##xxxxxx", where x = 0-9 or A-F; use two number signs or none.  
|             |                   |         | • Any color, in hexadecimal format  
|             |                   |         | • Black  
|             |                   |         | • Red  
|             |                   |         | • Blue  
|             |                   |         | • Magenta  
|             |                   |         | • Cyan  
|             |                   |         | • Orange  
|             |                   |         | • Darkgray  
|             |                   |         | • Pink  
|             |                   |         | • Gray  
|             |                   |         | • White  
|             |                   |         | • Lightgray  
|             |                   |         | • Yellow  
| bgColor     | Optional; All     | All     | Color of background of grid column.  
|             |                   |         | • Options: same as for the textColor attribute.  
| dataAlign   | Optional; All     | as specified by cfgrid | Column data alignment:  
|             |                   |         | • Left  
|             |                   |         | • Right  
<p>|             |                   |         | • Center  |</p>
<table>
<thead>
<tr>
<th>Attribute</th>
<th>Req/Opt; Formats</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mask</td>
<td>optional; Flash</td>
<td>A mask pattern that controls the character pattern that the form displays or allows users to input and sends to ColdFusion. For columns with the currency type attribute, the mask specifies the currency symbol. ColdFusion MX automatically inserts the character before the numeric value. For columns with text or numeric values, mask specifies the format to display or allow users to input, as follows: • A = [A-Za-z] • X = [A-Za-z0-9] • 9 = [0-9] • ? = Any character • All other characters = ColdFusion inserts the literal character. If the column values are dates or timestamps, ColdFusion uses the mask pattern to format the selected date. For details of the date/time mask format, see date/time formats in mask attribute.</td>
<td></td>
</tr>
<tr>
<td>href</td>
<td>Optional; Applet</td>
<td>URL or query column name that contains a URL to hyperlink each grid column with.</td>
<td></td>
</tr>
<tr>
<td>hrefKey</td>
<td>Optional; Applet</td>
<td>The query column to use for the value appended to the href URL of each column, instead of the column's value.</td>
<td></td>
</tr>
<tr>
<td>target</td>
<td>Optional; Applet</td>
<td>Frame in which to open link specified in href.</td>
<td></td>
</tr>
<tr>
<td>headerFont</td>
<td>Optional; Applet</td>
<td>as specified by cfgrid</td>
<td>Column header font.</td>
</tr>
<tr>
<td>headerFontSize</td>
<td>Optional; Applet</td>
<td>as specified by cfgrid</td>
<td>Column header text size, in pixels.</td>
</tr>
<tr>
<td>headerItalic</td>
<td>Optional; Applet</td>
<td>as specified by cfgrid</td>
<td>• yes: displays column header in italics. • no</td>
</tr>
<tr>
<td>headerBold</td>
<td>Optional; Applet</td>
<td>as specified by cfgrid</td>
<td>• yes: displays header in bold. • no</td>
</tr>
<tr>
<td>headerTextColor</td>
<td>Optional; Applet</td>
<td>as specified by cfgrid</td>
<td>Color of grid control column header text. • Options: same as for the textColor attribute.</td>
</tr>
<tr>
<td>headerAlign</td>
<td>Optional; Applet</td>
<td>as specified by cfgrid</td>
<td>Column header text alignment: • Left • Right • Center</td>
</tr>
</tbody>
</table>
In applet format only, you can use the following `numberFormat` attribute mask characters to format output in U.S. numeric and currency styles. For more information on using these mask characters, see `NumberFormat` on page 783. (The `cfgridcolumn` tag does not support international number formatting.)

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

By default, Flash displays date/time values in grid columns using a format that shows values such as Oct 29 2004 11:03:21. Use the mask attribute to display the date or time in a different format, as described in the following table:

<table>
<thead>
<tr>
<th>Pattern letter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Y</td>
<td>Year. If the number of pattern letters is two, the year is truncated to two digits; otherwise, it appears as four digits. The year can be zero-padded, as the third example shows in the following set of examples:</td>
</tr>
<tr>
<td></td>
<td>Examples:</td>
</tr>
<tr>
<td></td>
<td>YY = 03</td>
</tr>
<tr>
<td></td>
<td>YYYY = 2003</td>
</tr>
<tr>
<td></td>
<td>YYYYY = 02003</td>
</tr>
<tr>
<td>M</td>
<td>Month in year. The format depends on the following criteria:</td>
</tr>
<tr>
<td></td>
<td>• If the number of pattern letters is one, the format is interpreted as numeric in one or two digits.</td>
</tr>
<tr>
<td></td>
<td>• If the number of pattern letters is two, the format is interpreted as numeric in two digits.</td>
</tr>
<tr>
<td></td>
<td>• If the number of pattern letters is three, the format is interpreted as short text.</td>
</tr>
<tr>
<td></td>
<td>• If the number of pattern letters is four, the format is interpreted as full text.</td>
</tr>
<tr>
<td></td>
<td>Examples:</td>
</tr>
<tr>
<td></td>
<td>M = 7</td>
</tr>
<tr>
<td></td>
<td>MM = 07</td>
</tr>
<tr>
<td></td>
<td>MMM = Jul</td>
</tr>
<tr>
<td></td>
<td>MMMM = July</td>
</tr>
<tr>
<td>D</td>
<td>Day in month.</td>
</tr>
<tr>
<td></td>
<td>Examples:</td>
</tr>
<tr>
<td></td>
<td>D = 4</td>
</tr>
<tr>
<td></td>
<td>DD = 04</td>
</tr>
<tr>
<td></td>
<td>DD = 10</td>
</tr>
<tr>
<td>E</td>
<td>Day in week. The format depends on the following criteria:</td>
</tr>
<tr>
<td></td>
<td>• If the number of pattern letters is one, the format is interpreted as numeric in one or two digits.</td>
</tr>
<tr>
<td></td>
<td>• If the number of pattern letters is two, the format is interpreted as numeric in two digits.</td>
</tr>
<tr>
<td></td>
<td>• If the number of pattern letters is three, the format is interpreted as short text.</td>
</tr>
<tr>
<td></td>
<td>• If the number of pattern letters is four, the format is interpreted as full text.</td>
</tr>
<tr>
<td></td>
<td>Examples:</td>
</tr>
<tr>
<td></td>
<td>E = 1</td>
</tr>
<tr>
<td></td>
<td>EE = 01</td>
</tr>
<tr>
<td></td>
<td>EEE = Mon</td>
</tr>
<tr>
<td></td>
<td>EEEE = Monday</td>
</tr>
<tr>
<td>A</td>
<td>AM/PM indicator.</td>
</tr>
<tr>
<td>J</td>
<td>Hour in day (0-23).</td>
</tr>
<tr>
<td>H</td>
<td>Hour in day (1-24).</td>
</tr>
<tr>
<td>K</td>
<td>Hour in am/pm (0-11).</td>
</tr>
</tbody>
</table>
Example

The following example lets you update certain fields of the CourseList table in the cfdocexamples database. It uses cfgridcolumn tags to structure the table.

<!--- If the gridEntered field exists, the form has been submitted. Update the database. --->
<cfif IsDefined("form.gridEntered")>
<cfgridupdate grid = "FirstGrid" dataSource = "cfdocexamples"
    tableName = "CourseList" keyOnly = "Yes">
</cfif>

<!--- Query the database to fill up the grid. --->
<cfquery name = "GetCourses" dataSource = "cfdocexamples">
    SELECT Course_ID, Dept_ID, CorNumber,
    CorName, CorLevel, CorDesc
    FROM CourseList
    ORDER by Dept_ID ASC, CorNumber ASC
</cfquery>

<html>
<head>
<title>cfgrid Example</title>
</head>
<body>
<h3>cfgrid Example</h3>
</body>
</html>

<!--- You can update the Name, Level, and Description information for courses.<!--->
<!--- The cfform tag must surround a cfgrid control. --->
<cfform action = "#CGI.SCRIPT_NAME#">
    <cfgrid name = "FirstGrid" width = "500"
        query = "GetCourses" colheaderbold="Yes"
        font = "Tahoma" rowHeaders = "No"
        selectColor = "Red" selectMode = "Edit" >
        <!--- cfgridcolumn tags arrange the table and control the display. --->
        <!--- Hide the primary key, required for update --->
        <cfgridcolumn name = "Course_ID" display = "No" />
        <!--- select="No" does not seem to have any effect !!! --->
        <cfgridcolumn name = "Dept_ID" header = "Department" Select="No"
            width="75" textcolor="blue" bold="Yes">

Pattern letter | Description
------------- | -------------
L             | Hour in am/pm (1-12).
N             | Minute in hour.
Examples:
N = 3
NN = 03
S             | Second in minute.
Other text | You can add other text into the pattern string to further format the string. You can use punctuation, numbers, and all lowercase letters. You should avoid uppercase letters because they may be interpreted as pattern letters.
Example:
EEEE, MMM. D, YYYY at H:NN A = Tuesday, Sept. 8, 2003 at 1:26 PM
<cfgridcolumn name = "CorNumber" header = "Course #" Select="No" width="65">
<cfgridcolumn name = "CorName" header = "Name" width="125">
<cfgridcolumn name = "CorLevel" header = "Level" width="85">
<cfgridcolumn name = "CorDesc" header = "Description" width="125">
</cfgrid>
<br>
<cfinput type="submit" name="gridEntered">
</cfform>
</body>
</html>
**cfgridrow**

**Description**

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

**Category**

Forms tags

**Syntax**

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

**See also**

`cfgrid`, `cfgridcolumn`, `cfgridupdate`, `cfform`, `cfinput`, `cfselect`, `cfslider`, `cftextarea`, `cftree`

**Attributes**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Req/Opt</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>data</td>
<td>Required</td>
<td></td>
<td>Comma-delimited list of column values. If a value contains a comma, it must be escaped with another comma.</td>
</tr>
</tbody>
</table>

**Example**

The following example shows how you use the `cfgridrow` tag can populate a `cfgrid` tag from list data:

```xml
<!--- Set two lists, each with the data for a grid column. --->
<cfset cities = "Rome,Athens,Canberra,Brasilia,Paris">
<cfset countries = "Italy,Greece,Australia,Brazil,France">

<!--- Form definition --->
<cfform name = "cities">
  <cfgrid name="GeoGrid" autowidth = "yes" vspace = "4"
    height = "120" font="tahoma" rowheaders="no">
    <cfgridcolumn name="City" header="City">
    <cfgridcolumn name="Country" header="Country">
      <!--- Loop through the lists using cfgridrow to populate the grid. --->
      <cfloop index="i" from="1" to="#ListLen(cities)#">
        <cfgridrow data="#ListGetAt(cities, i)#,#ListGetAt(countries, i)#">
      </cfloop>
    </cfgridcolumn>
  </cfgrid>
</cfform>
```

---

cfgridrow 205
cfgridupdate

Description

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

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

Category

Forms tags

Syntax

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

See also

cfgrid, cfgridcolumn, cfgridrow, cfform, cfapplet, cfinput, cfselect, cfslider, cftextinput, cftree

History

ColdFusion MX: Deprecated the connectString, dbName, dbServer, dbtype, provider, and providerDSN attributes. They do not work, and might cause an error, in releases later than ColdFusion 5.

Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Req/Opt</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>grid</td>
<td>Required</td>
<td></td>
<td>Name of the cfgrid form element that is the source for the update action.</td>
</tr>
<tr>
<td>dataSource</td>
<td>Required</td>
<td></td>
<td>Name of the data source for the update action.</td>
</tr>
<tr>
<td>tableName</td>
<td>Required</td>
<td></td>
<td>Name of the table to update. For ORACLE drivers, entry must be upper-case. For Sybase driver, entry is case-sensitive; must be same case as used when table was created.</td>
</tr>
<tr>
<td>username</td>
<td>Optional</td>
<td></td>
<td>Overrides username value specified in ODBC setup.</td>
</tr>
<tr>
<td>password</td>
<td>Optional</td>
<td></td>
<td>Overrides password value specified in ODBC setup.</td>
</tr>
<tr>
<td>tableOwner</td>
<td>Optional</td>
<td></td>
<td>Table owner, if supported.</td>
</tr>
</tbody>
</table>
Example

The following example lets you update a database by using a cfgrid tag to add and delete entire records or to update the data in individual cells. The cfgridupdate tag processes the data from the submitted form and updates the database.

<!--- If the gridEntered form field exists, the form was submitted. --->
<cfif IsDefined("form.gridEntered") is True>
  <cfgridupdate grid = "FirstGrid" dataSource = "cfdocexamples" Keyonly="true"
    tableName = "CourseList">
  </cfif>

<!--- Query the database to fill up the grid. --->
<cfquery name = "GetCourses" dataSource = "cfdocexamples">
  SELECT Course_ID, Dept_ID, CorNumber,
         CorName, CorLevel, CorDesc
  FROM CourseList
  ORDER by Dept_ID ASC, CorNumber ASC
</cfquery>

<h3>cfgrid Example</h3>
Try adding a course to the database, and then deleting it.

<cfform>
  <cfgrid name = "FirstGrid" width = "450"
    query = "GetCourses" insert = "Yes" delete = "Yes"
    font = "Tahoma" rowHeaders = "No"
    colHeaderBold = "Yes"
    selectMode = "EDIT"
    insertButton = "Insert a Row" deleteButton = "Delete selected row" >
  </cfgrid>
  <cfinput type="submit" name="gridEntered">
</cfform>...
cfheader

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

Category
Data output tags, Page processing tags

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

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

History
ColdFusion MX 6.1: Changed behavior for the name attribute: cfheader name="Content-Disposition" uses the default file character encoding to encode this header's value, so the name of a file can include characters in the character encoding used in the file.

Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Req/Opt</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Required if statusCode not specified</td>
<td></td>
<td>Header name.</td>
</tr>
<tr>
<td>value</td>
<td>Optional</td>
<td></td>
<td>HTTP header value.</td>
</tr>
<tr>
<td>charset</td>
<td>Optional</td>
<td>UTF-8</td>
<td>The character encoding in which to encode the header value. The following list includes commonly used values:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• utf-8</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• iso-8859-1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• windows-1252</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• us-ascii</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• shift_jis</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• iso-2022;jp</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• euc-jp</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• euc-kr</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• big5</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• euc-cn</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• utf-16</td>
</tr>
</tbody>
</table>

For more information about character encodings, see www.w3.org/International/O-charset.html.
Usage

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

Example

<h3>cfheader Example</h3>

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

Description

Writes text to the head section of a generated HTML page.

Category

Page processing tags

Syntax

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

See also

cfcache, cfflush, cfheader, cfheader, cfinclude, cfsetting, cfslilent

Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Req/Opt</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>text</td>
<td>Required</td>
<td></td>
<td>Text to add to the &lt;head&gt; area of an HTML page.</td>
</tr>
</tbody>
</table>

Usage

Use this tag for embedding JavaScript code, or putting other HTML tags, such as meta, link, title, or base in an HTML page header.

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

Example

```html
<!--- This example displays the information provided by the Macromedia Designer & Developer Center XML feed, http://www.macromedia.com/devnet/resources/macromedia_resources.rdf
See http://www.macromedia.com/desdev/articles/xml_resource_feed.html
for more information on this feed. --->

<!--- Set the URL address. --->
<cfset urlAddress="http://www.macromedia.com/devnet/resources/macromedia_resources.rdf">

<!--- Use the CFHTTP tag to get the file content represented by urladdress
Note that />, not an end tag. terminates this tag. --->
<cfhttp url="#urladdress#" method="GET" resolveurl="Yes" throwOnError="Yes"/>

<!--- Parse the XML and output a list of resources. --->
<cfset xmlDoc = XmlParse(CFHTTP.FileContent)>

<!--- Get the array of resource elements, the xmlChildren of the xmlroot. --->
<cfset resources=xmlDoc.xmlroot.item>
<cfset numresources=ArrayLen(xmlDoc.xmlRoot.xmlChildren)-1>
<cfloop index="i" from="1" to="#numresources#">
  <cfset item=resources[i]>
  <cfoutput>
    <strong><a href="#item.link.xmltext#"#item.title.xmltext#, Author</a></strong>&nbsp;&nbsp;#item.creator.xmltext#
  </cfoutput>
```
<strong>Description</strong> #item.description.xmlText#<br>
<strong>Applies to these products</strong><br>
<cfloop index="i" from="1" to="#arrayLen(item.subject)#" step="1"> #item.subject[i].xmltext#<br>
</cfloop>
<br>
</cfoutput>
</cfloop>
cfhttp

Description
Generates an HTTP request and handles the response from the server.

Category
Internet Protocol tags

Syntax
<cfhttp
  url = "server_URL"
  port = "port_number"
  method = "method_name"
  proxyServer = "hostname"
  proxyPort = "port_number"
  proxyUser = "username"
  proxyPassword = "password"
  username = "username"
  password = "password"
  userAgent = "user_agent"
  charset = "character encoding"
  resolveURL = "yes" or "no"
  throwOnError = "yes" or "no"
  redirect = "yes" or "no"
  timeout = "timeout_period"
  getasbinary = "yes" or "no"
  multipart = "yes" or "no"
  path = "path"
  file = "filename"
  name = "queryname"
  columns = "query_columns"
  firstrowasheaders = "yes" or "no"
  delimiter = "character"
  textQualifier = "character"
  result = "result_name"
  cfhttpparam tags [optional for some methods]
</cfhttp>

See also
cfhttpparam. GetHttpRequestData. cfftp. cfldap. cfmail. cfpop. SetEncoding

History
ColdFusion MX 7: Added the result attribute, which allows you to specify an alternate variable in which to receive a result.

ColdFusion MX 6.1:
• Added support for the following methods: HEAD, PUT, DELETE, OPTIONS, TRACE.
• Added multipart, getAsBinary, proxyUser, and proxyPassword attributes.
• Changed httpparam behavior: all operations can have httpparam tags.
• Added the cfhttp.errorDetail return variable.
• Modified response body content types considered to be text.
• Changed behavior for multiple headers: multiple headers of the same type are returned in an array.
• Added support for HTTPS proxy tunneling.
• Fixed bugs in code and documentation.

ColdFusion MX:
• Added the `charset` and `firstrowasheaders` attributes.
• Changed Secure Sockets Layer (SSL) support: ColdFusion uses the Sun JSSE library, which supports 128-bit encryption, to support SSL.

Attributes
The following attributes control the HTTP transaction and can be used for all HTTP methods:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Req/Opt</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>url</td>
<td>Req</td>
<td>Uses the http protocol</td>
<td>Address of the resource on the server which will handle the request. The URL must include the hostname or IP address. If you do not specify the transaction protocol (http:// or https://), ColdFusion defaults to http. If you specify a port number in this attribute, it overrides any <code>port</code> attribute value. The <code>cfhttpparam</code> URL attribute appends query string attribute-value pairs to the URL.</td>
</tr>
<tr>
<td>port</td>
<td>Opt</td>
<td>80 for http 443 for https</td>
<td>Port number on the server to which to send the request. A port value in the <code>url</code> attribute overrides this value.</td>
</tr>
</tbody>
</table>
### HTTP Methods

- **GET**: requests information from the server. Any data that the server requires to identify the requested information must be in the URL or in `cfhttp type="URL"` tags.
- **POST**: sends information to the server for processing. Requires one or more `cfhttpparam` tags. Often used for submitting form-like data.
- **PUT**: requests the server to store the message body at the specified URL. Use this method to send files to the server.
- **DELETE**: requests the server to delete the specified URL.
- **HEAD**: identical to the GET method, but the server does not send a message body in the response. Use this method for testing hypertext links for validity and accessibility, determining the type or modification time of a document, or determining the type of server.
- **TRACE**: requests that the server echo the received HTTP headers back to the sender in the response body. Trace requests cannot have bodies. This method enables the ColdFusion application to see what is being received at the server, and use that data for testing or diagnostic information.
- **OPTIONS**: a request for information about the communication options available for the server or the specified URL. This method enables the ColdFusion application to determine the options and requirements associated with a URL, or the capabilities of a server, without requesting any additional activity by the server.

### Proxy Server Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Req/Opt</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>proxyServer</td>
<td>Opt</td>
<td></td>
<td>Host name or IP address of a proxy server to which to send the request.</td>
</tr>
<tr>
<td>proxyPort</td>
<td>Opt</td>
<td>80</td>
<td>Port number to use on the proxy server.</td>
</tr>
<tr>
<td>proxyUser</td>
<td>Opt</td>
<td></td>
<td>User name to provide to the proxy server.</td>
</tr>
<tr>
<td>proxyPassword</td>
<td>Opt</td>
<td></td>
<td>Password to provide to the proxy server.</td>
</tr>
<tr>
<td>username</td>
<td>Opt</td>
<td></td>
<td>Use to pass a user name to the target URL for Basic Authentication. Combined with <code>password</code> to form a base64 encoded string that is passed in the Authenticate header. Does not provide support for Integrated Windows, NTLM, or Kerebos authentication.</td>
</tr>
<tr>
<td>password</td>
<td>Opt</td>
<td></td>
<td>Use to pass a password to the target URL for Basic Authentication. Combined with <code>username</code> to form a base64 encoded string that is passed in the Authenticate header. Does not provide support for Integrated Windows, NTLM, or Kerebos authentication.</td>
</tr>
<tr>
<td>userAgent</td>
<td>Opt</td>
<td>ColdFusion</td>
<td>Text to put in the user agent request header. Used to identify the request client software. Can make the ColdFusion application appear to be a browser.</td>
</tr>
</tbody>
</table>
The character encoding of the request, including the URL query string and form or file data, and the response. The following list includes commonly used values:
- utf-8
- iso-8859-1
- windows-1252
- us-ascii
- shift_jis
- iso-2022-jp
- euc-jp
- euc-kr
- big5
- euc-cn
- utf-16

For more information character encodings, see www.w3.org/International/O-charset.html.

- no: does not resolve URLs in the response body. As a result, any relative URL links in the response body do not work.
- yes: resolves URLs in the response body to absolute URLs, including the port number, so that links in a retrieved page remain functional. Applies to these HTML tags:
  - img src
  - a href
  - form action
  - applet code
  - script src
  - embed src
  - embed pluginspace
  - body background
  - frame src
  - bgsound src
  - object data
  - object classid
  - object codebase
  - object usemap

Does not resolve URLs if the file and path attributes are used.

- yes: if the server returns an error response code, throws an exception that can be caught using the cftry and cfcatch or ColdFusion error pages.
- no: does not throw an exception if an error response is returned. In this case, your application can use the cfhttp.StatusCode variable to determine if there was an error and its cause.
<table>
<thead>
<tr>
<th>Attribute</th>
<th>Req/Opt</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
</table>
| redirect   | Opt     | yes     | If the response header includes a Location field AND ColdFusion receives a 300-series (redirection) status code, specifies whether to redirect execution to the URL specified in the field:  
• yes: redirects execution to the specified page.  
• no: stops execution and returns the response information in the cfhttp variable, or throws an error if the throwOnError attribute is True.  
The cfhttp.responseHeader.Location variable contains the redirection path. ColdFusion follows a maximum of four redirects on a request. If there are more, ColdFusion functions as if redirect = "no".  
Note: The cflocation tag generates an HTTP 302 response with the url attribute as the Location header value. |
| timeout    | Opt     |        | Value, in seconds, that is the maximum time the request can take. If the timeout passes without a response, ColdFusion considers the request to have failed.  
If the client specifies a timeout in the URL search parameter (for example, ?RequestTime=120) ColdFusion uses the lesser of the URL timeout and the timeout attribute value; this ensures that the request times out before, or at the same time as, the page.  
If the URL does not specify a timeout, ColdFusion uses the lesser of the Administrator timeout and the timeout attribute value.  
If the timeout is not set in any of these, ColdFusion waits indefinitely for the cfhttp request to process. |
| getAsBinary| Opt     | no      | • no: if ColdFusion does not recognize the response body type as text, converts it to a ColdFusion object.  
• Auto: if ColdFusion does not recognize the response body type as text, converts it to ColdFusion Binary type data.  
• yes: always converts the response body content into ColdFusion Binary type data, even if ColdFusion recognizes the response body type as text.  
ColdFusion recognizes the response body as text if:  
• the header does not specify a content type.  
• the content type starts with "text".  
• the content type starts with "message".  
• the content type is "application/octet-stream".  
If ColdFusion does not recognize the body as text and converts it to an object, but the body consists of text, the cfoutput tag can display it. The cfoutput tag cannot display Binary type data. (To convert binary data to text, use the ToString function.) |
The following attribute is used with the PUT method to determine how to send data specified with $httpparam type="formField"$:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Req/Opt</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>multipart</td>
<td>Optional</td>
<td>no</td>
<td>Tells ColdFusion to send all data specified by $cfhttpparam type=&quot;formField&quot;$ tags as multipart form data, with a Content-Type of multipart/form-data. By default, ColdFusion sends cfhttp requests that contain only formField data with a Content Type of application/x-www-form-urlencoded. If the request also includes File type data, ColdFusion uses the multipart/form-data content type for all parts. If yes, ColdFusion also sends the request's charset in each Content-Type description. All form field data must be encoded in this character encoding, and ColdFusion does not URLEncode the data. (The field name must be in ISO-88591-1 or ASCII.) Some http parsers, including the one used by previous versions of ColdFusion, ignore the multipart form field character encoding description.</td>
</tr>
</tbody>
</table>

The following attribute allows you to specify the name of the variable in which you would like the results of the operation returned. The name you specify replaces cfhttp as the prefix by which you access the returned variables. For example, if you set the result attribute to myResult, you would access FileContent as #myResult.FileContent#. The result attribute allows functions or CFCs that are called from multiple pages at the same time to avoid overwriting the results of one call with another. For information about the variables returned by a cfhttp get operation, see “Variables returned by a cfhttp get operation” in the Usage section.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Req/Opt</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>result</td>
<td>Optional</td>
<td></td>
<td>Specifies the name of the variable in which you want the result returned.</td>
</tr>
</tbody>
</table>

The following attributes tell ColdFusion to put the HTTP response body in a file. You can put the response body in a file for GET, POST, PUT, DELETE, OPTIONS, and TRACE methods, but it is generally not useful with the DELETE or OPTIONS method.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Req/Opt</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>path</td>
<td>Required</td>
<td></td>
<td>Tells ColdFusion to save the HTTP response body in a file. Contains the absolute path to the directory in which to store the file.</td>
</tr>
<tr>
<td>file</td>
<td>Required</td>
<td></td>
<td>Name of the file in which to store the response body. For a GET operation, the default is the file requested in the URL, if there is one. For example, if the URL in a GET method is <a href="http://www.myco.com/test.htm">http://www.myco.com/test.htm</a>, the default file is test.htm. Do not specify the path to the directory in this attribute; use the path attribute.</td>
</tr>
</tbody>
</table>
The following attributes tell ColdFusion to convert the HTTP response body into a ColdFusion query object. They can be used with the GET and POST methods only.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Req/Opt</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Opt</td>
<td></td>
<td>Tells ColdFusion to create a query object with the given name from the returned HTTP response body.</td>
</tr>
<tr>
<td>columns</td>
<td>Opt</td>
<td>First row of response contains column names.</td>
<td>The column names for the query, separated by commas, with no spaces. Column names must start with a letter. The remaining characters can be letters, numbers, or underscore characters (_). If there are no column name headers in the response, specify this attribute to identify the column names. If you specify this attribute, and the firstrowasHeader attribute is True (the default), the column names specified by this attribute replace the first line of the response. You can use this behavior to replace the column names retrieved by the request with your own names. If a duplicate column heading is encountered in either this attribute or in the column names from the response, ColdFusion appends an underscore to the name to make it unique. If the number of columns specified by this attribute does not equal the number of columns in the HTTP response body, ColdFusion generates an error.</td>
</tr>
<tr>
<td>firstrowasHeaders</td>
<td>Opt</td>
<td>yes</td>
<td>Determines how ColdFusion processes the first row of the query record set:</td>
</tr>
<tr>
<td>delimiter</td>
<td>Opt</td>
<td>, [comma]</td>
<td>A character that separates query columns. The response body must use this character to separate the query columns.</td>
</tr>
</tbody>
</table>
| textQualifier | Opt     | " [double-quotation mark] | A character that, optionally, specifies the start and end of a text column. This character must surround any text fields in the response body that contain the delimiter character as part of the field value. To include this character in column text, escape it by using two characters in place of one. For example, if the qualifier is a double-quotation mark, escape it as " ".

Usage

The cfhttp tag is a general-purpose tool for creating HTTP requests and handling the returned results. It enables you to generate most standard HTTP request types. You use embedded cfhttpparam tags to specify request headers and body content.
When ColdFusion receives a response to a `cfhttp` request, it can put the response body (if any) in a file or the `cfhttp.FileContent` string variable. If the body text is structured as a result set, ColdFusion can put the body text in query object. You can also access the values of all returned headers and specify how to handle error status and redirections, and specify a timeout to prevent requests from hanging.

The HTTP protocol is the backbone of the World Wide Web and is used for every web transaction. Because the `cfhttp` tag can generate most types of requests, it provides significant flexibility. Possible uses include:

- Interacting with dynamic web sites and services that are not available as web services. (Use the `cfinvoke` tag to access SOAP web services.)
- Getting the contents of an HTML page or other file such as an image on a web server for use in your CFML page or storage in a file.
- Sending a secure request to a server by specifying the `https` protocol in the `url` attribute.
- Using the POST method to send a multipart/form-data style post to any URL that can handle such data and return results, including CGI executables or even other ColdFusion pages.
- Using the PUT method to upload files to a server that does not accept FTP requests.

This tag can, and for PUT and POST requests must, have a body that contains `cfhttpparam` tags. If this tag has `cfhttpparam` tags, it must have a `</cfhttp>` end tag.

To use HTTPS with the `cfhttp` tag, you might need to manually import the certificate for each web server into the keystore for the JRE that ColdFusion uses. This procedure should not be necessary if the certificate is signed (issued) by an authority that the JSSE (Java Secure Sockets Extension) recognizes (for example, Verisign); that is, if the signing authority is in the cacerts already. However, you might need to use the procedure if you are issuing SSL (secure sockets layer) certificates yourself.

**To manually import a certificate:**
1. Go to a page on the SSL server in question.
2. Double-click the lock icon.
3. Click the Details tab.
4. Click Copy To File.
5. Select the `base64` option and save the file.
6. Copy the CER file into `C:\CFusionMX7\runtime\jre\lib\security` (or whichever JRE ColdFusion is using).
7. Run the following command in the same directory (keytool.exe is located in `C:\CFusionMX7\runtime\jre\bin`):
   
   ```
   keytool -import -keystore cacerts -alias giveUniqueName -file filename.cer
   ```
Variables returned by a cfhttp get operation

The cfhttp tag returns the following variables. If you set the result attribute, the name you assign replaces cfhttp as the prefix. For additional information, see the result attribute.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cfhttp.charSet</td>
<td>Response character character set (character encoding) specified by the response Content-Type header.</td>
</tr>
<tr>
<td>cfhttp.errorDetail</td>
<td>If the connection to the HTTP server fails, contains details about the failure. For instance: &quot;Unknown host: my.co.com&quot;; otherwise, the empty string. Macromedia recommends that you check this variable for an error condition before checking other variables.</td>
</tr>
<tr>
<td>cfhttp.fileContent</td>
<td>Response body; for example, the contents of a html page retrieved by a GET operation. Empty if you save the response in a file.</td>
</tr>
<tr>
<td>cfhttp.header</td>
<td>Raw response header containing all header information in a single string. Contains the same information as the cfhttp.responseHeader variable.</td>
</tr>
<tr>
<td>cfhttp.mimeType</td>
<td>MIME type specified by the response Content-Type header; for example, text/html.</td>
</tr>
<tr>
<td>cfhttp.responseHeader</td>
<td>The response headers formatted into a structure. Each element key is the header name, such as Content-Type or Status_Code. If there is more than one instance of a header type, the type values are put in an array. One common technique is to dynamically access the cfhttp.responseHeader structure as a dynamic array; for example, #cfhttp.responseHeader[fieldVariable]#.</td>
</tr>
<tr>
<td>cfhttp.statusCode</td>
<td>The HTTP status_code header value followed by the HTTP Explanation header value; for example, “200 OK”.</td>
</tr>
</tbody>
</table>
| cfhttp.text           | Boolean; True if the response body content type is text. ColdFusion recognizes the response body as text if:  
  • the header does not specify a content type.  
  • the content type starts with "text".  
  • the content type starts with "message".  
  • the content type is "application/octet-stream". |

Building a query from a delimited text file

The cfhttp tag can create a ColdFusion query object form the response body. To do so, the response body must consist of lines of text, with each line having fields that are delimited by a character that identifies the column breaks. The default delimiter is a comma (,). The response data can also use a text qualifier; the default is a double-quotation mark ("). If you surround a string field in the text qualifier, the field can contain the delimiter character. To include the text qualifier in field text, escape it by using a double character. The following line shows a two-line request body that is converted into a query. It has three comma-delimited fields:

```
Field1,Field2,Field3
"A comma in text","A quote: "Oh My!"",Plain text
```
Run the following code to show how ColdFusion treats this data:

```cfc
<cfhttp method="Get"
    url="127.0.0.1:8500/tests/escapetest.txt"
    name="onerow">
    <cfdump var="#onerow#"/>
</cfhttp>
```

Column names can be specified in three ways:

- By default, ColdFusion uses the first row of the response as the column names.
- If you specify a comma-delimited `columns` attribute, ColdFusion uses the names specified in the attribute as the column names. Set `firstRowAsHeaders="no"` if the first row of the response contains data. Otherwise, ColdFusion ignores the first row.
- If you do not specify a `columns` attribute and set `firstRowAsHeaders="no"`, ColdFusion generates column names of the form `Column_1, Column2, etc.`

The `cfhttp` tag checks to ensure that column names in the data returned by the tag start with a letter and contain only letters, numbers, and underscore characters (_).

ColdFusion checks for invalid column names. Column names must start with a letter. The remaining characters can be letters, numbers, or underscores (_). If a column name is not valid, ColdFusion generates an error.

**Notes**

- For the ColdFusion MX Administrator timeout and the URL timeout to take effect, you must enable the timeout in the ColdFusion MX Administrator, Server Settings page. For more information, see *Configuring and Administering ColdFusion MX*.
- The `cfhttp` tag supports Basic Authentication for all operations.
- The `cfhttp` tag uses SSL to negotiate secure transactions.
- If you put the HTTP response body in a file, ColdFusion does not put it in the `CFHTTP.FileContent` variable or generate a query object. If you do not put the response body in a file, ColdFusion puts it in the `CFHTTP.FileContent` variable; if you specify a `name` attribute ColdFusion generates a query object.
- The `cfhttp` tag does not support NTLM or Digest Authentication.

**Example**

```cfc
<!--- This example displays the information provided by the Macromedia Designer & Developer Center XML feed, http://www.macromedia.com/desdev/resources/macromedia_resources.xml
See http://www.macromedia.com/desdev/articles/xml_resource_feed.html
for more information on this feed. --->
<cfset urlAddress="http://www.macromedia.com/desdev/resources/macromedia_resources.xml"/>

<!--- Set the URL address. --->
<cfset urlAddress="http://www.macromedia.com/desdev/resources/macromedia_resources.xml"/>

<!--- Use the CFHTTP tag to get the file content represented by urladdress.
Note that />, not an end tag, terminates this tag. --->
<cfhttp url="#urlAddress#" method="GET" resolveurl="Yes" throwOnError="Yes"/>
```
<cfset xmlDoc = XmlParse(CFHTTP.FileContent)>
<!--- Get the array of resource elements, the xmlChildren of the xmlRoot. --->
<cfset resources = xmlDoc.xmlRoot.xmlChildren>
<cfset numresources = ArrayLen(resources)>

<cfloop index="i" from="1" to="#numresources#">
  <cfset item = resources[i]>
  <cfoutput>
    <strong><a href="#item.url.xmltext#">#item.title.xmltext#</a></strong><br>
    <strong>Author</strong>&nbsp;&nbsp;#item.author.xmltext#<br>
    <strong>Applies to these products</strong><br>
    <cfloop index="i" from="4" to="#arraylen(item.xmlChildren)#">
      #item.xmlChildren[i].xmlAttributes.Name#<br>
    </cfloop>
  </cfoutput>
</cfloop>
cfhttpparam

Description

Allowed inside cfhttp tag bodies only. Required for cfhttp POST operations. Optional for all others. Specifies parameters to build an HTTP request.

Category

Internet Protocol tags

Syntax

<cfhttpparam
    type = "transaction type"
    name = "data name"
    value = "data value"
    file = "filename"
    encoded = "yes" or "no"
    mimeType = "MIME type designator"/>

See also

cfhttp, GetHttpRequestData, cfftp, cfldap, cfmail, cfmailparam, cfpop

History

ColdFusion MX 6.1:
• Added the header and body types.
• Added the encoded and mimeType attributes.
• Changed HTTP method behavior: all HTTP methods can have httpparam tags.
• Changed the name attribute requirements: it is not required for all types.
## Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Req/Opt</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
</table>
| type      | Required | Information type:  
  • header: specifies an HTTP header. ColdFusion does not URL encode the header.  
  • CGI: specifies an HTTP header. ColdFusion URL encodes the header by default.  
  • body: specifies the body of the HTTP request. ColdFusion does not automatically set a content-type header or URL encode the body contents. To specify the content-type, use a separate `cfhttp` tag with `type=header`.  
  • XML: identifies the request as having a content-type of text/xml. Specifies that the `value` attribute contains the body of the HTTP request. Used to send XML to the destination URL. ColdFusion does not URL encode the XML data.  
  • file: tells ColdFusion to send the contents of the specified file. ColdFusion does not URL encode the file contents.  
  • URL: specifies a URL query string name-value pair to append to the `cfhttp.url` attribute. ColdFusion URL encodes the query string.  
  • formField: specifies a form field to send. ColdFusion URL encodes the Form field by default.  
  • cookie: specifies a cookie to send as an HTTP header. ColdFusion URL encodes the cookie. |
| name      | Required. Optional (and ignored) for Body and XML types | Variable name for data that is passed. Ignored for Body and XML types. For File type, specifies the filename to send in the request. |
| value     | Required. Optional (and ignored) for File type | Value of the data that is sent. Ignored for File type. The value must contain string data or data that ColdFusion can convert to a string for all `type` attributes except Body. Body types can have string or binary values. |
| file      | Required only if `type=*File` | Applies to File type; ignored for all other types. The absolute path to the file that is sent in the request body. |
| encoded   | Optional yes | Applies to FormField and CGI types; ignored for all other types. Specifies whether to URL encode the form field or header. |
| mimeType  | Optional | Applies to File type; invalid for all other types. Specifies the MIME media type of the file contents. The content type can include an identifier for the character encoding of the file; for example, text/html; charset=ISO-8859-1 indicates that the file is HTML text in the ISO Latin-1 character encoding. |
Usage

Specifies header or body data to send in the HTTP request. The type attribute identifies the information that the parameter specifies. A cfhttp tag can have multiple cfhttpparam tags, subject to the following limitations:

- An XML type attribute cannot be used with additional XML type attributes, or with body, file, or formField type attributes.
- A body type attribute cannot be used with additional body type attributes, or with XML, file, or formField type attributes.
- The XML and body type attributes cannot be used with the cfhttp tag TRACE method.
- The file type attribute is only meaningful with the cfhttp tag POST and PUT methods.
- The formField type attribute is only meaningful with the cfhttp tag POST and GET methods.

If you send an HTTP request to a ColdFusion page, all HTTP headers, not just those sent using the CGI type, are available as CGI scope variables. However, any custom variables (such as "myVar") do not appear in a dump of the CGI scope.

When you send a file using the type="file" attribute, the file content is sent in the body of a multipart/form-data request. If you send the file to a ColdFusion page, the Form scope of the receiving page contains an entry with the name you specified in the cfhttpparam tag name attribute as the key. The value of this variable is the path to a temporary file containing the file that you sent. If you also send Form field data, the location of the filename in the form.fieldnames key list depends on the position of the cfhttpparam tag with the file relative to the cfhttp tags with the form data.

URL-encoding preserves special characters (such as the ampersand) when they are passed to the server. For more information, see the function URLEncodedFormat on page 899.

To send arbitrary data in a "raw" HTTP message, use a cfhttpparam tag with a type="body" attribute to specify the body content and use cfhttpparam tags with a type="header" attributes to specify the headers.

Example

<!--- This example consists of two CFML pages. The first page posts to the second. --->

<!--- The first, posting page. This page posts variables to another page and displays the body of the response from the second page. Change the URL and port as necessary for your environment. --->

<cfhttp
    method="post"
    url="http://127.0.0.1/tests/http/cfhttpparamexample.cfm"
    port="8500"
    throwonerror="Yes">
   <cfhttpparam name="form_test" type="FormField" value="This is a form variable">
   <cfhttpparam name="url_test" type="URL" value="This is a URL variable">
   <cfhttpparam name="body_test" type="body" value="This is a body variable">
   </cfhttp>

<cfhttpparam name="form_test" type="FormField" value="This is a form variable">
<cfhttpparam name="url_test" type="URL" value="This is a URL variable">
<cfhttpparam name="body_test" type="body" value="This is a body variable">
</cfhttpparam>
<cfhttpparam name="cgi_test" type="CGI" value="This is a CGI variable">
<cfhttpparam name="cookie_test" type="Cookie" value="This is a cookie">
</cfhttp>

<!--- Output the results returned by the posted-to page. --->
<cfoutput>
#cfhttp.fileContent#
</cfoutput>

<!--- This is the cfhttpparamexample.cfm page that receives and processes the
Post request. Its response body is the generated HTML output. --->

<h3>Output the passed variables</h3>
<cfoutput>
Form variable: #form.form_test#
<br>URL variable: #URL.url_test#
<br>Cookie variable: #Cookie.cookie_test#
<br>CGI variable: #CGI.cgi_test#<br>
<br>Note that the CGI variable is URL encoded.
</cfoutput>
**cfif**

**Description**
Creates simple and compound conditional statements in CFML. Tests an expression, variable, function return value, or string. Used, optionally, with the `cfelse` and `cfelseif` tags.

**Category**
Flow-control tags

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

**See also**
`cfelse, cfelseif, cfabort, cbreak, cfexecute, cfexit, cffunction, cflocation, cfloop, cfswitch, cfthrow, cftry`

**Usage**
If the value of the expression in the `cfif` tag is True, ColdFusion processes all the code that follows, up to any `cfelseif` or `cfelse` tag, and then skips to the `cfif` end tag. Otherwise, ColdFusion does not process the code that immediately follows the `cfif` tag, and continues processing at any `cfelseif` or `cfelse` tag, or with the code that follows the `cfif` end tag.

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

```cfif IsArray(myarray)`
If successful, `IsArray` evaluates to yes, the string equivalent of the Boolean True. This is preferred over explicitly defining the True condition this way:
```
<cfif IsArray(myarray) IS True>`
This tag requires an end tag.

**Example**
In this example, variables are shown within number signs. This is not required.
```html
<!--- This example shows the interaction of cfif, cfelse, and cfelseif. --->
<!----- First, perform a query to get some data. ----->
<cfquery name="getCenters" datasource="cfdocexamples">
  SELECT Center_ID, Name, Address1, Address2, City, State, Country, PostalCode, Phone, Contact
  FROM Centers
  ORDER by City, State, Name
</cfquery>
<p>CFIF gives us the ability to perform conditional logic based on a condition or set of conditions.
For example, we can output the list of Centers from the snippets datasource by group and only display them <b>IF</b> City = San Diego.
<hr>
<!---- Use CFIF to test a condition when outputting a query. ----->
<p>The following centers are in San Diego:
<cfoutput query="getCenters">
  <cfif Trim(City) is "San Diego">
    <br><b>Name/Address:</b>#Name#, #Address1#, #City#, #State#
    <br><b>Contact:</b> #Contact#
    <br>
  </cfif>
</cfoutput>
<hr>
<p>If we would like more than one condition to be the case, we can ask for a list of the centers in San Diego <b>OR</b> Santa Ana. If the center does not follow this condition, we can use CFELSE to show only the names and cities of the other centers.
<p>Notice how a nested CFIF is used to specify the location of the featured site (Santa Ana or San Diego).
<!----- Use CFIF to specify a conditional choice for multiple options; also note the nested CFIF. --->
<p>Complete information is shown for centers in San Diego or Santa Ana. All other centers are listed in italics:
<cfoutput query="getCenters">
  <cfif Trim(City) is "San Diego" OR Trim(City) is "Santa Ana">
    <h4>Featured Center in</h4> <cfif Trim(City) is "San Diego">San Diego</cfif> <cfelse>Santa Ana</cfif>
    <br><b>Name/Address:</b>#Name#, #Address1#, #City#, #State#
    <br><b>Contact:</b> #Contact#<br>
  </cfif>
  <cfelse>
    <br><i>#Name#, #City#</i>
  </cfif>
</cfoutput>
<hr>
<p>Finally, we can use CFELSEIF to cycle through a number of conditions and produce varying output. Note that you can use CFCASE and CF SWITCH for a more elegant representation of this behavior.
<!----- Use CFIF in conjunction with CFELSEIF to specify more than one branch in a conditional situation. --->
<cfoutput query="getCenters">
  <cfif Trim(City) is "San Diego" OR Trim(City) is "Santa Ana">
    <br><i>#Name#, #City#</i> (this one is in <cfif Trim(City) is "San Diego">San Diego</cfif> <cfelse>Santa Ana</cfif>)
  </cfif>
  <cfelseif Trim(City) is "San Francisco">
    <br><i>#Name#, #City#</i> (this one is in San Francisco)
  </cfelseif Trim(City) is "Suisun">
    <br><i>#Name#, #City#</i> (this one is in Suisun)
  </cfelseif Trim(City) is "Not in a city we track">
    <br><i>#Name#</i>
  </cfif>
</cfoutput>
cfimpersonate

Description

This tag is obsolete. Use the newer security tools; see “Conversion functions” on page 453 and Chapter 16, “Securing Applications” in ColdFusion MX Developer’s Guide.

History

ColdFusion MX: This tag is obsolete. It does not work in ColdFusion MX and later releases.
**cfimport**

**Description**

You can use the `cfimport` tag to import either of the following:

- All ColdFusion pages in a directory, as a tag custom tag library.
- A Java Server Page (JSP) tag library. A JSP tag library is a packaged set of tag handlers that conform to the JSP 1.1 tag extension API.

**Category**

Application framework tags

**Syntax**

```cfml
<cfimport
taglib = "taglib-location"
prefix = "custom">
```

**See also**

cfapplication

**History**

ColdFusion MX: Added this tag.

**Attributes**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Req/Opt</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
</table>
| taglib    | Required|         | Tag library URI. The path must be relative to the web root (and start with /), the current page location, or a directory specified in the Administrator ColdFusion mappings page.  
• A directory in which custom ColdFusion tags are stored. In this case, all the cfm pages in this directory are treated as custom tags in a tag library.  
• A path to a JAR in a web-application; for example, "/WEB-INF/lib/sometags.jar"  
• A path to a tag library descriptor; for example, "/sometags.tld"  
**Note:** You must put JSP custom tag libraries in the /WEB-INF/lib directory. This limitation does not apply to ColdFusion pages. |
| prefix    | Required|         | Prefix by which to access the imported custom CFML tags JSP tags.  
If you import a CFML custom tag directory and specify an empty value, "" , for this attribute, you can call the custom tags without using a prefix. You must specify and use a prefix for a JSP tag library. |
Usage

The following example imports the tags from the directory myCustomTags:

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

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

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

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

To use JSP custom tags in a ColdFusion page:

1. Put a JSP tag library JAR file (for example, myjsptags.jar) into the ColdFusion server directory `wwwroot/WEB-INF/lib`. If the tag library has a separate TLD file, put it in the same directory as the JAR file.

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

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

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

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

The `cfimport` tag must be on the page that uses the imported tags. For example, if you use a `cfimport` tag on a page that you include with the `cfinclude` call, you cannot use the imported tags on the page that has the `cfinclude` tag. Similarly, if you have a `cfimport` tag on your `Application.cfm` page, the imported tags are available on the `Application.cfm` page only, not on the other pages in the application. ColdFusion does not throw an error in these situations, but the imported tags do not run.

You cannot use the `cfimport` tag to suppress output from a tag library.

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

Example

```<h3>cfimport example</h3>
<p>This example uses the random JSP tag library that is available from the Jakarta Taglibs project, at http://jakarta.apache.org/taglibs/

```<cfimport taglib="/WEB-INF/lib/taglibs-random.jar" prefix="randomnum">

```<randomnum:number id="randPass" range="000000-999999" algorithm="SHA1PRNG"
    provider="SUN" />
```

```<cfset myPassword = randPass.random>
```

```<cfoutput>
    Your password is #myPassword#
</cfoutput>
```
**cfinclude**

**Description**

Embeds references to ColdFusion pages in CFML. You can embed `cfinclude` tags recursively. For another way to encapsulate CFML, see `cfmodule` on page 305. (A ColdFusion page was formerly sometimes called a ColdFusion template or a template.)

**Category**

Flow-control tags, Page processing tags

**Syntax**

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

**See also**

cfcache, cfflush, cfheader, cfhtmlhead, cfsetting, cfsilent

**History**

ColdFusion MX: Changed error behavior: if you use this tag to include a CFML page whose length is zero bytes, you do not get an error.

**Attributes**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Req/Opt</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>template</td>
<td>Required</td>
<td>A logical path to a ColdFusion page.</td>
<td></td>
</tr>
</tbody>
</table>

**Usage**

ColdFusion searches for included files in the following locations:

1. In the directory of the current page or a directory relative to the current page
2. In directories mapped in the ColdFusion MX Administrator

You cannot specify an absolute URL or file system path for the file to include. You can only use paths relative to the directory of the including page or a directory that is registered in the ColdFusion MX Administrator Mappings. The following `cfinclude` statements will work, assuming that the `myinclude.cfm` file exists in the specified directory:

```
<cfinclude template="myinclude.cfm">
<cfinclude template="/CFIDE/debug/myinclude.cfm">
```

But these will not work:

```
<cfinclude template="C:\CFusionMX7\wwwroot\doccomments\myinclude.cfm">
<cfinclude template="http://localhost:8500/doccomments/myinclude.cfm">
```

The included file must be a syntactically correct and complete CFML page. For example, to output data from within the included page, you must have a `cfoutput` tag, including the end tag, on the included page, not the referring page. Similarly, you cannot span a `cfif` tag across the referring page and the included page; it must be complete within the included page.
You can specify a variable for the template attribute, as the following example shows:

```cfml
<cfset templatetouse="../header/header.cfm">
<cfinclude template="#templatetouse#">
```

**Example**

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

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

**Description**

Populates a Verity collection with metadata and creates indexes for searching it. Verity is a search engine that you can integrate in a ColdFusion application to search physical files of various types or a database query. Indexing database columns that result from a query lets users search the query data much faster than they could if you used multiple SQL queries to return the same data.

You must define a Verity collection using the ColdFusion MX Administrator or the `cfcollection` tag before creating indexes for the collection.

You also can index a Verity collection using the ColdFusion MX Administrator or by using a native Verity indexing tool, such as Vspider or MKVDK. These options, however, limit you to indexing a collection of files in a directory path.

For more information on creating, indexing, and searching a Verity collection, see Chapter 24, “Building a Search Interface” in *ColdFusion MX Developer's Guide*.

**Category**

Extensibility tags

**Syntax**

```xml
<cfindex
    collection = "collection_name"
    action = "action"
    type = "type"
    title = "title"
    key = "ID"
    body = "body"
    custom1 = "custom_value"
    custom2 = "custom_value"
    custom3 = "custom_value"
    custom4 = "custom_value"
    category = "category_name"
    categoryTree = "category_tree"
    URLpath = "URL"
    extensions = "file_extensions"
    query = "query_name"
    recurse = "yes" or "no"
    language = "language">
    status = "status">
```

**See also**

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

**History**

ColdFusion MX 7:

- Added the status, category, and categoryTree attributes.
- Removed reference to external collections.
- Removed suggested cflock usage.
ColdFusion MX:

- The `action` attribute value `optimize` is obsolete. It does not work, and might cause an error, in ColdFusion MX.
- Changed the `external` attribute behavior: it is not necessary to specify the `external` attribute. (ColdFusion automatically detects whether a collection is internal or external.)
- Changed Verity operations behavior: ColdFusion supports Verity operations on Acrobat PDF files.
- Changed thrown exceptions: this tag can throw the `SEARCHENGINE` exception.
- Changed acceptable collection naming: this tag accepts collection names that include spaces.
- Changed query result behavior: the `cfindex` tag can index the query results from a `cfsearch` tag.

Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Req/Opt</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>collection</td>
<td>Required</td>
<td>Name of a collection that is registered by ColdFusion; for example, &quot;personnel&quot;.</td>
<td></td>
</tr>
<tr>
<td>action</td>
<td>Required</td>
<td></td>
<td>• <code>update</code>: updates a collection and adds <code>key</code> to the index.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• <code>delete</code>: removes collection documents as specified by the <code>key</code> attribute.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• <code>purge</code>: deletes all of the documents in a collection. Causes the collection to be taken offline, preventing searches.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• <code>refresh</code>: deletes all of the documents in a collection, and then performs an update.</td>
</tr>
<tr>
<td>type</td>
<td>Optional</td>
<td><code>custom</code>, if <code>query</code> attribute is specified. Otherwise, <code>file</code>.</td>
<td>• <code>file</code>: applies <code>action</code> value to filename, including path. Expects a filename in the <code>key</code> attribute.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• <code>path</code>: applies <code>action</code> to files in a directory path that pass the <code>extensions</code> filter. Expects a directory name in the <code>key</code> attribute.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• <code>custom</code>: applies <code>action</code> to custom data; for example, to data from a query.</td>
</tr>
<tr>
<td>title</td>
<td>Optional</td>
<td></td>
<td>Provides a title for the document if one cannot be extracted from the document.</td>
</tr>
<tr>
<td>key</td>
<td>Required</td>
<td>(empty string)</td>
<td>The value specified for <code>key</code> depends on the <code>type</code> attribute:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• If <code>type</code> = &quot;file&quot;, the directory path and filename for the file.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• If <code>type</code> = &quot;path&quot;, the directory path for the location of the files.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• If <code>type</code> = &quot;custom&quot;, a unique identifier that specifies the location of the data. For a query, the name of the column that holds the primary key, for example. If not a query, an identifier such as the URL for a web page, for example.</td>
</tr>
<tr>
<td>Attribute</td>
<td>Req/Opt</td>
<td>Default</td>
<td>Description</td>
</tr>
<tr>
<td>------------</td>
<td>---------</td>
<td>---------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
</tbody>
</table>
| body       | Required if `type = custom` | • ASCII text to index.  
• Query column names, if name is specified in query.  
You can specify columns in a delimited list; for example: "emp_name, dept_name, location".  
This attribute is ignored if `type` is file or path, and is invalid if `action` is delete. |
| custom1    | Optional | Use to index discrete values in collection records, which lets you search for specific records using the Verity MATCHES operator. By contrast, values specified in the body attribute are concatenated and searched as a body of text using the specified criteria.  
If `type = query`, a query column name. If `type = custom`, a data field to be indexed. |
| custom2    | Optional | Usage is the same as for custom1. |
| custom3    | Optional | Usage is the same as for custom1. |
| custom4    | Optional | Usage is the same as for custom1. |
| category   | Optional | A string value that specifies one or more search categories for which to index the data. You can define multiple categories, separated by commas, for a single index. |
| categoryTree | Optional | A string value that specifies a hierarchical category or category tree for searching. It is a series of categories separated by forward slashes (/). You can specify only one category tree. |
| URLpath    | Optional | If `type` is file or path, specifies the URL path. During indexing, this pathname is prefixed to filenames and returned from a search as the url. |
| extensions | Optional | htm, html, cfm, cfml, dbm, dbml  
Delimited list of file extensions that ColdFusion uses to index files, if `type = "Path"`.  
"*.*" returns files with no extension. ".*" returns all files.  
For example, the following code returns files with a listed extension or no extension: extensions == ".htm, .html, .cfm, .cfml, ".*" |
| query      | Optional. | The name of the query against which the collection is generated. |
| recurse    | Optional | no  
• yes: if `type = "path"`, indexes qualified files in directories below the path specified in the key attribute.  
• no |
| language   | Optional | English  
For options, see cfcollection. Requires the appropriate Verity Locales language pack (Western Europe, Asia, Multilanguage, Eastern Europe/Middle Eastern). |
| status     | Optional | The name of the structure into which ColdFusion MX returns status information. |
Usage

The attributes settings that the cfindex tag requires depend on whether you set the query attribute. If you set the query attribute to a valid query name, it specifies that cfindex is to index the data in the query rather than indexing documents on a disk. If you do not set the query attribute, cfindex assumes it is indexing a file (type = file), a set of files in a directory path (type = path), or text that you provide in the body attribute (type = custom).

If you set the query attribute to a valid query name, the cfindex tag creates indexes as specified by the following attributes and their values:

<table>
<thead>
<tr>
<th>Type</th>
<th>Attribute values</th>
</tr>
</thead>
<tbody>
<tr>
<td>File</td>
<td>The key attribute is the name of a column in the query that contains a full filename (including path).</td>
</tr>
<tr>
<td>Path</td>
<td>The key attribute is the name of a column in the query that contains a directory pathname.</td>
</tr>
<tr>
<td></td>
<td>The extensions and recurse attributes, if specified, elaborate on which files are included.</td>
</tr>
<tr>
<td></td>
<td>If the action is delete, cfindex deletes keys for the collection.</td>
</tr>
<tr>
<td>Custom</td>
<td>The key attribute specifies a column name that contains anything you want; for example, the primary key value in the database. It must be unique because this is the primary key in the collection. If the action is delete, the key attribute is the name of a column in the query that contains the keys to delete.</td>
</tr>
<tr>
<td></td>
<td>The body attribute is required and is a comma-delimited list of the names of the columns that contain the text data to be indexed.</td>
</tr>
</tbody>
</table>

If you do not set the query attribute, the cfindex tag creates indexes as specified by the following attributes and their values:

<table>
<thead>
<tr>
<th>Type</th>
<th>Attribute values</th>
</tr>
</thead>
<tbody>
<tr>
<td>File</td>
<td>The key attribute is required and is a full pathname to a file.</td>
</tr>
<tr>
<td>Path</td>
<td>The key attribute is required and it is a directory pathname.</td>
</tr>
<tr>
<td></td>
<td>The extensions and recurse attributes, if specified, designate which types of files are included.</td>
</tr>
<tr>
<td></td>
<td>If the action is delete, both the keys and the document files are deleted.</td>
</tr>
<tr>
<td>Custom</td>
<td>The key attribute is an identifier that specifies the key. If the action is delete, the key attribute is the document key to delete.</td>
</tr>
<tr>
<td></td>
<td>The body attribute is required and is the text to be indexed.</td>
</tr>
</tbody>
</table>

If type is not specified but query is set, ColdFusion sets the type to the default value of custom.

If neither type nor query is set, ColdFusion sets type to the default value of file.

If type equals custom, all attributes except for key and body can specify a literal value, not only a column name. This allows you to change a field to empty in the collection.
Status attribute

The status attribute provides the following information and diagnostics about the result of a cfindex operation:

<table>
<thead>
<tr>
<th>Key</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BADKEYS</td>
<td>Struct</td>
<td>A structure of keys with diagnostic messages about the indexing of these keys. If there are no bad keys, this key does not exist.</td>
</tr>
<tr>
<td>DELETED</td>
<td>Number</td>
<td>The number of keys deleted.</td>
</tr>
<tr>
<td>MESSAGES</td>
<td>Array</td>
<td>An array of diagnostic messages, including nonfatal errors and warnings, returned from the Verity K2 Index server. If there are no messages, this key does not exist.</td>
</tr>
<tr>
<td>INSERTED</td>
<td>Number</td>
<td>The number of keys inserted into the collection.</td>
</tr>
<tr>
<td>UPDATED</td>
<td>Number</td>
<td>The number of keys updated in the collection.</td>
</tr>
</tbody>
</table>

Example

<!--- EXAMPLE #1 Index a file, type = "file". ------------------------------>  
<!--- Example dumps content of status variable (info). --------------------->  
<cfindex collection="CodeColl"  
    action="refresh"  
    type="file"  
    key="C:\blackstone\wwwroot\vw_files\cfindex.htm"  
    urlpath="http://localhost:8500/vw_files/"  
    language="English"  
    title="Cfindex Reference page"  
    status="info">  

<!--- Search for Attributes. --->
<cfsearch
    name = "mySearch"  
    collection = "CodeColl"  
    criteria = "Attributes"  
    contextpassages = "1"  
    maxrows = "100">  
<cfoutput>
    key=#mySearch.key#<br />
    title=#mySearch.title#<br />
    context=#mySearch.context#<br />
    url=#mySearch.url#<br />
</cfoutput>

<cfdump var="#info#">  

<!--- EXAMPLE #2 Index a path (type = "path"). ------------------------------>  
<cfindex collection="CodeColl"  
    action="refresh"  
    type="path"  
    key="C:\inetpub\wwwroot\vw_files\newspaper\sports"  
    urlpath="http://localhost/vw_files/newspaper/sports"  
    extensions = ".htm, .html"  
    recurse="no"  

language="English"
categoryTree="vw_files/newspaper/sports"
category="Giants">

<!--- Search for any references to criteria. --->
<cfsearch
   name = "mySearch"
collection = "CodeColl"
categoryTree="vw_files/newspaper/sports"
category="Giants"
criteria = "Williams"
contextpassages = "1"
maxrows = "100">
<cfoutput>
   key=#mySearch.key#<br />
title=#mySearch.title#<br />
context=#mySearch.context#<br />
url=#mySearch.url#<br />
</cfoutput>
</cfsearch>

<!---EXAMPLE #3: Index a QUERY (type = "custom") using custom1. ------------>
<!--- Retrieve data from the table. --->
<cfquery name="getCourses" datasource="cfdocexamples">
   SELECT * FROM COURSES
</cfquery>

<!--- Update the collection with the above query results. --->
<!--- key is Course_ID in the Courses table. ---->
<!--- body specifies the columns to be indexed for searching. --->
<!--- custom1 specifies the value of the Course_Number column. --->
<cfindex
   query="getCourses"
collection="CodeColl"
action="Update"
type="Custom"
Key="Course_ID"
title="Courses"
body="Course_ID,Descript"
custom1="Course_Number">
<h2>Indexing Complete</h2>
<!--- cno supplies value for searching custom1; could be form input instead. --->
<cfset cno = "540">
<cfsearch
   name = "mySearch"
collection = "CodeColl"
criteria = "CF_CUSTOM1 <MATCHES> #cno#"
contextpassages = "1"
maxrows = "100">
<!--- Returns indexed values (Course_ID and Descript) for Course_Number 540. --->
<cfoutput>
   key=#mySearch.key#<br />
</cfoutput>
title=#mySearch.title#
context=#mySearch.context#
url=#mySearch.url#
</cfoutput>

<!--- EXAMPLE #4 Index a FILE within a QUERY (type= "file"). -------------
<!--- Retrieve row with a column that contains a filename (Contract_File). --->
<cfquery name="getEmps" datasource="cfdocexamples">
   SELECT * FROM EMPLOYEE WHERE EMP_ID = 1
</cfquery>

<!--- Update the collection with the above query results. --->
<!--- key specifies the column that contains a complete filename. --->
<!--- file is indexed in same way as if no query involved. --->
<cfindex
   query="getEmps"
   collection="CodeColl"
   action="Update"
   type="file"
   key="Contract_File"
   title="Contract_File"
   body="Emp_ID,FirstName,LastName,Contract_File">
<h2>Indexing Complete</h2>
<cfsearch
   name = "mySearch"
   collection = "CodeColl"
   criteria = "vacation"
   contextpassages = "1"
   maxrows = "100">
<cfoutput>
   key=#mySearch.key#
   title=#mySearch.title#
   context=#mySearch.context#
   url=#mySearch.url#
</cfoutput>
</cfsearch>

<!--- EXAMPLE # 5 Index a PATH within a QUERY. ----------------------------
<!--- Retrieve a row with a column that contains a path (Project_Docs). --->
<cfquery name="getEmps" datasource="cfdocexamples">
   SELECT * FROM EMPLOYEE WHERE Emp_ID = 15
</cfquery>

<!--- Update the collection with the above query results. --->
<!--- key specifies a column that contains a directory path. --->
<!--- path is indexed in same way as if no query involved. --->
<cfindex
   query="getEmps"
   collection="CodeColl"
   action="update"
   type="path"
   key="Project_Docs"
   title="Project_Docs"
   body="Emp_ID,FirstName,LastName,Project_Docs"
<h2>Indexing Complete</h2>

<cfsearch
  name = "getEmps"
  collection = "CodeColl"
  criteria = "cfsetting"
  contextpassages = "1"
  maxrows = "100">
  <cfoutput>
    key=#getEmps.key#<br />
    title=#getEmps.title#<br />
    context=#getEmps.context#<br />
    url=#getEmps.url#<br />
  </cfoutput>
</cfsearch>

<!--- EXAMPLE #6 Deletes keys in the CodeColl collection for html files --->
<!--- in the specified directory (but not in subdirectories). --------------->

<cfindex collection="CodeColl"
  action="delete"
  type="path"
  key="C:\CFusionMX7\wwwroot\vw_files\newspaper"
  urlpath="http://localhost:8500/vw_files/newspaper"
  extensions = ".htm, .html"
  recurse="no">

<!--- EXAMPLE #7 Purges all keys in the CodeColl collection --->
<!--- with recursion. ------------------------------------------------------->

<cfindex collection="CodeColl"
  action="purge"
  type="path"
  key="C:\CFusionMX7\wwwroot\vw_files\newspaper">
**cfinput**

**Description**

Used within the `cfform` tag, to place input controls that support input validation on a form.

**Category**

Forms tags

**Syntax**

```
<cfinput
  name = "name"
  type = "input type"
  label = "text"
  style = "style specification"
  required = "yes" or "no"
  mask = "masking pattern"
  validate = "data type"
  validateAt= one or more of "onBlur", "onServer", "onSubmit"
  message = "text"
  range = "min_value, max_value"
  maxlength = "number"
  pattern = "regexp"
  onValidate = "script name"
  onError = "script name"
  size = "integer"
  value = "initial value"
  bind = "bind expression"
  checked
  disabled = "true" or "false" or no attribute value
  src = "image URL"
  onKeyUp = "JavaScript or ActionScript"
  onKeyDown = "JavaScript or ActionScript"
  onMouseUp = "JavaScript or ActionScript"
  onMouseDown = "JavaScript or ActionScript"
  onChange = "JavaScript or ActionScript"
  onClick = "JavaScript or ActionScript"
  firstDayOfWeek = "day name"
  dayNames = "days-of-the-week labels"
  monthNames = "month labels"
  enabled = "Yes" or "No"
  visible = "Yes" or "No"
  tooltip = "Tip text"
  height = "number of pixels"
  width = "number of pixels"
/>
```

**See also**

`cfapplet`, `cfcalendar`, `cfgrid`, `cfitem`, `cfselect`, `cfslider`, `cftextarea`, `cfselect`
History

ColdFusion MX 7:

• Added support for button, file, hidden, image, reset, and submit controls.
• Added support for generating Flash and XML controls (specified in the cfform tag).
• Added datefield type (Flash forms only) and the supporting daynames and monthnames attributes.
• Added bind, enabled, height, label, tooltip, visible, and width attributes for use in Flash forms.
• Added support for onBlur and onServer validation, including the validateAt attribute.
• Added USdate, range, boolean, email, URL, uuid, guid, maxlength, noblanks, and submitOnce validation attribute values.
• Added support for preventing multiple submissions.
• Added the mask attribute.
• Deprecated the passthrough attribute. The tag now supports all HTML input tag attributes directly.

ColdFusion MX: Changed the cfform tag preserveData attribute behavior: if it is set to True, ColdFusion checks radio and check box values only if their value matches the posted value for the control. (In earlier releases, if the posted value did not match any of the cfinput check boxes or radio buttons for the control, the checked attribute was used.

Attributes

The following table lists attributes that ColdFusion uses directly. The tag also supports all HTML form tag attributes that are not on this list, and passes them directly to the browser.

Note: Attributes that are not marked as supported in XML are not handled by the skins provided with ColdFusion MX. They are, however, included in the generated XML.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Req/Opt; Formats</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Required; All</td>
<td></td>
<td>Name for form input element.</td>
</tr>
<tr>
<td>type</td>
<td>Optional; text</td>
<td></td>
<td>The input control type to create: button; push button. checkbox; check box. file; file selector; not supported in Flash. hidden; invisible control. image; clickable button with an image. password; password entry control; hides input values. radio; radio button. reset; form reset button. submit; form submission button. text; text entry box. datefield; Flash only; date entry field with an expanding calendar for selecting dates.</td>
</tr>
<tr>
<td>Attribute</td>
<td>Req/Opt; Formats</td>
<td>Default</td>
<td>Description</td>
</tr>
<tr>
<td>-----------</td>
<td>-----------------</td>
<td>---------</td>
<td>-------------</td>
</tr>
<tr>
<td>label</td>
<td>Optional; Flash and XML</td>
<td></td>
<td>Label to put next to the control on a Flash form. Not used for button, hidden, image, reset, or submit types.</td>
</tr>
<tr>
<td>style</td>
<td>Optional; All</td>
<td></td>
<td>In HTML or XML format, ColdFusion passes the style attribute to the browser or XML. In Flash format, must be a style specification in CSS format. For detailed information on specifying Flash styles, see Chapter 29, &quot;Creating Forms in Macromedia Flash&quot; in ColdFusion MX Developer's Guide. In XML format, ColdFusion passes the style attribute to the XML.</td>
</tr>
</tbody>
</table>
| required  | Optional; no All |         | • yes: the field must contain data.  
• no: allows an empty field. |
| mask      | Optional; Flash and HTML |         | A mask pattern that controls the character pattern that users can enter, or that the form sends to ColdFusion. **HTML and Flash:** For tags with `type="text"`. Mask characters and the corresponding valid input characters are:  
• A = [A-Za-z]  
• X = [A-Za-z0-9]  
• 9 = [0-9]  
• ? = Any character  
• All other characters = insert the literal character  
**Flash only:** For tags with `type="datefield"`. ColdFusion uses the mask pattern to format the selected date. Mask characters are:  
• D = day; can use 0-2 mask characters.  
• M = month; can use 0-4 mask characters.  
• Y = year; can use 0, 2, or 4 characters.  
• E = day in week; can use 0-4 characters.  
For more information, see "Masking input data" in Usage. |
| validate  | Optional; All |         | The type or types of validation to do. Available validation types and algorithms depend on the format. For details, see Usage. |
| validateAt | Optional; onSubmit |         | How to do the validation; one or more of the following:  
• onSubmit  
• onServer  
• onBlur  
onBlur and onSubmit are identical in Flash forms. For multiple values, use a comma-delimited list. For details, see Usage. |
| message   | Optional; All |         | Message text to display if validation fails. |

244  Chapter 2: ColdFusion Tags
<table>
<thead>
<tr>
<th>Attribute</th>
<th>Req/Opt; Formats</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>range</td>
<td>Optional; All</td>
<td>Minimum and maximum allowed numeric values. ColdFusion uses this attribute only if you specify <code>range</code> in the <code>validate</code> attribute. If you specify a single number or a single number followed by a comma, it is treated as a minimum, with no maximum. If you specify a comma followed by a number, the maximum is set to the specified number, with no minimum.</td>
<td></td>
</tr>
<tr>
<td>maxLength</td>
<td>Optional; All</td>
<td>Maximum length of text entered, if <code>type = “Text”</code> or “password”. For complete length validation, specify <code>maxLength</code> validation in a <code>validate</code> attribute; otherwise, this attribute prevents users from typing beyond the specified length, but does not prevent them from pasting in a longer value.</td>
<td></td>
</tr>
<tr>
<td>pattern</td>
<td>Required if <code>validate = “regex”</code>; HTML and XML format only</td>
<td>JavaScript regular expression pattern to validate input. ColdFusion uses this attribute only if you specify <code>regex</code> in the <code>validate</code> attribute. Omit leading and trailing slashes. For examples and syntax, see Chapter 27, &quot;Building Dynamic Forms with cfform Tags&quot; in ColdFusion MX Developer's Guide.</td>
<td></td>
</tr>
<tr>
<td>onValidate</td>
<td>Optional; HTML and XML only</td>
<td>Custom JavaScript function to validate user input. The form object, input object, and input object values are passed to the routine, which should return True if validation succeeds, and False otherwise. If used, the <code>validate</code> attribute is ignored.</td>
<td></td>
</tr>
<tr>
<td>onError</td>
<td>Optional; HTML and XML only</td>
<td>Custom JavaScript function to execute if validation fails.</td>
<td></td>
</tr>
<tr>
<td>size</td>
<td>Optional; All</td>
<td>Size of input control. Ignored, if <code>type = “radio”</code> or “checkbox”. If specified in a Flash form, ColdFusion sets the control width pixel value to 10 times the specified size and ignores the <code>width</code> attribute.</td>
<td></td>
</tr>
<tr>
<td>value</td>
<td>depends on type setting; All</td>
<td>HTML: corresponds to the HTML <code>value</code> attribute. Its use depends on control type. Flash: optional; specifies text for button type inputs: button, submit, and image.</td>
<td></td>
</tr>
<tr>
<td>bind</td>
<td>Optional; Flash only</td>
<td>A Flash bind expression that populates the field with information from other form fields. For details, see Usage.</td>
<td></td>
</tr>
</tbody>
</table>
| checked   | Optional; All    | False | Selects a radio or checkbox control:  
|           |                  |        | • True  
<p>|           |                  |        | • False |</p>
<table>
<thead>
<tr>
<th>Attribute</th>
<th>Req/Opt; Formats</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>disabled</td>
<td>Optional; All</td>
<td>not disabled</td>
<td>Disables user input, making the control read-only. To disable input, specify disabled without an attribute or disabled=&quot;yes&quot; (or any ColdFusion positive Boolean value, such as True). To enable input, omit the attribute or specify disabled=&quot;No&quot; (or any ColdFusion negative Boolean value, such as False).</td>
</tr>
<tr>
<td>src</td>
<td>Optional; Flash and HTML</td>
<td></td>
<td>Applies to Flash button, reset, submit, and image types, and the HTML image type. URL of an image to use on the button. Flash does not support GIF images.</td>
</tr>
<tr>
<td>onKeyUp</td>
<td>Optional; All</td>
<td></td>
<td>JavaScript (HTML/XML) or ActionScript (Flash) to run when the user releases a keyboard key in the control.</td>
</tr>
<tr>
<td>onKeyDown</td>
<td>Optional; All</td>
<td></td>
<td>JavaScript (HTML/XML) or ActionScript (Flash) ActionScript to run when the user presses a keyboard key in the control.</td>
</tr>
<tr>
<td>onMouseUp</td>
<td>Optional; All</td>
<td></td>
<td>JavaScript (HTML/XML) or ActionScript (Flash) to run when the user presses a mouse button in the control.</td>
</tr>
<tr>
<td>onMouseDown</td>
<td>Optional; All</td>
<td></td>
<td>JavaScript (HTML/XML) or ActionScript (Flash) to run when the user releases a mouse button in the control.</td>
</tr>
<tr>
<td>onChange</td>
<td>Optional; All</td>
<td></td>
<td>JavaScript (HTML/XML) or ActionScript (Flash) to run when the control changes due to user action. In Flash, applies to datefield, password, and text types only.</td>
</tr>
<tr>
<td>onClick</td>
<td>Optional; All</td>
<td></td>
<td>JavaScript (HTML/XML) or ActionScript (Flash) to run when the user clicks the control. In Flash, applies to button, checkbox, image, radio, reset, and submit types only.</td>
</tr>
<tr>
<td>dayNames</td>
<td>Optional; All</td>
<td>S, M, T, W, Th, F, S</td>
<td>Applies to datefield type only. A comma-delimited list that sets the names of the weekdays displayed in the calendar. Sunday is the first day; the rest of the weekday names follow in the normal order.</td>
</tr>
<tr>
<td>firstDayOfWeek</td>
<td>Optional; All</td>
<td>0</td>
<td>Applies to datefield type only. Integer in the range 0-6 specifying the first day of the week in the calendar: 0 indicates Sunday; 6 indicates Saturday.</td>
</tr>
<tr>
<td>monthNames</td>
<td>Optional; All</td>
<td>January, February, March, April, May, June, July, August, September, October, November, December</td>
<td>Applies to datefield type only. A comma-delimited list of the month names that are displayed at the top of the calendar.</td>
</tr>
</tbody>
</table>
Note: Attributes that are marked as not supported in XML are not handled by the skins provided with ColdFusion MX. They are, however, included in the generated XML.

Usage

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

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

You can use the keyboard to access and select dates from a datefield Flash input: press Tab to go to the field and press the Spacebar to open the menu. Use the Up, Down, Left, and Right Arrow keys to change the selected date. Use the Home and End keys to reach the first and last enabled date in a month, respectively. Use the Page Up and Page Down keys to reach the previous and next month, respectively.

Note: To clear a datefield entry, select the field to open the menu, and click the selected date.

For more information, see cfform. For information on using JavaScript regular expressions with this tag, see Chapter 27, “Building Dynamic Forms with cfform Tags” in ColdFusion MX Developer’s Guide.

Validation

The following sections describe how to do validation in cfinput tags.

Validation methods ColdFusion provides four methods of validation of cfinput text and password fields.

You can specify one or a combination of the following in the validateAt attribute:

- onSubmit The form page on the browser includes JavaScript functions that perform validation before the form is submitted to the server. In Flash format forms, this option is identical to onBlur.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Req/Opt; Formats</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>enabled</td>
<td>Optional; Flash</td>
<td>Yes</td>
<td>Boolean value specifying whether the control is enabled. A disabled control appears in light gray. The inverse of the disabled attribute.</td>
</tr>
<tr>
<td>visible</td>
<td>Optional; Flash</td>
<td>Yes</td>
<td>Boolean value specifying whether to show the control. Space that would be occupied by an invisible control is blank.</td>
</tr>
<tr>
<td>tooltip</td>
<td>Optional; Flash</td>
<td></td>
<td>Text to display when the mouse pointer hovers over the control.</td>
</tr>
<tr>
<td>height</td>
<td>Optional; see Description</td>
<td></td>
<td>Applies to most Flash types, HTML image type on some browsers. The height of the control, in pixels. The displayed height might be less than the specified size.</td>
</tr>
<tr>
<td>width</td>
<td>Optional; see Description</td>
<td></td>
<td>Applies to most Flash types, HTML image type on some browsers. The width of the control, in pixels. For Flash forms, ColdFusion ignores this attribute if you also specify a size attribute value.</td>
</tr>
</tbody>
</table>
• onBlur  In HTML format the form page on the browser includes JavaScript functions that perform validation when the field loses the focus. In Flash format, the attribute is equivalent to onSubmit. OnBlur validation uses the same algorithms as onSubmit validation. OnBlur validation was added in ColdFusion MX 7.

• onServer  ColdFusion performs the validation on the server. Some onServer algorithms vary from the onSubmit algorithms. OnServer Date and Time validation allow more patterns than onSubmit validation. OnServer validation was added in ColdFusion MX 7, and automatically generates hidden fields to support the validation.

You can also omit a validate attribute and specify the type of validation for the field in a separate hidden form field. This form of validation is equivalent to onServer validation, but it allows you to specify separate messages for each validation that you do on the field. It is backward compatible with previous ColdFusion releases. For more information on hidden form field validation, see cfform and “Validating form data using hidden fields” in Chapter 28, “Validating form data using hidden fields,” in ColdFusion MX Developer’s Guide.

**validation types**  You can use the following values in the validate attribute to specify input validation for all validation methods. Most attributes apply only to password or text fields. You can specify multiple validation types in a comma-delimited list, but only some combinations are meaningful.

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>date</td>
<td>If validateAt=&quot;onServer&quot;, allows any date format that returns True in the IsDate function; otherwise, same as USdate.</td>
</tr>
<tr>
<td>USdate</td>
<td>A US date of the format mm/dd/yy mm-dd-yy or mm.dd.yy, with 1-2 digit days and months, 1-4 digit years.</td>
</tr>
<tr>
<td>eurodate</td>
<td>A date of the format dd/mm/yy, with 1-2 digit days and months, 1-4 digit years. The format can use /, -, or . characters as delimiters.</td>
</tr>
<tr>
<td>time</td>
<td>Time format hh:mm:ss</td>
</tr>
<tr>
<td>float or numeric</td>
<td>A number; allows integers.</td>
</tr>
<tr>
<td>integer</td>
<td>An integer.</td>
</tr>
<tr>
<td>range</td>
<td>A numeric range.</td>
</tr>
<tr>
<td>boolean</td>
<td>A value that can be converted to a Boolean value: Yes, No, True, False, or a number.</td>
</tr>
<tr>
<td>telephone</td>
<td>Standard U.S. telephone formats. Allows an initial 1 long-distance designator and up to 5-digit extensions, optionally starting with x.</td>
</tr>
<tr>
<td>zipcode</td>
<td>U.S. 5- or 9-digit ZIP code format ######-####. The separator can be a hyphen (-) or a space.</td>
</tr>
<tr>
<td>creditcard</td>
<td>Strips blanks and dashes; verifies number using mod10 algorithm. Number must have 13-16 digits.</td>
</tr>
<tr>
<td>ssn or social_security_number</td>
<td>US, Social Security number format, ####-####-####. The separator can be a hyphen (-) or a space.</td>
</tr>
</tbody>
</table>
**Validation differences** The preceding table describes the general validation behavior. The underlying validation code must differ depending on the validation method and the form type. As a result, the algorithms used vary in some instances, including the following:

- The validation algorithms used for date/time values varies between onSubmit/OnBlur and OnServer.
- The algorithms used for onSubmit/OnBlur validation in Flash vary from those used for HTML/XML format, and generally follow simpler rules.

The table describes the onSubmit/OnBlur behavior in HTML format. For detailed information on the OnServer validation algorithms, see “Data validation types” in Chapter 28, “Data validation types,” in *ColdFusion MX Developer’s Guide*.

For more information on validation, including discussions of the advantages and disadvantages of different validation types, see Chapter 28, “Validating Data” in *ColdFusion MX Developer’s Guide*.

### Masking input data

The *mask* attribute controls the format of data that can be entered into a field. You can combine masking and validation on a field.

- In HTML and Flash form format the mask can control the format of data entered into a *text* field.
- In Flash format, the mask can also control the format of the date chosen using the *datefield* input control.

In text fields, ColdFusion automatically inserts any literal mask characters, such as - characters in telephone numbers. Users type only the variable part of the field.

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>email</td>
<td>A valid e-mail address of the form <a href="mailto:name@server.domain">name@server.domain</a>. ColdFusion validates the format only; it does not check that entry is a valid active e-mail address.</td>
</tr>
<tr>
<td>URL</td>
<td>A valid URL pattern; supports http, https, ftp file, mailto, and news URLs.</td>
</tr>
<tr>
<td>guid</td>
<td>A unique identifier that follows the Microsoft/DCE format, xxxxxxx-xxxx-xxxx-xxxx-xxxx-xxxx-xxxx-xxxx, where x is a hexadecimal number.</td>
</tr>
<tr>
<td>uuid</td>
<td>A universally unique identifier (UUID) that follows the ColdFusion format, xxxxxxx-xxxx-xxxx-xxxxxxxxxxxxxxxx, where x is a hexadecimal number.</td>
</tr>
<tr>
<td>maxlength</td>
<td>Limits the input to a maximum number of characters.</td>
</tr>
<tr>
<td>noblanks</td>
<td>Does not allow fields that consist only of blanks.</td>
</tr>
<tr>
<td>regex or regular_expression</td>
<td>Matches input against the <em>pattern</em> attribute. Valid in HTML and XML format only; ignored in Flash format.</td>
</tr>
<tr>
<td>SubmitOnce</td>
<td>Used only with submit and image types; prevents the user from submitting the same form multiple times before until the next page loads (for example, submitting an order a second time before getting the first order confirmation). Valid in HTML and XML format only; ignored in Flash format.</td>
</tr>
</tbody>
</table>
The following pattern enforces entry of a part number of the format EB-1234-c1-098765, where the user starts the entry by typing the first numeric character, such as 3. ColdFusion fills in the preceding EA prefix and all - characters. The user must enter four numbers, followed by two alphanumeric characters, followed by six numbers.

```coldfusion
<cfinput type="text" name="newPart" mask="EB-9999-XX-999999" />
```

**Note:** To force a pattern to be all-uppercase or all-lowercase, use the ColdFusion **UCase** or **LCase** functions in the action page.

For tags with **type="datefield"** (and **cfcalendar** tags), the number of pattern characters determines the format of the output, as follows:

<table>
<thead>
<tr>
<th>Mask</th>
<th>Pattern</th>
</tr>
</thead>
<tbody>
<tr>
<td>D</td>
<td>Single- or double-digit day of month, such as 1 or 28</td>
</tr>
<tr>
<td>DD</td>
<td>Double-digit day of month, such as 01 or 28</td>
</tr>
<tr>
<td>M</td>
<td>Single- or double-digit month, such as 1 or 12</td>
</tr>
<tr>
<td>MM</td>
<td>Double-digit month, such as 01 or 12</td>
</tr>
<tr>
<td>MMM</td>
<td>Abbreviated month name, such as Jan or Dec</td>
</tr>
<tr>
<td>MMMM</td>
<td>Full month name, such as January or December</td>
</tr>
<tr>
<td>YY</td>
<td>Two-character year, such as 05</td>
</tr>
<tr>
<td>YYYY</td>
<td>Four-character year, such as 2005</td>
</tr>
<tr>
<td>E</td>
<td>Single-digit day of week, such as 1 or 7</td>
</tr>
<tr>
<td>EEE</td>
<td>Abbreviated day of week name, such as Mon or Sun</td>
</tr>
<tr>
<td>EEEE</td>
<td>Full month day of week name, such as Monday or Sunday</td>
</tr>
</tbody>
</table>

The following pattern specifies that the Flash forms sends the date selected using a **datefield** input control to ColdFusion as text in the format 04/29/2004:

```coldfusion
<cfinput name="startDate" type="datefield" label="date:" mask="mm/dd/yyyy"/>
```

**Flash form data binding**

The **bind** attribute lets you populate form fields using the contents of other form fields. To specify text from another field in a **cftextarea** bind attribute, use the following format:

```coldfusion
{sourceTagName.text}
```

For example, the following line uses the values from the **firstName** and **lastName** fields to construct an email address. (The user can change or replace this value with a typed entry.)

```coldfusion
<cfinput type="text" name="email" label="email"
    bind="[firstName.text].(lastName.text)@mm.com">
```

**Example**

```coldfusion
<!--- This example shows the use of cfinput within a cfform to ensure simple validation of text items. --->
<cfform action = "cfinput.cfm">
<cfform action = "cfinput.cfm">
<!---- Phone number validation. --->
```
Phone Number Validation (enter a properly formatted phone number): <br>
<cfinput
type = "Text" name = "MyPhone"
message = "Enter telephone number, formatted xxx-xxx-xxxx (e.g. 617-761-2000)"
validate = "telephone" required = "yes">
<font size = -1 color = red>Required</font>
<!--- Zip code validation. --->
<p>Zip Code Validation (enter a properly formatted zip code):<br>
<cfinput
type = "Text" name = "MyZip"
message = "Enter zip code, formatted xxxxx or xxxxx-xxxx"
validate = "zipcode" required = "yes">
<font size = -1 color = red>Required</font>
<!--- Range validation. --->
<p>Range Validation (enter an integer from 1 to 5): <br>
<cfinput
type = "Text" name = "MyRange" range = "1,5"
message = "You must enter an integer from 1 to 5"
validate = "integer" required = "no">
<!--- Date validation. --->
<p>Date Validation (enter a properly formatted date):<br>
<cfinput
type = "Text" name = "MyDate"
message = "Enter a correctly formatted date (dd/mm/yy)"
validate = "date" required = "no">
<input
type = "Submit" name = ""
value = "send my information">
cfinsert

Description

Inserts records in data sources from data in a ColdFusion form or form Scope.

Category

Database manipulation tags

Syntax

```xml
<cfinsert
  dataSource = "ds_name"
  tableName = "tbl_name"
  tableOwner = "owner"
  tableQualifier = "tbl_qualifier"
  username = "username"
  password = "password"
  formFields = "formfield1, formfield2, ...">
```

See also

cfprocparam, cfprocresult, cfquery, cfqueryparam, cfstoredproc, cftransaction, cfupdate

History

ColdFusion MX: Deprecated the connectString, dbName, dbServer, dbtype, provider, and providerDSN attributes. They do not work, and might cause an error, in releases later than ColdFusion 5.

Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Req/Opt</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dataSource</td>
<td>Required</td>
<td>Data source; contains table.</td>
<td></td>
</tr>
<tr>
<td>tableName</td>
<td>Required</td>
<td>Table in which to insert form fields. ORACLE drivers: must be uppercase. Sybase driver: case-sensitive. Must be the same case used when table was created</td>
<td></td>
</tr>
<tr>
<td>tableOwner</td>
<td>Optional</td>
<td>For data sources that support table ownership (such as SQL Server, Oracle, and Sybase SQL Anywhere), use this field to specify the owner of the table.</td>
<td></td>
</tr>
<tr>
<td>tableQualifier</td>
<td>Optional</td>
<td>For data sources that support table qualifiers, use this field to specify qualifier for table. The purpose of table qualifiers varies among drivers. For SQL Server and Oracle, qualifier refers to name of database that contains table. For Intersolv dBASE driver, qualifier refers to directory where DBF files are located.</td>
<td></td>
</tr>
<tr>
<td>username</td>
<td>Optional</td>
<td>Overrides username specified in ODBC setup.</td>
<td></td>
</tr>
</tbody>
</table>
<!--- This example shows how to use cfinsert instead of cfquery to put data in a datasource. --->
<!--- If form.POSTED exists, we insert new record, so begin cfinsert tag. --->
<cfif IsDefined("form.posted")>
  <cfinsert dataSource="cfdocexamples"
          tableName="Comments"
          formFields="Email,FromUser,Subject,MessText,Posted">
      <h3>Your record was added to the database.</h3>
  </cfif>
</cfif>

<!--- Use a query to show the existing state of the database. --->
<cfquery name="GetComments" dataSource="cfdocexamples">
  SELECT CommentID, EMail, FromUser, Subject, CommtType, MessText, Posted, Processed FROM Comments
</cfquery>

<html>
<head></head>
<h3>cfinsert Example</h3>
<p>First, show a list of the comments in the cfdocexamples datasource.
<!--- show all the comments in the db --->
<table>
<tr>
  <td>From User</td><td>Subject</td><td>Comment Type</td>
  <td>Message</td><td>Date Posted</td>
</tr>
<cfoutput query="GetComments">
  <tr>
    <td valign=top><a href="mailto:#Email#">#FromUser#</A></td>
    <td valign=top>#Subject#</td>
    <td valign=top>#CommtType#</td>
    <td valign=top>#Message#</td>
    <td valign=top>#Date Posted#</td>
  </tr>
</cfoutput>
</table>
</html>
Next, we’ll offer the opportunity to enter a comment:

```html
<form action = "cfinsert.cfm" method = "post">
<pre>
Email: <input type = "Text" name = "email">
From: <input type = "Text" name = "fromUser">
Subject:<input type = "Text" name = "subject">
Message:<textarea name = "MessText" COLS = "40" ROWS = "6"></textarea>
Date Posted: <cfoutput>#DateFormat(Now())#</cfoutput>
<input type = "hidden"
    name = "posted" value = "<cfoutput>#Now()#"/>
</pre>
<input type = "Submit" name = "" value = "insert my comment">  
</form>
```
cfinvoke

Description

Does either of the following:

- Invokes a component method from within a ColdFusion page or component.
- Invokes a web service.

This tag works as follows:

- Transiently instantiates a component or web service and invokes a method on it.
- Invokes a method on an instantiated component or web service.

This tag can pass parameters to a method in the following ways:

- With the cfinvokeargument tag
- As named attribute-value pairs, one attribute per parameter
- As a structure, in the argumentCollection attribute

Category

Extensibility tags

Syntax

Syntax 1

<!--- This syntax invokes a method of a component. --->
<cfinvoke
  component = "component name or reference"
  method = "method name"
  returnVariable = "variable name"
  argumentCollection = "argument collection"
  ...>
  OR

Syntax 2

<!--- This syntax can invoke a method of a component only from within the component. --->
<cfinvoke
  method = "method name"
  returnVariable = "variable name"
  argumentCollection = "argument collection"
  ...>
  OR

Syntax 3

<!--- This syntax invokes a web service. --->
<cfinvoke
  webservice = "URLtoWSDL_location"
  method = "operation_name"
  username = user name"
  password = "password"
  timeout = "request timeout in seconds"
  ...>
proxyServer = "WSDL proxy server URL
proxyPort = "port on proxy server"
proxyUser = "user id for proxy server"
proxyPassword = "password for proxy server"
servicePort = "WSDL port name"
inputParam1 = "value1"
inputParam2 = "value2"
...
returnVariable = "var_name"
...>
OR

Syntax 4A

<!--- This syntax invokes a component. This syntax shows instantiation with the cfobject tag. This cfinvoke syntax applies to instantiating a component with the cfobject tag and to instantiating a component with the CreateObject function. --->
<cfobject
component = "component name"
name = "mystringname for instantiated object">
<cfinvoke
component = "#mystringname for instantiated component#"
method = "method name"
returnVariable = "variable name"
argumentCollection = "argument collection"
...>
OR

Syntax 4B

<!--- This syntax invokes a web service. This syntax shows instantiation with the cfobject tag. This cfinvoke syntax applies to instantiating a web service with the cfobject tag and to instantiating a web service with the CreateObject function. --->
<cfobject
webservice = "web service name"
name = "mystringname for instantiated object"
method = "operation_name">
<cfinvoke
webservice = "#mystringname for instantiated web service#"
timeout = "request timeout in seconds"
proxyServer = "WSDL proxy server url"
proxyPort = "numeric port on proxy server"
proxyUser = "string user id for proxy server"
servicePort = "WSDL port name"
proxyPassword = "string user password for proxy server"
>

See also

cfargument, cfcomponent, cffunction, cfinvokeargument, cfobject, cfproperty, cfreturn
History

ColdFusion MX 7: Added the `servicePort` attribute.

ColdFusion MX 6.1: Added the following attributes: `timeout`, `proxyServer`, `proxyPort`, `proxyUser`, and `proxyPassword`.

ColdFusion MX: Added this tag.

Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Req/Opt</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>component</td>
<td>See</td>
<td>Usage.</td>
<td>String or component object; a reference to a component, or component to instantiate.</td>
</tr>
<tr>
<td>method</td>
<td>See</td>
<td>Usage.</td>
<td>Name of a method. For a web service, the name of an operation.</td>
</tr>
<tr>
<td>returnVariable</td>
<td>Optional</td>
<td></td>
<td>Name of a variable for the invocation result.</td>
</tr>
<tr>
<td>argumentCollection</td>
<td>Optional</td>
<td></td>
<td>Name of a structure; associative array of arguments to pass to the method.</td>
</tr>
<tr>
<td>username</td>
<td>Optional</td>
<td></td>
<td>Overrides username specified in <code>Administrator &gt; Web Services</code>.</td>
</tr>
<tr>
<td>password</td>
<td>Optional</td>
<td></td>
<td>Overrides password specified in <code>Administrator &gt; Web Services</code>.</td>
</tr>
<tr>
<td>webservice</td>
<td>Required</td>
<td></td>
<td>The URL of the WSDL file for the web service.</td>
</tr>
<tr>
<td>timeout</td>
<td>Optional</td>
<td></td>
<td>The timeout for the web service request, in seconds</td>
</tr>
<tr>
<td>proxyServer</td>
<td>Optional</td>
<td>http.proxyHost system property, if any.</td>
<td>The proxy server required to access the webservice URL.</td>
</tr>
<tr>
<td>proxyPort</td>
<td>Optional</td>
<td>http.proxyPort system property, if any.</td>
<td>The port to use on The proxy server.</td>
</tr>
<tr>
<td>proxyUser</td>
<td>Optional</td>
<td>http.proxyUser system property, if any.</td>
<td>The user ID to send to the proxy server.</td>
</tr>
<tr>
<td>proxyPassword</td>
<td>Optional</td>
<td>http.proxyPassword system property, if any.</td>
<td>The user’s password on the proxy server.</td>
</tr>
</tbody>
</table>
If you do not specify any of the proxy attributes, and a corresponding system property is set (typically in the JVM startup arguments) ColdFusion uses the system property value.

**Usage**

The following table shows when you can use each attribute:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Req/Opt</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>servicePort</td>
<td>Optional</td>
<td>First port found in the WSDL</td>
<td>The port name for the web service. This value is case-sensitive and corresponds to the port element’s name attribute under the service element.</td>
</tr>
<tr>
<td>input_params ...</td>
<td></td>
<td>Input parameters. For each named input parameter specify paramName=paramValue.</td>
<td></td>
</tr>
</tbody>
</table>

**Note:**

If the component attribute specifies a component name, the component with the corresponding name is instantiated, the requested method is invoked, and then the component instance is immediately destroyed. If the attribute contains a reference to an instantiated component object, no instantiation or destruction of the component occurs.

On UNIX systems, ColdFusion searches first for a file with a name that matches the specified component name, but is all lower case. If it does not find the file, it looks for a file name that matches the component name exactly, with the identical character casing.
Method arguments can be passed in any of the following ways. If an argument is passed in more than one way with the same name, this order of precedence applies:

1. Using the `cfinvokeargument` tag
2. Passing directly as attributes of the `cfinvoke` tag (they cannot have the same name as a registered `cfinvoke` attribute: method, component, webservice, returnVariable)
3. Passing as struct keys, using the `argumentCollection` attribute

For example, the `params` struct contains three keys: a=1, b=1, c=1. The following call is evaluated as if the arguments were passed to the method in the order a=3, b=2, c=1:

```xml
<cfinvoke ... a=2 b=2 argumentCollection=params>
    <cfinvokeargument name="a" value="3">
</cfinvoke>
</cfinvoke>
```

**Note:** The following `cfinvoke` tag attribute names are reserved; they cannot be used for argument names: component, method, argumentCollection, and result.

**Example 1**

This example uses Syntax 1.

```xml
<!---- Immediate instantiation and destruction. --->
<cfinvoke
    component="nasdaq.quote"
    method="getLastTradePrice"
    returnVariable="res">
    <cfinvokeargument
        name="symbol"
        value="macr">
</cfinvoke>
</cfoutput>#res#</cfoutput>
```

**Example 2**

This example uses Syntax 1.

```xml
<!---- Passing the arguments using argumentCollection. --->
<cfset args = StructNew()>
<cfset args.symbol = "macr">  
<cfinvoke
    component="nasdaq.quote"
    method="getLastTradePrice"
    argumentCollection="#args#"
    returnVariable="res">
</cfoutput>#res#</cfoutput>
```

**Example 3**

This example uses Syntax 2.

```xml
<!---- Called only from within a component, MyComponent.---->
<cfinvoke
    method = "a method name of MyComponent"
    returnVariable = "variable name">
```
Example 4

This example uses Syntax 3.

<!--- Using cfinvoke to consume a web service using a ColdFusion component.--->
<cfinvoke
  webservice="http://www.xmethods.net/sd/2001/TemperatureService.wsdl"
  method="getTemp"
  returnvariable="aTemp">
  <cfinvokeargument name="zipcode" value="55987"/>
</cfinvoke>
<cfoutput>The temperature at zip code 55987 is #aTemp#</cfoutput>

For more information on web services, see Chapter 36, "Using Web Services" in ColdFusion MX Developer's Guide.

Example 5

This example uses Syntax 4A.

<!--- Separate instantiation and method invocation; useful for multiple invocations using different methods or values. --->
<cfobject
  name="quoteService"
  component="nasdaq.quote">
  <cfinvoke
    component="#quoteService#"
    method="getLastTradePrice"
    symbol="macr"
    returnVariable="res_macr"/>
  <cfoutput>#res#</cfoutput>
  <cfinvoke
    component="#quoteService#"
    method="getLastTradePrice"
    symbol="mot"
    returnVariable="res_mot"/>
  <cfoutput>#res#</cfoutput>
</cfobject>
cfinvokeargument

**Description**

Passes the name and value of a parameter to a component method or a web service. This tag is used within the `cfinvoke` tag.

**Category**

*Extensibility tags*

**Syntax**

```
<cfinvokeargument
  name="argument name"
  value="argument value"
  omit = "yes" or "no">
```

**See also**

*cfargument, cfcomponent,cffunction, cfinvoke, cfobject, cfproperty, cfreturn*

**History**

ColdFusion MX 7: Added the `omit` attribute.

ColdFusion MX: Added this tag.

**Attributes**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Req/Opt</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Required</td>
<td></td>
<td>Argument name.</td>
</tr>
<tr>
<td>value</td>
<td>Required</td>
<td></td>
<td>Argument value.</td>
</tr>
<tr>
<td>omit</td>
<td>Optional</td>
<td>&quot;no&quot;</td>
<td>Enables you to omit a parameter when invoking a web service. It is an error to specify <code>omit=&quot;yes&quot;</code> if the <code>cfinvoke webservice</code> attribute is not specified. &quot;yes&quot;: omit this parameter when invoking a web service. &quot;no&quot;: do not omit this parameter when invoking a web service.</td>
</tr>
</tbody>
</table>

**Usage**

You can have multiple `cfinvokeargument` tags in a `cfinvoke` tag body.

You can use `cfinvokeargument` tag to dynamically determine the arguments to be passed. For example, you can use conditional processing to determine the argument name, or you can use a `cff` tag to determine whether to execute the `cfinvokeargument` tag.

If you are invoking a web service, you can omit a parameter by setting the `omit` attribute to "yes". If the WSDL specifies that the argument is nillable, ColdFusion MX sets the associated argument to null. If the WSDL specifies minoccurs=0, ColdFusion MX omits the argument from the WSDL.
Example1
<cfinvoke
  component="nasdaq.quote"
  method="getLastTradePrice"
  returnVariable="res">
  <cfinvokeargument
    name="symbol" value="mot">
  </cfinvokeargument>
  <cfinvokeargument
    name="symbol" value="macr">
  </cfinvokeargument>
</cfinvoke>
<cfoutput>#res#</cfoutput>

Example2
<cfinvoke
  webservice = "http://www.xmethods.net/sd/2001/BabelFishService.wsdl"
  method = "BabelFish"
  returnVariable = "varName">
  <cfinvokeargument
    name="translationmode" value="en_es">
  </cfinvokeargument>
  <cfinvokeargument
    name="sourcedata" value="Hello world, friend">
  </cfinvokeargument>
</cfinvoke>
<cfoutput>#varName#</cfoutput>
**cfldap**

**Description**

Provides an interface to a Lightweight Directory Access Protocol (LDAP) directory server, such as the Netscape Directory Server.

**Category**

Internet Protocol tags

**Syntax**

```<cfldap
    server = "server_name"
    port = "port_number"
    username = "name"
    password = "password"
    action = "action"
    name = "name"
    timeout = "seconds"
    maxRows = "number"
    start = "distinguished_name"
    scope = "scope"
    attributes = "attribute, attribute"
    returnAsBinary = "column_name, column_name"
    filter = "filter"
    sort = "attribute[, attribute]..."
    sortControl = "nocase" and/or "desc" or "asc"
    dn = "distinguished_name"
    startRow = "row_number"
    modifyType = "replace" or "add" or "delete"
    rebind = "yes" or "no"
    referral = "number_of_allowed_hops"
    secure = "multi_field_security_string"
    separator = "separator_character"
    delimiter = "delimiter_character">
```

**See also**

cfftp, cfhttp, cfmail, cfmailparam, cfpop, Chapter 23, “Managing LDAP Directories” in ColdFusion MX Developer’s Guide

**History**

ColdFusion MX 7: Added the returnAsBinary attribute.

ColdFusion MX:

- Changed the name attribute behavior: this tag validates the query name in the name attribute.
- Changed sorting behavior: this tag does not support client-side sorting of query results. (It supports server-side sorting; use the sort and sortcontrol attributes.)
- Changed how results are sorted: server-side sorting results might be sorted slightly differently than in ColdFusion 5. If you attempt a sort against a server that does not support it, ColdFusion MX throws an error.
• Deprecated the `filterfile` attribute. It might not work, and might cause an error, in later releases.

**Attributes**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Req/Opt</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>server</td>
<td>Required</td>
<td></td>
<td>Host name or IP address of LDAP server.</td>
</tr>
<tr>
<td>port</td>
<td>Optional</td>
<td>389</td>
<td>Port.</td>
</tr>
<tr>
<td>username</td>
<td>Required if <code>secure = &quot;CFSSL_BASIC&quot;</code></td>
<td>(anonymous)</td>
<td>User ID.</td>
</tr>
<tr>
<td>password</td>
<td>Required if <code>secure = &quot;CFSSL_BASIC&quot;</code></td>
<td></td>
<td>Password that corresponds to user name.</td>
</tr>
<tr>
<td>action</td>
<td>Required</td>
<td>query</td>
<td>• <em>query</em>: returns LDAP entry information only. Requires <code>name</code>, <code>start</code>, and <code>attributes</code> attributes.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• <em>add</em>: adds LDAP entries to LDAP server. Requires <code>attributes</code> attribute.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• <em>modify</em>: modifies LDAP entries, except distinguished name <code>dn</code> attribute, on LDAP server. Requires <code>dn</code>. See <code>modifyType</code> attribute.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• <em>modifyDN</em>: modifies distinguished name <code>dn</code> attribute for LDAP entries on LDAP server. Requires <code>dn</code>.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• <em>delete</em>: deletes LDAP entries on an LDAP server. Requires <code>dn</code>.</td>
</tr>
<tr>
<td>name</td>
<td>Required if <code>action = &quot;Query&quot;</code></td>
<td></td>
<td>Name of LDAP query. The tag validates the value.</td>
</tr>
<tr>
<td>timeout</td>
<td>Optional</td>
<td>60000</td>
<td>Maximum length of time, in milliseconds, to wait for LDAP processing.</td>
</tr>
<tr>
<td>maxRows</td>
<td>Optional</td>
<td></td>
<td>Maximum number of entries for LDAP queries.</td>
</tr>
<tr>
<td>start</td>
<td>Required if <code>action = &quot;Query&quot;</code></td>
<td></td>
<td>Distinguished name of entry to be used to start a search.</td>
</tr>
<tr>
<td>scope</td>
<td>Optional</td>
<td>oneLevel</td>
<td>Scope of search, from entry specified in <code>start</code> attribute for <code>action = &quot;Query&quot;</code>.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• oneLevel: entries one level below entry.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• base: only the entry.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• subtree: entry and all levels below it.</td>
</tr>
<tr>
<td>Attribute</td>
<td>Req/Opt</td>
<td>Default</td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>---------</td>
<td>---------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>attributes</td>
<td>Required if action = &quot;Query&quot;, &quot;Add&quot;, &quot;ModifyDN&quot;, or &quot;Modify&quot;</td>
<td></td>
<td>For queries: comma-delimited list of attributes to return. For queries, to get all attributes, specify &quot;*&quot;. If action = &quot;add&quot; or &quot;modify&quot;, you can specify a list of update columns. Separate attributes with a semicolon. If action = &quot;ModifyDN&quot;, ColdFusion passes attributes to the LDAP server without syntax checking.</td>
</tr>
<tr>
<td>returnAsBinary</td>
<td>Optional</td>
<td></td>
<td>A comma-delimited list of columns that are to be returned as binary values.</td>
</tr>
<tr>
<td>filter</td>
<td>Optional</td>
<td>&quot;objectclass = *&quot;</td>
<td>Search criteria for action = &quot;query&quot;. List attributes in the form: &quot;(attribute operator value)&quot; For example: &quot;(sn = Smith)&quot;</td>
</tr>
<tr>
<td>sort</td>
<td>Optional</td>
<td></td>
<td>Attribute(s) by which to sort query results. Use a comma delimiter.</td>
</tr>
<tr>
<td>sortControl</td>
<td>Optional</td>
<td>asc</td>
<td>• nocase: case-insensitive sort. • asc: ascending (a to z) case-sensitive sort. • desc: descending (z to a) case-sensitive sort. You can enter a combination of sort types; for example, sortControl = &quot;nocase, asc&quot;.</td>
</tr>
<tr>
<td>dn</td>
<td>Required if action = &quot;Add&quot;, &quot;Modify&quot;, &quot;ModifyDN&quot;, or &quot;delete&quot;</td>
<td></td>
<td>Distinguished name, for update action; for example, &quot;cn = Bob Jensen, o = Ace Industry, c = US&quot;</td>
</tr>
<tr>
<td>startRow</td>
<td>Optional</td>
<td>1</td>
<td>Used with action = &quot;query&quot;. First row of LDAP query to insert into a ColdFusion query.</td>
</tr>
<tr>
<td>modifyType</td>
<td>Optional</td>
<td>replace</td>
<td>How to process an attribute in a multi-value list: • add: appends it to any attributes. • delete: deletes it from the set of attributes. • replace: replaces it with specified attributes. You cannot add an attribute that is already present or that is empty.</td>
</tr>
<tr>
<td>rebind</td>
<td>Optional</td>
<td>no</td>
<td>• yes: attempts to rebind referral callback and reissue query by referred address using original credentials. • no: referred connections are anonymous.</td>
</tr>
<tr>
<td>referral</td>
<td>Optional</td>
<td></td>
<td>Integer. Number of hops allowed in a referral. A value of 0 disables referred addresses for LDAP; no data is returned.</td>
</tr>
</tbody>
</table>
If you use the query action, `cfldap` creates a query object, allowing access to information in the query variables, as follows:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Req/Opt</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
</table>
| secure          | Optional|         | Security to employ, and required information. One option:  
|                 |         |         | • CFSSL_BASIC |
|                 |         |         | "CFSSL_BASIC" provides V2 SSL encryption and server authentication. |
| separator       | Optional| . [comma] | Delimiter to separate attribute values of multi-value attributes. Used by `query`, `add`, and `modify` actions, and by `cfldap` to output multi-value attributes.  
|                 |         |         | For example, if $ (dollar sign), the attributes attribute could be "objectclass=top$person", where the first value of objectclass is top, and the second value is person. This avoids confusion if values include commas. |
| delimiter       | Optional| ; [semicolon] | Separator between attribute name-value pairs. Use this attribute if either of these situations exist:  
|                 |         |         | • the attributes attribute specifies more than one item  
|                 |         |         | • an attribute contains the default delimiter (semicolon). For example: `mgrpmsgrejecttext;lang-en` |
|                 |         |         | Used by `query`, `add`, and `modify` actions, and by `cfldap` to output multivalue attributes.  
|                 |         |         | For example, if $ (dollar sign), you could specify "cn=Double Tree Inn$street=1111 Elm; Suite 100", where the semicolon is part of the street value. |

Usage

If you use the query action, `cfldap` creates a query object, allowing access to information in the query variables, as follows:

<table>
<thead>
<tr>
<th>Variable name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>queryname.recordCount</code></td>
<td>Number of records returned by query</td>
</tr>
<tr>
<td><code>queryname.currentRow</code></td>
<td>Current row of query that <code>cfoutput</code> is processing</td>
</tr>
<tr>
<td><code>queryname.columnList</code></td>
<td>Column names in query</td>
</tr>
</tbody>
</table>

If you use the `security="CFSSL_BASIC"` option, ColdFusion determines whether to trust the server by comparing the server's certificate with the information in the jre/lib/security/cacerts keystore of the JRE used by ColdFusion MX. The ColdFusion MX default cacerts file contains information about many certificate granting authorities. If you must update the file with additional information, you can use the keytool utility in the ColdFusion jre/bin directory to import certificates that are in X.509 format. For example, enter the following:

```
keytool -import -keystore cacerts -alias ldap -file ldap.crt -keypass bl19mq
```
Then restart ColdFusion MX. The keytool utility initial keypass password is "change it". For more information on using the keytool utility, see the Sun JDK documentation.

Characters that are illegal in ColdFusion can be used in LDAP attribute names. As a result, the cfldap tag could create columns in the query result set whose names contain illegal characters and are, therefore, inaccessible in CFML. In ColdFusion, illegal characters are automatically mapped to the underscore character; therefore, column names in the query result set might not exactly match the names of the LDAP attributes.

For usage examples, see ColdFusion MX Developer's Guide.

Example

Provides an interface to LDAP directory servers. The example uses the University of Connecticut public LDAP server. For more public LDAP servers, see a href="http://www.emailman.com" http://www.emailman.com/a.".</p><p>Enter a name and search the public LDAP resource. An asterisk before or after the name acts as a wildcard.</p><cif IsDefined("form.name")>
<!--- If form.name exists, the form was submitted; run the query. --->
<!--- Check to see that there is a name listed. --->
<!--- Make the LDAP query. --->
cfldap
  server = "ldap.uconn.edu"
  action = "query"
  name = "results"
  start = "dc=uconn,dc=edu"
  filter = "cn=#name#"
  attributes = "cn,o,title,mail,telephonenumber"
  sort = "cn ASC"
</cifldap>
</cif>
</center><table border = 0 cellspacing = 2 cellpadding = 2>
<tr>
  <th colspan = 5>
    <cfoutput>#results.recordCount# matches found</cfoutput></TH>
</tr>
<tr>
  <th><font size = "-2">Name</font></th>
  <th><font size = "-2">Organization</font></th>
  <th><font size = "-2">Title</font></th>
  <th><font size = "-2">E-Mail</font></th>
  <th><font size = "-2">Phone</font></th>
</tr>
<cfoutput query = "results">
  <tr>
    <td><font size = "-2"><A href = "mailto:#mail#">#mail#</A></font></td>
  </tr>
</cfoutput>
</table>
</form>
<form action="@cgi.script_name@" method="POST">
<p>Enter a name to search in the database.
</p>
<input type="Text" name="name">
<input type="Submit" value="Search" name="">
</form>
**cflocation**

**Description**

Stops execution of the current page and opens a ColdFusion page or HTML file.

**Category**

*Flow-control tags, Page processing tags*

**Syntax**

```
<cflocation
  url = "url"
  addToken = "yes" or "no">
```

**See also**

`cfabort`, `cfbreak`, `cfexecute`, `cfexit`, `cfif`, `cfloop`, `cfswitch`, `cfthrow`, `cftry`

**Attributes**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Req/Opt</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>url</td>
<td>Required</td>
<td></td>
<td>URL of HTML file or CFML page to open.</td>
</tr>
</tbody>
</table>
| addToken  | Optional|         | clientManagement must be enabled (see `cfapplication` on page 40).  
  • yes: appends client variable information to URL.  
  • no |

**Usage**

You might write a standard message or response in a file, and call it from several applications. You could use this tag to redirect the user's browser to the standard file.

This tag has no effect if you use it after the `cfflush` tag on a page.

**Example**

```
<h3>cflocation Example</h3>
<p>This tag redirects the browser to a web resource; normally, you would use this tag to go to a CF page or an HTML file on the same server. The addToken attribute lets you send client information to the target page.</p>

<p>If you remove the comments, this code redirects you to CFDOCS home page:</p>

<!--- <cflocation url = "http://localhost:8500/cfdocs/dochome.htm"
  addToken = "no"> --->

```
cflock

Description
Ensures the integrity of shared data. Instantiates the following kinds of locks:

**Exclusive**  Allows single-thread access to the CFML constructs in its body. The tag body can be executed by one request at a time. No other requests can start executing code within the tag while a request has an exclusive lock. ColdFusion issues exclusive locks on a first-come, first-served basis.

**Read-only**  Allows multiple requests to access CFML constructs within the tag body concurrently. Use a read-only lock only when shared data is read and not modified. If another request has an exclusive lock on shared data, the new request waits for the exclusive lock to be released.

Category
Application framework tags

Syntax
```
<cflock
   timeout = "timeout in seconds ",
   scope = "Application" or "Server" or "Session",
   name = "lockname",
   throwOnTimeout = "yes" or "no",
   type = "readOnly" or "exclusive ">
   <!--- CFML to be synchronized --->
</cflock>
```

See also
cfapplication, cfassociate, cfmodule. Chapter 15, “Using Persistent Data and Locking” in *ColdFusion MX Developer’s Guide*
Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Req/Opt</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>timeout</td>
<td>Required</td>
<td></td>
<td>Maximum length of time, in seconds, to wait to obtain a lock. If lock is obtained, tag execution continues. Otherwise, behavior depends on throwOnTimeout attribute value. If you set timeout=&quot;0&quot;, the timeout is determined by the &quot;Timeout Requests after x&quot; setting in the ColdFusion MX Administrator Settings page, if that setting is enabled. However, if the setting is not enabled, and you set timeout=&quot;0&quot;, ColdFusion can wait indefinitely to obtain the lock.</td>
</tr>
<tr>
<td>scope</td>
<td>Optional</td>
<td></td>
<td>Lock scope. Mutually exclusive with the name attribute. Lock name. Only one request in the specified scope can execute the code within this tag (or within any other cflock tag with the same lock scope) at a time.</td>
</tr>
<tr>
<td>name</td>
<td>Optional</td>
<td></td>
<td>Lock name. Mutually exclusive with the scope attribute. Only one request can execute the code within a cflock tag with a given name at a time. Cannot be an empty string. Permits synchronizing access to resources from different parts of an application. Lock names are global to a ColdFusion server. They are shared among applications and user sessions, but not clustered servers.</td>
</tr>
<tr>
<td>throwOnTimeout</td>
<td>Optional</td>
<td>yes</td>
<td>How timeout conditions are handled:</td>
</tr>
<tr>
<td>type</td>
<td>Optional</td>
<td>Exclusive</td>
<td>• readOnly: lets more than one request read shared data.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• exclusive: lets one request read or write shared data.</td>
</tr>
</tbody>
</table>

**Note:** Limit the scope of code that updates shared data structures, files, and CFXs. Exclusive locks are required to ensure the integrity of updates, but read-only locks are faster. In a performance-sensitive application, substitute read-only locks for exclusive locks where possible; for example, when reading shared data.

**Usage**

ColdFusion MX is a multithreaded server; it can process multiple page requests at a time. Use the cflock tag for these purposes:

- To ensure that modifications to shared data and objects made in concurrently executing requests occur sequentially.
- Around file manipulation constructs, to ensure that file updates do not fail because files are open for writing by other applications or tags.
- Around CFX invocations, to ensure that ColdFusion can safely invoke CFXs that are not implemented in a thread-safe manner. (This applies only to CFXs developed in C++.)
To work safely with ColdFusion, a C++ CFX that maintains and manipulates shared (global) data structures must be made thread-safe; however, this requires advanced knowledge. You can use a CFML custom tag wrapper around a CFX to make its invocation thread-safe.

When you display, set, or update variables in a shared scope, use the `scope` attribute to identify the scope as Server, Application or Session.

**Deadlocks**

A deadlock is a state in which no request can execute the locked section of a page. Once a deadlock occurs, neither user can break it, because all requests to the protected section of the page are blocked until the deadlock can be resolved by a lock timeout.

The `cflock` tag uses kernel level synchronization objects that are released automatically upon timeout and/or the abnormal termination of the thread that owns them. Therefore, while processing a `cflock` tag, ColdFusion never deadlocks for an infinite period of time. However, very large timeouts can block request threads for long periods, and radically decrease throughput. To prevent this, always use the minimum timeout value.

Another cause of blocked request threads is inconsistent nesting of `cflock` tags and inconsistent naming of locks. If you nest locks, everyone accessing the locked variables must consistently nest `cflock` tags in the same order. Otherwise, a deadlock can occur.

These examples show situations that cause deadlocks:

**Example deadlock with two users**

<table>
<thead>
<tr>
<th>User 1</th>
<th>User 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Locks the session scope.</td>
<td>Locks the Application scope.</td>
</tr>
<tr>
<td>Deadlock: Tries to lock the Application scope, but it is already locked by User 2.</td>
<td>Deadlock: Tries to lock the Session scope, but it is already locked by User 1.</td>
</tr>
</tbody>
</table>

The following deadlock could occur if you tried to nest an exclusive lock inside a read lock:

**Example deadlock with one user**

<table>
<thead>
<tr>
<th>User 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Locks the Session scope with a read lock.</td>
</tr>
<tr>
<td>Attempts to lock the Session scope with an exclusive lock.</td>
</tr>
<tr>
<td>Deadlock: Cannot lock the Session scope with an exclusive lock because the scope is already locked for reading.</td>
</tr>
</tbody>
</table>

The following code shows this scenario:

```coldfusion
<cflock timeout = "60" scope = "SESSION" type = "readOnly">
    ............
<cflock timeout = "60" scope = "SESSION" type = "Exclusive">
    ............
</cflock>
</cflock>
```
To avoid a deadlock, everyone who nests locks must do so in a well-specified order and name the locks consistently. If you must lock access to the Server, Application, and Session scopes, you must do so in this order:

1. Lock the Session scope. In the `cflock` tag, specify `scope = "session"`. 
2. Lock the Application scope. In the `cflock` tag, specify `scope = "Application"`. 
3. Lock the Server scope. In the `cflock` tag, specify `scope = "server"`. 
4. Unlock the Server scope. 
5. Unlock the Application scope. 
6. Unlock the Session scope. 

**Note:** If you do not have to lock a scope, you can skip any pair of these lock/unlock steps. For example, if you do not have to lock the Server scope, you can skip Steps 3 and 4. Similar rules apply for named locks.

For more information, see the following:

- Chapter 15, “Using Persistent Data and Locking” in *ColdFusion MX Developer's Guide*

**Example**

```cf
<!--- This example shows how cflock can guarantee consistency of data updates to variables in the Application, Server, and Session scopes. --->

<!--- Copy the following code into an Application.cfm file in the application root directory. --->
<br><br>
<!--- Beginning of Application.cfm code ---><br><br>
<cfapplication name = "ETurtle" sessionTimeout = #CreateTimeSpan(0,0, 0, 60)# sessionManagement = "yes">

<!--- Initialize session and application variables used by E-Turtleneck. Use session scope for the session variables. --->
<cflock scope = "Session" timeout = "30" type = "Exclusive">
    <cfif NOT IsDefined("session.size")>
        <cfset session.size = ">
    </cfif>
    <cfif NOT IsDefined("session.color")>
        <cfset session.color = ">
    </cfif>
</cflock>
```

---

To avoid a deadlock, everyone who nests locks must do so in a well-specified order and name the locks consistently. If you must lock access to the Server, Application, and Session scopes, you must do so in this order:

1. Lock the Session scope. In the `cflock` tag, specify `scope = "session"`. 
2. Lock the Application scope. In the `cflock` tag, specify `scope = "Application"`. 
3. Lock the Server scope. In the `cflock` tag, specify `scope = "server"`. 
4. Unlock the Server scope. 
5. Unlock the Application scope. 
6. Unlock the Session scope. 

**Note:** If you do not have to lock a scope, you can skip any pair of these lock/unlock steps. For example, if you do not have to lock the Server scope, you can skip Steps 3 and 4. Similar rules apply for named locks.

For more information, see the following:

- Chapter 15, “Using Persistent Data and Locking” in *ColdFusion MX Developer's Guide*

**Example**

```cf
<!--- This example shows how cflock can guarantee consistency of data updates to variables in the Application, Server, and Session scopes. --->

<!--- Copy the following code into an Application.cfm file in the application root directory. --->
<br><br>
<!--- Beginning of Application.cfm code ---><br><br>
<cfapplication name = "ETurtle" sessionTimeout = #CreateTimeSpan(0,0, 0, 60)# sessionManagement = "yes">

<!--- Initialize session and application variables used by E-Turtleneck. Use session scope for the session variables. --->
<cflock scope = "Session" timeout = "30" type = "Exclusive">
    <cfif NOT IsDefined("session.size")>
        <cfset session.size = "">
    </cfif>
    <cfif NOT IsDefined("session.color")>
        <cfset session.color = "">
    </cfif>
</cflock>
```
<!--- Use an application lock for the application-wide variable that keeps track of the number of turtlenecks sold. For a more efficient, but more complex, way of handling Application scope locking, see "ColdFusion MX Developer's Guide"--->
<cflock scope = "Application" timeout = "30" type = "Exclusive">
<cfif NOT IsDefined("application.number")>
<cfset application.number = 0>
</cfif>
</cflock>
<!----------------------- End of Application.cfm ----------------------->
<h3>cflock Example</h3>
<cfif IsDefined("form.submit")>
<!--- The form has been submitted; process the request. --->
<cfoutput>
Thanks for shopping E-Turtleneck. You chose size <b>#form.size#</b>, color <b>#form.color#</b>.<br><br>
</cfoutput>
<!--- Lock the code that assigns values to session variables. ---->
<cflock scope = "Session" timeout = "30" type = "Exclusive">
<cfparam name = session.size Default = #form.size#>
<cfparam name = session.color Default = #form.color#>
</cflock>
<!---- Lock the code that updates the Application scope number of turtlenecks sold. --->
<cflock scope = "Application" timeout = "30" type = "Exclusive">
<cfset application.number = application.number + 1>
<cfoutput>
E-Turtleneck has now sold #application.number# turtlenecks!
</cfoutput>
</cflock>
</cfif>
<cfelse>
<!--- Show the form only if it has not been submitted. --->
<cflock scope = "Application" timeout = "30" type = "Readonly">
<cfoutput>
E-Turtleneck has sold #application.number# turtlenecks to date.
</cfoutput>
</cflock>
<cfelse>
<!--- Show the form only if it has not been submitted. --->
<cflock scope = "Application" timeout = "30" type = "Readonly">
<cfoutput>
E-Turtleneck has sold #application.number# turtlenecks to date.
</cfoutput>
</cflock>
<form method="post" action="cflocktest.cfm">
<p>Congratulations! You selected the most comfortable turtleneck in the world. Please select color and size.</p>
<table cellspacing = "2" cellpadding = "2" border = "0">
<tr>
<td>Select a color.</td>
<td><select type = "Text" name = "color">
<option>red
<option>white
<option>blue
<table>
<thead>
<tr>
<th>Color</th>
<th>Option</th>
</tr>
</thead>
<tbody>
<tr>
<td>turquoise</td>
<td>&lt;option&gt;</td>
</tr>
<tr>
<td>black</td>
<td>&lt;option&gt;</td>
</tr>
<tr>
<td>forest green</td>
<td>&lt;option&gt;</td>
</tr>
</tbody>
</table>

Select a size.

<table>
<thead>
<tr>
<th>Size</th>
<th>Option</th>
</tr>
</thead>
<tbody>
<tr>
<td>XXsmall</td>
<td>&lt;option&gt;</td>
</tr>
<tr>
<td>Xsmall</td>
<td>&lt;option&gt;</td>
</tr>
<tr>
<td>small</td>
<td>&lt;option&gt;</td>
</tr>
<tr>
<td>medium</td>
<td>&lt;option&gt;</td>
</tr>
<tr>
<td>large</td>
<td>&lt;option&gt;</td>
</tr>
<tr>
<td>Xlarge</td>
<td>&lt;option&gt;</td>
</tr>
</tbody>
</table>

Press Submit when you are finished making your selection.

Submit
**cflog**

**Description**

Writes a message to a log file.

**Category**

Data output tags

**Syntax**

```xml
<cflog
  text = "text"
  log = "log type"
  file = "filename"
  type = "message type"
  application = "yes" or "no">
</cflog>
```

**See also**

cfcol, cfcontent, cfoutput, cftable

**History**

ColdFusion MX: Deprecated the thread, date, and time attributes. They might not work, and might cause an error, in later releases. (In earlier releases, these attributes determined whether the respective data items were output to the log. In ColdFusion MX, this data is always output.)

**Attributes**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Req/Opt</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>text</td>
<td>Required</td>
<td></td>
<td>Message text to log.</td>
</tr>
<tr>
<td>log</td>
<td>Optional</td>
<td></td>
<td>If you omit the file attribute, writes messages to standard log file. Ignored, if you specify file attribute.</td>
</tr>
<tr>
<td>file</td>
<td>Optional</td>
<td></td>
<td>Message file. Specify only the main part of the filename. For example, to log to the Testing.log file, specify “Testing”. The file must be located in the default log directory. You cannot specify a directory path. If the file does not exist, it is created automatically, with the suffix .log.</td>
</tr>
<tr>
<td>type</td>
<td>Optional</td>
<td>Information</td>
<td>Type (severity) of the message:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Information</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Warning</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Error</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Fatal</td>
</tr>
<tr>
<td>application</td>
<td>Optional</td>
<td>yes</td>
<td>• yes: logs the application name, if it is specified in a cfapplication tag or Application.cfc file.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• no</td>
</tr>
</tbody>
</table>
Usage

This tag logs custom messages to standard or custom log files. You can specify a file for the log message or send messages to the default application or scheduler log. The log message can include ColdFusion expressions. Log files must have the suffix .log and must be located in the ColdFusion log directory.

Log entries are written as comma-delimited lists with these fields:

- type
- coio
- date
- time
- application
- text

Values are enclosed in double- quotation marks. If you specify no for the application attribute, the corresponding entry in the list is empty.

You can disable cflog tag execution. For more information, see the ColdFusion Administrator Basic Security page.

The following example logs the name of a user that logs on an application. The message is logged to the file myAppLog.log in the ColdFusion log directory. It includes the date, time, and thread ID, but not the application name.

```
<cflog file="myAppLog" application="no"
   text="User #Form.username# logged on."/>
```

For example, if a user enters "Sang Thornfield" in a form's username field, this entry is added to the myAppLog.log file entry:

"Information","153","02/28/01","14:53:40",,"User Sang Thornfield logged on."


**cflogin**

**Description**

A container for user login and authentication code. ColdFusion runs the code in this tag if a user is not already logged in. You put code in the tag that authenticates the user and identifies the user with a set of roles. Used with `cfloginuser` tag.

**Category**

Security tags

**Syntax**

```xml
<cflogin
    idletimeout = "value"
    applicationToken = "token"
    cookieDomain = "domain"
...
<cfloginuser
    name = "name"
    password = "password-string"
    roles = "roles">
...
</cflogin>
```

**See also**


**History**

ColdFusion MX 6.1: Changed behavior: the `cflogin` variable exists when ColdFusion receives a request with NTLM or Digest (CFHTTP Negotiated header) authentication information.

ColdFusion MX: Added this tag.

**Attributes**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Req/Opt</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>idletimeout</td>
<td>Optional</td>
<td>1800</td>
<td>Time interval, in seconds, after which ColdFusion logs off the user.</td>
</tr>
<tr>
<td>applicationToken</td>
<td>Optional</td>
<td></td>
<td>Unique application identifier. Limits the login validity to one application, as specified by the <code>cfapplication</code> tag or the Application.cfc file.</td>
</tr>
<tr>
<td>cookieDomain</td>
<td>Optional</td>
<td></td>
<td>Domain of the cookie that is used to mark a user as logged in. Use this attribute to enable a user login cookie to work with multiple clustered servers in the same domain.</td>
</tr>
</tbody>
</table>
Usage

The body of this tag executes only if there is no logged-in user. When using application-based security, you put code in the body of the `cflogin` tag to check the user-provided ID and password against a data source, LDAP directory, or other repository of login identification. The body must include a `cfloginuser` tag to establish the authenticated user’s identity in ColdFusion.

You control the data source and are responsible for coding the SQL within the `cflogin` tag, and you must make sure that the associated database has user, password, and role information.

The `cflogin` tag has a built-in `cflogin` structure that contains two variables, `cflogin.name` and `cflogin.password`, if the page is executing in response to any of the following:

- Submission of a form that contains input fields with the names `j_username` and `j_password`.
- A request that uses CFHTTP Basic authentication, and therefore includes an Authorization header with the username and password.
- A request that uses NTLM or Digest authentication. In this case, the username and password are hashed using a one-way algorithm in the Authorization header; ColdFusion gets the username from the web server and sets the `cflogin.password` value to the empty string.

You can use these values in the `cflogin` tag body to authenticate the user, and, in the `cfloginuser` tag, to log the user in. The structure is only available in the `cflogin` tag body.

Example

The following example shows a simple authentication. This code is typically in the `Application.cfc` `onRequestStart` method or in the `application.cfm` page.

```cfml
<cflogin>
  <cfif NOT IsDefined("cflogin")>
    <cfinclude template="loginform.cfm">
    <cfabort>
  <cfelse>
    <cfif cflogin.name eq "admin">
      <cfset roles = "user,admin">
    <cfelse>
      <cfset roles = "user">
    </cfif>
  </cfif>
  <cfloginuser name = "#cflogin.name#" password = "#cflogin.password#" roles = "#roles#" />
</cfif>
</cflogin>

The following view-only example checks the user ID and password against a data source:

```
<cfif qSecurity.recordcount gt 0>
<cfloginuser name="#cflogin.name#" password="#cflogin.password#" roles="#trim(qSecurity.Roles)#" />
</cfif>
cfloginuser

Description

Identifies an authenticated user to ColdFusion. Specifies the user ID and roles. Used within a cflogin tag.

Category

Security tags

Syntax

```xml
<cfloginuser
    name = "name"
    password = "password-string"
    roles = "roles">
</cfloginuser>
```

See also


History

ColdFusion MX 6.1: Changed behavior: if the Session scope is enabled, and the cfapplication tag loginStorage attribute is set to Session, the login remains in effect until the session expires or the user is logged out by the cflogout tag.

ColdFusion MX: Added this tag.

Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Req/Opt</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Required</td>
<td>A username.</td>
<td></td>
</tr>
<tr>
<td>password</td>
<td>Required</td>
<td>A user password.</td>
<td></td>
</tr>
<tr>
<td>roles</td>
<td>Required</td>
<td>A comma-delimited list of role identifiers. ColdFusion processes spaces in a list element as part of the element.</td>
<td></td>
</tr>
</tbody>
</table>

Usage

Used inside the cflogin tag to identify the authenticated user to ColdFusion. After you call this function, the GetAuthUser and IsUserInRoles return the user name and role information.

Note: By default, the user information is stored as memory-only cookies. The cfapplication tag or the Application.cfc This.loginStorage variable can specify that login information be stored in the Session scope.

Example

See cflogin on page 278.
**cflogout**

**Description**

Logs the current user out. Removes knowledge of the user ID, password, and roles from the server. If you do not use this tag, the user is automatically logged out when the session ends.

**Category**

*Security tags*

**Syntax**

```cflogout```

**See also**

*cflogin, cfloginuser*. Chapter 16, “Securing Applications” in *ColdFusion MX Developer’s Guide*

**History**

ColdFusion MX 6.1: Changed behavior: if the Session scope is enabled, a login remains in effect until the session expires or the user is logged out by the `cflogout` tag.

ColdFusion MX: Added this tag.

**Example**

```cflogin```

```cfloginuser```

```name = "foo"
password = "bar"
roles = "admin"```n
```
```n
```</cfloginuser```

```<cflogout>
<cfoutput>Authorized user: #getAuthUser()#</cfoutput>
<cfoutput>Authorized user: #getAuthUser()#</cfoutput>```n
```
```n
```
**cfloop**

**Description**

Looping is a programming technique that repeats a set of instructions or displays output repeatedly until one or more conditions are met. This tag supports the following types of loops:

- “cfloop: index loop” on page 284
- “cfloop: conditional loop” on page 286
- “cfloop: looping over a date or time range” on page 287
- “cfloop: looping over a query” on page 288
- “cfloop: looping over a list or file” on page 290
- “cfloop: looping over a COM collection or structure” on page 292

For more information, see “cfloop and cfbreak” in Chapter 2, “Elements of CFML,” and “Populating arrays with data” in Chapter 5, “Using Arrays and Structures,” in *ColdFusion MX Developer’s Guide*.

**Category**

Flow-control tags
cfloop: index loop

Description

An index loop repeats for a number of times that is determined by a numeric value. An index loop is also known as a FOR loop.

Syntax

```<cfloop
index = "parameter_name"
from = "beginning_value"
to = "ending_value"
step = "increment">
... HTML or CFML code ...
</cfloop>```

See also

cfabsort, cfbreak, cfdirectory, cfexecute, cfexit, cfif, cflocation, cfrethrow, cfswitch, cftthrow, cftry; "cfloop and cfbreak" in Chapter 2, “Elements of CFML,” in ColdFusion MX Developer’s Guide

Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Req/Opt</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>index</td>
<td>Required</td>
<td></td>
<td>Index value. ColdFusion sets it to the from value and increments or decrements by step value, until it equals the to value.</td>
</tr>
<tr>
<td>from</td>
<td>Required</td>
<td></td>
<td>Beginning value of index.</td>
</tr>
<tr>
<td>to</td>
<td>Required</td>
<td></td>
<td>Ending value of index.</td>
</tr>
<tr>
<td>step</td>
<td>Optional</td>
<td>1</td>
<td>Step by which to increment or decrement the index value.</td>
</tr>
</tbody>
</table>

Usage

Using anything other than integer values in the from and to attributes of an index loop can produce unexpected results. For example, if you increment through an index loop from 1 to 2, with a step of 0.1, ColdFusion outputs "1,1.1,1.2,...,1.9", but not "2". This is a programming language problem regarding the internal representation of floating point numbers.

Note: The to value is evaluated once, when the cfloop tag is encountered. Any change to this value within the loop block, or within the expression that evaluates to this value, does not affect the number of times the loop is executed.

Example

In this example, the code loops five times, displaying the index value each time:

```<cfloop index = "LoopCount" from = "1" to = "5">
    The loop index is <cfoutput>#LoopCount#</cfoutput>.<br>
</cfloop>```

The output of this loop is as follows:

The loop index is 1.
The loop index is 2.
The loop index is 3.
The loop index is 4.
The loop index is 5.

In this example, the code loops four times, displaying the index value each time. The value of \( j \) is decreased by one for each iteration. This does not affect the value of to, because it is a copy of \( j \) that is made before entering the loop.

```cftext
<cfset j = 4>
<cfloop index = "LoopCount" from = "1" to = #j#>
  <cfoutput>The loop index is #LoopCount#</cfoutput>.<br>
  <cfset j = j - 1>
</cfloop>
```

The output of this loop is as follows:
The loop index is 1.
The loop index is 2.
The loop index is 3.
The loop index is 4.

As before, the value of \( j \) is decremented by one for each iteration, but this does not affect the value of to, because its value is a copy of \( j \) that is made before the loop is entered.

In this example, step has the default value, 1. The code decrements the index:

```cftext
<cfloop index = "LoopCount"
  from = "5"
  to = "1"
  step = "-1">
  The loop index is #LoopCount#.<br>
</cfloop>
```

The output of this loop is as follows:
The loop index is 5.
The loop index is 4.
The loop index is 3.
The loop index is 2.
The loop index is 1.
cfloop: conditional loop

Description
A conditional loop iterates over a set of instructions as long as a condition is True. To use this type of loop correctly, the instructions must change the condition every time the loop iterates, until the condition is False. Conditional loops are known as WHILE loops, as in, "loop WHILE this condition is true."

Syntax
<cfloop
    condition = "expression">
    ...
</cfloop>

See also
 cfabort, cfbreak, cfexecute, cfexit, cfif, cflocation, cfswitch, cfthrow, cftry; “cfloop and cfbreak” in Chapter 2, “Elements of CFML,” in ColdFusion MX Developer’s Guide

Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Req/Opt</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>condition</td>
<td>Required</td>
<td></td>
<td>Condition that controls the loop.</td>
</tr>
</tbody>
</table>

Example
The following example increments CountVar from 1 to 5.
<!--- Set the variable CountVar to 0. --->
<cfset CountVar = 0>
<!--- Loop until CountVar = 5. --->
<cfloop condition = "CountVar LESS THAN OR EQUAL TO 5">
    <cfset CountVar = CountVar + 1>
    The loop index is <cfoutput>#CountVar#</cfoutput>.<br>
</cfloop>

The output of this loop is as follows:
The loop index is 1.
The loop index is 2.
The loop index is 3.
The loop index is 4.
The loop index is 5.
cfloop: looping over a date or time range

Description

Loops over the date or time range specified by the from and to attributes. By default, the step is 1 day, but you can change the step by creating a timespan. The cfloop tag loops over tags that cannot be used within a cfoutput tag.

Syntax

```cfml
<cfloop
    from = "start_time"
    to = "end_time"
    index = "current_value"
    step = "increment">
</cfloop>
```

See also
cfabort, cfbreak, cfdirectory, cfexecute, cfexit, cfif, cflocation, cfrethrow, cfswitch, cfthrow, cftry; “cfloop and cfbreak” in Chapter 2, “Elements of CFML,” in ColdFusion MX Developer’s Guide

Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Req/Opt</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>from</td>
<td>Required</td>
<td></td>
<td>The beginning of the date or time range.</td>
</tr>
<tr>
<td>to</td>
<td>Required</td>
<td></td>
<td>The end of the date or time range.</td>
</tr>
<tr>
<td>index</td>
<td>Required</td>
<td>1 day</td>
<td>Index value. ColdFusion sets it to the from value and increments by the step value, until it equals the to value.</td>
</tr>
<tr>
<td>step</td>
<td>Optional</td>
<td></td>
<td>Step, expressed as a timespan, by which the index increments.</td>
</tr>
</tbody>
</table>

Example

The following example loops from today’s date to today’s date plus 30 days, stepping by 7 days at a time and displaying the date:

```cfml
<cfset startDate = Now()>
<cfset endDate = Now() + 30>
<cfloop from="#startDate#" to="#endDate#" index="i"
    step="#CreateTimeSpan(7,0,0,0)#">
    <cfoutput>#dateformat(i, "mm/dd/yyyy")<br /></cfoutput>
</cfloop>
```

The following example displays the time in 30-minute increments, starting from midnight and ending 23 hours, 59 minutes, and 59 seconds later:

```cfml
<cfset startTime = CreateTime(0,0,0)>
<cfset endTime = CreateTime(23,59,59)>
<cfloop from="#startTime#" to="#endTime#" index="i"
    step="#CreateTimeSpan(0,0,30,0)#">
    <cfoutput>#TimeFormat(i, "hh:mm tt")<br /></cfoutput>
</cfloop>
```
**cfloop: looping over a query**

**Description**
A loop over a query executes for each record in a query record set. The results are similar to those of the `cfoutput` tag. During each iteration, the columns of the current row are available for output. The `cfloop` tag loops over tags that cannot be used within a `cfoutput` tag.

**Syntax**
```cfml
<cfloop
    query = "query_name"
    startRow = "row_num"
    endRow = "row_num"
</cfloop>
```

**See also**
`cfabort`, `cfbreak`, `cfexecute`, `cfexit`, `cfif`, `cflocation`, `cfoutput`, `cfswitch`, `cfthrow`, `cftry`; For more information, see “cfloop and cfbreak” in Chapter 2, “Elements of CFML,” in *ColdFusion MX Developer’s Guide*

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Req/Opt</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>query</td>
<td>Required</td>
<td></td>
<td>Query that controls the loop.</td>
</tr>
<tr>
<td>startRow</td>
<td>Optional</td>
<td></td>
<td>First row of query that is included in the loop.</td>
</tr>
<tr>
<td>endRow</td>
<td>Optional</td>
<td></td>
<td>Last row of query that is included in the loop.</td>
</tr>
</tbody>
</table>

**Example**
```cfml
<cfquery name = "MessageRecords"
    dataSource = "cfdocexamples">
    SELECT * FROM Messages
</cfquery>
<cfloop query = "MessageRecords">
    <cfoutput>#Message_ID#</cfoutput><br>
</cfloop>
```

The `cfloop` tag also iterates over a record set with dynamic start and stop points. This gets the next $n$ sets of records from a query. This example loops from the fifth through the tenth record returned by the `MessageRecords` query:
```cfml
<cfset Start = 5>
<cfset End = 10>
<cfloop query = "MessageRecords"
    startRow = "#Start#"
    endRow = "#End#">
    <cfoutput>#MessageRecords.Message_ID#</cfoutput><br>
</cfloop>
```

The loop stops when there are no more records, or when the current record index is greater than the value of the `endRow` attribute.
The following example combines the pages that are returned by a query of a list of page names into one document, using the `cfinclude` tag:

```cfml
<cfquery name = "GetTemplate" dataSource = "Library" maxRows = "5">
SELECT TemplateName
FROM Templates
</cfquery>
<cfloop query = "GetTemplate">
<cfinclude template = "#TemplateName#">
</cfloop>
```
cfloop: looping over a list or file

Description
Looping over a list steps through elements contained in any of these entities:
• A variable
• A value that is returned from an expression
• A file

Syntax
<cfloop
   index = "index_name"
   list = "list_items"
   delimiters = "item_delimiter">
   ...
</cfloop>

See also
cfabort, cfbreak, cfexecute, cfexit, cfif, cflocation, cfswitch, cfthrow, cftry; “cfloop and cfbreak” in Chapter 2, “Elements of CFML,” in ColdFusion MX Developer’s Guide

Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Req/Opt</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>index</td>
<td>Required</td>
<td></td>
<td>In a list loop, the variable to receive the next list element.</td>
</tr>
<tr>
<td>list</td>
<td>Required</td>
<td></td>
<td>A list, variable, or filename; contains a list.</td>
</tr>
<tr>
<td>delimiters</td>
<td>Optional</td>
<td></td>
<td>Character(s) that separates items in list.</td>
</tr>
</tbody>
</table>

Example
This loop displays four names:
<cfloop index = "ListElement"
   list = "John,Paul,George,Ringo"
   delimiters = ".:/">
   <cfoutput>#ListElement#</cfoutput><br>
</cfloop>

You can put more than one character in the delimiters attribute, in any order. For example, this loop processes commas, colons, and slashes as list delimiters:
<cfloop index = "ListElement"
   list = "John/Paul,George::Ringo"
   delimiters = ",:/">
   <cfoutput>#ListElement#</cfoutput><br>
</cfloop>

ColdFusion skips the second and subsequent consecutive delimiters between list elements. Thus, in the example, the two colons between "George" and "Ringo" are processed as one delimiter.
To loop over each line of a file, use the tag this way:

```cfml
<cfloop list="#theFile#"
    index="curLine"
    delimiters="#chr(10)##chr(13)#">
    ...
</cfloop>
```
cfloop: looping over a COM collection or structure

Description

The cfloop collection attribute loops over every object within a COM/DCOM collection object, or every element in a structure:

• A COM/DCOM collection object is a set of similar items referenced as a group. For example, the group of open documents in an application is a collection.

• A structure contains a related set of items, or it can be used as an associative array. Looping is particularly useful when using a structure as an associative array.

In the loop, each item is referenced by the variable name in the item attribute. The loop executes until all items have been accessed.

The collection attribute is used with the item attribute. In the example that follows, item is assigned a variable called file2, so that with each cycle in the cfloop, each item in the collection is referenced. In the cfoutput section, the name property of the file2 item is referenced for display.

For more information, see Chapter 38, “Integrating COM and CORBA Objects in CFML Applications,” in ColdFusion MX Developer’s Guide.

Example

This example uses a COM object to output a list of files. In this example, FFunc is a collection of file2 objects.

```coldfusion
<cfobject class = FileFunctions.files name = FFunc action = Create>
<cfset FFunc.Path = "c:\">
<cfset FFunc.Mask = ".*" >
<cfset FFunc.attributes = 16 >
<cfset x = FFunc.GetFileList()>
<cfloop collection = #FFUNC# item = "file2">
  <cfoutput><br>#file2.name#</cfoutput>
</cfloop>
<!---Loop through a structure that is used as an associative array: --->
...<!--- Create a structure and loop through its contents. --->
<cfset Departments = StructNew()>
<cfset val = StructInsert(Departments, "John ", "Sales ")>
<cfset val = StructInsert(Departments, "Tom ", "Finance ")>
<cfset val = StructInsert(Departments, "Mike ", "Education ")>
<!--- Build a table to display the contents --->
<cfoutput>
<table cellpadding = "2 " cellspacing = "2 ">
  <tr>
    <td><b>Employee</b></td>
    <td><b>Dept.</b></td>
  </tr>
  <!--- Use item to create the variable person to hold value of key as loop runs. --->
  <cfloop collection = #Departments# item = "person ">
    <tr>
      <td><b><cfoutput>
          Employee</cfoutput></b></td>
      <td><cfoutput>Dept.</cfoutput></td>
    </tr>
  </cfloop>
</cfoutput>
```

292 Chapter 2: ColdFusion Tags
<tr>
  <td>#person#</td>
  <td>#StructFind(Departments, person)#</td>
</tr>
cfmail

Description
Sends an e-mail message that optionally contains query output, using an SMTP server.

Category
Internet Protocol tags

Syntax
```<cfmail
  to = "recipient"
  from = "sender"
  cc = "copy_to"
  bcc = "blind_copy_to"
  subject = "msg_subject"
  replyto = "reply_to_addr"
  failto = "fail_message_addr"
  username = "user name"
  password = "password"
  wraptext = "column number"
  charset = "character encoding"
  type = "msg_type"
  mimeattach = "path"
  query = "query_name"
  group = "query_column"
  groupcasesensitive = "yes" or "no"
  startrow = "query_row"
  maxrows = "max_msgs"
  server = "serverspecs"
  port = "port_id"
  mailerid = "headerid"
  timeout = "seconds"
  spoolenable = "yes" or "no"
  debug = "yes" or "no">
  (Optional) Mail message body and/or cfhttpparam tags
</cfmail>
```

See also
cfmailparam, cfmailpart, cfpop, cfhttp, cfldap, Wrap; "Using ColdFusion with mail servers" in Chapter 39, “Sending and Receiving E-Mail,” in ColdFusion MX Developer’s Guide

ColdFusion MX 6.1:
• Added the following attributes: charset, failto, replyto, username, password and wraptext.
• Added support for multiple mail servers in the server attribute.
• Added several configuration options to the ColdFusion MX Administrator Mail Settings page.

ColdFusion MX: Added the SpoolEnable attribute.
### Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Req/Opt</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
</table>
| **to**    | Required|         | Message recipient e-mail addresses:  
|           |         |         | • Static address; for example, "support@macromedia.com".  
|           |         |         | • Variable that contains an address; for example, "#0Form.Email#".  
|           |         |         | • Name of a query column that contains an address; for example, "#EMail#". An e-mail message is sent for each returned row. |
| **from**  | Required|         | E-mail message sender:  
|           |         |         | • A static string; for example, "support@mex.com"  
<p>|           |         |         | • A variable; for example, &quot;#GetUser.EMailAddress#&quot;. This attribute does not have to be a valid Internet address; it can be any text string. |
| <strong>cc</strong>    | Optional|         | Address(es) to which to copy the message. |
| <strong>bcc</strong>   | Optional|         | Address(es) to which to copy the message, without listing them in the message header. |
| <strong>subject</strong> | Required|         | Message subject. Can be dynamically generated. For example, to send messages that give customers status updates: &quot;Status of Order Number #Order_ID#&quot;. |
| <strong>replyto</strong> | Optional|         | Address(es) to which the recipient is directed to send replies. |
| <strong>failto</strong> | Optional|         | Address to which mailing systems should send delivery failure notifications. Sets the mail envelope reverse-path value. |
| <strong>username</strong> | Optional|         | A user name to send to SMTP servers that require authentication. Requires a <strong>password</strong> attribute. |
| <strong>password</strong> | Optional|         | A password to send to SMTP servers that require authentication. Requires a <strong>username</strong> attribute. |
| <strong>wraptext</strong> | Optional| Do not wrap text | Specifies the maximum line length, in characters of the mail text. If a line has more than the specified number of characters, replaces the last white space character, such as a tab or space, preceding the specified position with a line break. If there are no white space characters, inserts a line break at the specified position. A common value for this attribute is 72. |</p>
<table>
<thead>
<tr>
<th>Attribute</th>
<th>Req/Opt</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>charset</td>
<td>Optional</td>
<td>Character encoding</td>
<td>Character encoding of the mail message, including the headers. The following list includes commonly used values:</td>
</tr>
</tbody>
</table>
|             |           | selected in ColdFusion   | • utf-8  
|             |           | MX Administrator Mail page; default is UTF-8 | • iso-8859-1  
|             |           |                           | • windows-1252  
|             |           |                           | • us-ascii  
|             |           |                           | • shift_jis  
|             |           |                           | • iso-2022-jp  
|             |           |                           | • euc-jp  
|             |           |                           | • euc-kr  
|             |           |                           | • iso-2022-kr  
|             |           |                           | • big5  
|             |           |                           | • hz-gb-2312  
|             |           |                           | • euc-cn  
|             |           |                           | • utf-16  
|             |           |                           | For more information on character encodings, see [www.w3.org/International/O-charset.html](http://www.w3.org/International/O-charset.html). |
| type        | Optional  | text/plain MIME type     | MIME type of the message. Can be a valid MIME media type or one of the following:                                                           |
|             |           | of the message.          | • text: specifies text/plain type.  
|             |           |                           | • plain: specifies text/plain type.  
|             |           |                           | • html: specifies text/html type.  
|             |           |                           | For a list of all registered MIME media types, see [www.iana.org/assignments/media-types/](http://www.iana.org/assignments/media-types/). |
| MIMEAttach  | Optional  | Path of file to attach   | Path of file to attach to message. Attached file is MIME-encoded. ColdFusion attempts to determine the MIME type of the file; use the cfmailparam tag to send an attachment and specify the MIME type. |
|             |           | to message.              |                                                                                                                                           |
| query       | Optional  | Name of cfquery from     | Name of cfquery from which to draw data for message(s). Use this attribute to send more than one message, or to send query results within a message. |
|             |           | which to draw data for    |                                                                                                                                           |
|             |           | message(s).              |                                                                                                                                           |
| group       | Optional  | CurrentRow Query column  | Query column to use when you group sets of records to send as a message. For example, to send a set of billing statements to a customer, group on "customer_id." Case-sensitive. Eliminates adjacent duplicates when data is sorted by the specified field. |
|             |           | to use when you group    |                                                                                                                                           |
|             |           | sets of records to send  |                                                                                                                                           |
|             |           | as a message.            |                                                                                                                                           |
|             |           | to send as a message.    |                                                                                                                                           |
| groupCase   | Optional  | No                       | Boolean. Whether to consider case when using the group attribute. To group on case-sensitive records, set this attribute to Yes.              |
| Sensitive   |           |                           |                                                                                                                                           |
| startRow    | Optional  | 1                        | Row in a query to start from.                                                                                                               |
| maxRows     | Optional  |                           | Maximum number of messages to send when looping over a query.                                                                           |
Usage

Sends a mail message to the specified address. Mail messages can include attachments. The tag body can include CFML code to generate mail output. The cfmailparam and cfmailpart tags can only be used in the cfmail tag body.

Mail messages can be single or multipart. If you send a multi-part mail message, all message content must be in cfmailpart tags; ColdFusion ignores multipart message text that is not in cfmailpart tags.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Req/Opt</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>server</td>
<td>Optional</td>
<td>SMTP server address, or (Enterprise edition only) a comma-delimited list of server addresses, to use for sending messages. At least one server must be specified here or in the ColdFusion MX Administrator. A value here overrides the Administrator. A value that includes a port specification overrides the port attribute. See the Usage section for details.</td>
<td></td>
</tr>
<tr>
<td>port</td>
<td>Optional</td>
<td>TCP/IP port on which SMTP server listens for requests (normally 25). A value here overrides the Administrator.</td>
<td></td>
</tr>
<tr>
<td>mailerID</td>
<td>Optional</td>
<td>ColdFusion MX Application Server Mailer ID to be passed in X-Mailer SMTP header, which identifies the mailer application.</td>
<td></td>
</tr>
<tr>
<td>timeout</td>
<td>Optional</td>
<td>Number of seconds to wait before timing out connection to SMTP server. A value here overrides the Administrator.</td>
<td></td>
</tr>
<tr>
<td>spoolEnable</td>
<td>Optional</td>
<td>Specifies whether to spool mail or always send it immediately. Overrides the ColdFusion MX Administrator Spool mail messages to disk for delivery setting.</td>
<td></td>
</tr>
<tr>
<td>debug</td>
<td>Optional</td>
<td>No • Yes: sends debugging output to standard output. By default, if the console window is unavailable, ColdFusion sends output to cf_root\runtime\logs\coldfusion-out.log on server configurations. On J2EE configurations, with JRun, the default location is jrun_home/logs/servername-out.log. Caution: If you set this option to Yes, ColdFusion MX writes detailed debugging information to the log, including all message contents, and can generate large logs quickly. • No: does not generate debugging output.</td>
<td></td>
</tr>
</tbody>
</table>

Usage

Sends a mail message to the specified address. Mail messages can include attachments. The tag body can include CFML code to generate mail output. The cfmailparam and cfmailpart tags can only be used in the cfmail tag body.

Mail messages can be single or multipart. If you send a multi-part mail message, all message content must be in cfmailpart tags; ColdFusion ignores multipart message text that is not in cfmailpart tags.
Note: The cfmail tag does not make copies of attachments when spooling mail to disk. If you use the cfmail tag to send a message with an attachment with spooling enabled and you use the cffile tag to delete the attachment file, ColdFusion might not send the mail because the mailing process might execute after the file was is deleted. (When this happens, the mail log includes a FileNotFound exception and the e-mail is not sent.) You can prevent this problem by setting SpoolEnable="No" in the attribute or disabling spooling in the ColdFusion MX Administrator. Disabling spooling causes the e-mail to be delivered immediately.

### Mail addressing

Mail addresses can have any of the following forms:

<table>
<thead>
<tr>
<th>Format</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>user@server</td>
<td><a href="mailto:rsmith@company.com">rsmith@company.com</a></td>
</tr>
<tr>
<td><a href="mailto:user@server">user@server</a></td>
<td><a href="mailto:rsmith@company.com">rsmith@company.com</a></td>
</tr>
<tr>
<td>DisplayName <a href="mailto:user@server">user@server</a></td>
<td>Rob Smith <a href="mailto:rsmith@company.com">rsmith@company.com</a></td>
</tr>
<tr>
<td>&quot;DisplayName&quot; <a href="mailto:user@server">user@server</a></td>
<td>&quot;Rob Smith&quot; <a href="mailto:rsmith@company.com">rsmith@company.com</a></td>
</tr>
<tr>
<td>user@server (DisplayName)</td>
<td><a href="mailto:rsmith@company.com">rsmith@company.com</a> (Rob Smith)</td>
</tr>
</tbody>
</table>

### Specifying mail servers

The server attribute can specify one or more mail servers.

Note: If you specify multiple mail servers in ColdFusion MX Standard, the cfmail tag uses only the first server in the specification. ColdFusion logs a warning message to the mail log file and ignores the additional servers.

For each server, you can optionally specify a username, password, and port. These values override the corresponding attributes, if any. The server attribute has the following format:

[user:password@]server[:port],[user:password@]server[:port],....

For example, the following line specifies one server, mail.myco.com that uses the default port and no user or password, and a second server with a user, password, and specific port:

```
server@mail.myco.com,mail_admin:adm2qzf@mail2.myco.com:24
```

When you specify multiple mail servers in ColdFusion Enterprise, ColdFusion tries the available servers in the order they are listed until it connects to a server. ColdFusion does not try to connect to a server that was unavailable in the last 60 seconds.

### Example

```html
<!—- Delete the surrounding comments to use this example. ——>

<cfif IsDefined("form.mailto")>
  <cfif form.mailto is not ""
    AND form.mailfrom is not ""
    AND form.Subject is not "">
    <cfmail to = ";#form.mailto#" 
      from = ";#form.mailFrom#" 
      subject = ";#form.subject#">
  </cfif>
</cfif>
```
This message was sent by an automatic mailer built with cfmail:

#form.body#
</cfmail>

<h3>Thank you</h3>
<p>Thank you, <cfoutput>#mailfrom#: your message, #subject#, has been sent to #mailto#</cfoutput>.</p>
</cfif>
</cfif>

<form action = "cfmail.cfm">
<pre>
TO:  <input type = "Text" name = "MailTo">
FROM: <input type = "Text" name = "MailFrom">
SUBJECT: <input type = "Text" name = "Subject">
<hr>
MESSAGE BODY:
<textarea name ="body" cols="40" rows="5" wrap="virtual"></textarea></pre>
<!--- Establish required fields. --->
<input type = "hidden" name = "MailTo_required" value = "You must enter a recipient">
<input type = "hidden" name = "MailFrom_required" value = "You must enter a sender">
<input type = "hidden" name = "Subject_required" value = "You must enter a subject">
<input type = "hidden" name = "Body_required" value = "You must enter some text">
<p><input type = "Submit" name = ""></p>
</form>
--->

</pre>
cfmailparam

**Description**

Attaches a file or adds a header to an e-mail message.

**Category**

Internet Protocol tags

**Syntax**

```
<cfmail
to = "recipient"
subject = "msg_subject"
from = "sender"
...more attributes... 

<cfmailparam
  file = "file-name"
  type = "media type"
  contentID = "content ID"
  disposition = "disposition type"
>

or

<cfmailparam
  name = "header-name"
  value = "header-value"
>
...
</cfmail>
```

**See also**

cfmail, cfmailpart, cfttp, cfhtp, cfldap, cfpop; "Using the cfmailparam tag" in Chapter 39, "Sending and Receiving E-Mail," in ColdFusion MX Developer's Guide

**History**

ColdFusion MX 6.x: Added the Disposition and ContentID attributes.

ColdFusion MX 6.1: Added the type attribute.

**Attributes**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Req/Opt</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>file</td>
<td>Required if you do not specify name attribute</td>
<td></td>
<td>Attaches a file in a message. Mutually exclusive with name attribute. The file is MIME encoded before sending.</td>
</tr>
<tr>
<td>name</td>
<td>Required if you do not specify file attribute</td>
<td></td>
<td>Name of header. Case-insensitive. Mutually exclusive with file attribute.</td>
</tr>
</tbody>
</table>
cfmailparam 301

Usage

This tag attaches a file or adds a header to an e-mail message. It can only be used in the cfmail tag. You can use multiple cfmailparam tags within a cfmail tag.

You can use this tag to include a file, such as an image, in an HTML mail message. The file can be displayed inline in an HTML message, or as an attachment. To include multiple files, use multiple cfmailparam tags.

To display a file inline in a mail message:
1. Specify type="html" in the cfmail tag.
2. Specify disposition="inline" and a ContentID attribute in the cfmailparam tag.
3. Use a src="cid:ContentIDValue" attribute to identify the content to include in the HTML tag such as the img tag.

The second example shows this use.

Examples

<h3>cfmailparam Examples</h3>
<p>This view-only example uses cfmailparam to 1) add header to a message and 2) attach files and 3) to return a receipt to the sender.</p>
<cfmail from = "peter@domain.com" To = "paul@domain.com"
Subject = "See Important Attachments and Reply">
<cfmailparam name = "Importance" value = "High">
Please review the new logo. Tell us what you think.
<cfmailparam name = "Disposition-Notification-To" value = "peter@domain.com">
<cfmailparam file = "c:\work\logo.gif" type = "image/gif">
</cfmailparam>
<cfmailparam file = "c:\work\readme.txt" type = "text/plain">
</cfmailparam>
</cfmail>
This view-only example displays an image in the body of an HTML message.

```cfmail
type="HTML"
to="#form.mailto#"
from="#form.mailFrom#"
subject="Sample inline image" 
<cfmailparam file="C:\Inetpub\wwwroot\web.gif"
    disposition="inline"
    contentID="image1">
    <p>There should be an image here</p>
    <img src="cid:image1">
    <p>After the picture</p>
</cfmail>
```
cfmailpart

Description

Specifies one part of a multipart e-mail message. Can only be used in the cfmail tag. You can use more than one cfmailpart tag within a cfmail tag.

Category

Internet Protocol tags

Syntax

```
<cfmail
 ...>
(Optional cfmailparam entries)
<cfmailpart
   type="mime type"
   charset="character encoding"
   wraptext="number"
>
 Mail part contents
 </cfmailpart>
 ...
 </cfmail>
```

History

ColdFusion MX 6.1: Added this tag.

See also

cfmail, cfmailparam, cfpop, cfftp, cfhttp, cfldap, cfcontent, Wrap; “E-mail” in Chapter 17, “Developing Globalized Applications,” in ColdFusion MX Developer’s Guide

Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Req/Opt</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
</table>
| type      | Required|         | The MIME media type of the part. Can be a can be valid MIME media type or one of the following:  
• text: specifies text/plain type.  
• plain: specifies text/plain type.  
• html: specifies text/html type.  
Note: For a list of all registered MIME media types, see www.iana.org/assignments/media-types/.
Usage

Use this tag to create mail messages with alternative versions of the message that duplicate the content in multiple formats. The most common use is to send a plain text version of the message that can be read by all mail readers followed by a version formatted in HTML for display by HTML-compatible mail readers. Specify the simplest version first, with more complex versions afterwards. For more information, see www.ietf.org/rfc/rfc2046.txt.

Example

```html
<h3>cfmailpart Example</h3>
<cfmail from = "peter@domain.com" To = "paul@domain.com"
    Subject = "Which version do you see?">
    <cfmailpart type="text"
        wraptext="74">
        You are reading this message as plain text, because your mail reader does not handle HTML text.
    </cfmailpart>
    <cfmailpart type="html">
        <h3>HTML Mail Message</h3>
        <p>You are reading this message as <strong>HTML</strong>.</p>
        <p>Your mail reader handles HTML text.</p>
    </cfmailpart>
</cfmail>
```

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Req/Opt</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>wraptext</td>
<td>Optional</td>
<td>Do not wrap text</td>
<td>Specifies the maximum line length, in characters of the mail text. If a line has more than the specified number of characters, replaces the last white space character, such as a tab or space, preceding the specified position with a line break. If there are no white space characters, inserts a line break at the specified position. A common value for this attribute is 72.</td>
</tr>
<tr>
<td>charset</td>
<td>Optional</td>
<td>Character encoding specified by charset attribute of cfmail tag</td>
<td>The character encoding in which the part text is encoded. The following list includes commonly used values: • utf-8 • iso-8859-1 • windows-1252 • us-ascii • shift_jis • iso-2022-jp • euc-jp • euc-kr • iso-2022-kr • big5 • hz-gb-2312 • euc-cn • utf-16 For more information on character encodings, see <a href="http://www.w3.org/International/O-charset.html">www.w3.org/International/O-charset.html</a>.</td>
</tr>
</tbody>
</table>
cfmodule

Description
Invokes a custom tag for use in ColdFusion application pages. This tag processes custom tag name conflicts.

Category
Application framework tags

Syntax
<cfmodule
template = "path"
name = "tag_name"
attributeCollection = "collection_structure"
attribute_name1 = "valuea"
attribute_name2 = "valueb"
...
>

See also
cfapplication, cfassociate, cflock; Chapter 11, “Creating and Using Custom CFML Tags” in ColdFusion MX Developer’s Guide

History
ColdFusion MX: Changed behavior when using this tag within a custom tag: if the attribute_name parameter is the same as a key element within the attributeCollection parameter, ColdFusion now uses the name value that is within the attributeCollection parameter. (Earlier releases did not process this consistently.)

Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Req/Opt</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
</table>
| template      | Required| unless name attribute is used | Mutually exclusive with the name attribute. A path to the page that implements the tag.  
- Relative path: expanded from the current page.  
- Absolute path: expanded using ColdFusion mapping.  
A physical path is not valid. |
| name          | Required| unless template attribute is used | Mutually exclusive with the template attribute. A custom tag name, in the form “Name.Name.Name...” identifies subdirectory, under the ColdFusion tag root directory, that contains custom tag page. For example (Windows format):  
<cfmodule name = "macromedia.Forums40.GetUserOptions"/>
This identifies the page GetUserOptions.cfm in the directory CustomTags\macromedia\Forums40 under the ColdFusion root directory. |
Usage

To name a ColdFusion page that contains the custom tag definition, including its path, use the `template` attribute. To refer to the custom tag in the ColdFusion installation directory, using dot notation to indicate its location, use the `name` attribute.

On UNIX systems, ColdFusion searches first for a file with a name that matches the `name` attribute, but is all lower case. If it does not find the file, it looks for a file name that matches the attribute with identical character casing.

You can use `attributeCollection` and `attribute_name` in the same call.

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

Similarly, if the custom tag uses a `cfassociate` tag to save its attributes, the attributes passed with `attributeCollection` are saved as independent attribute values, with no indication that they are grouped into a structure by the custom tag’s caller.

If you specify an end tag to `cfmodule`, ColdFusion calls your custom tag as if it had both a start and an end tag. For more information, see “Handling end tags” of Chapter 11, “Creating and Using Custom CFML Tags” in ColdFusion MX Developer's Guide.

Example

```html
<h3>cfmodule Example</h3>
<p>This view-only example shows use of cfmodule to call a custom tag inline.</p>
<p>This example uses a sample custom tag that is saved in myTag.cfm in the snippets directory. You can also save ColdFusion custom tags in the CFusionMX7\CustomTags directory.
<cfset attrCollection1 = StructNew()>
<cfparam name="attrCollection1.value1" default="22">
<cfparam name="attrCollection1.value2" default="45">
<cfparam name="attrCollection1.value3" default="88">
<!--- Call the tag with CFMODULE with Name--->
<cfmodule
    Template="myTag.cfm"
    X="3"
    attributeCollection=#attrCollection1#
    Y="4">
<!--- Show the code. --->
<HR size="2" color="#0000A0">
<P>Here is one way in which to invoke the custom tag.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Req/Opt</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>attributeCollection</td>
<td>Optional</td>
<td></td>
<td>Structure. A collection of key-value pairs that represent attribute names and values. You can specify multiple key-value pairs. You can specify this attribute only once.</td>
</tr>
<tr>
<td>attribute_name</td>
<td>Optional</td>
<td></td>
<td>Attribute for a custom tag. You can include multiple instances of this attribute to specify the parameters of a custom tag.</td>
</tr>
</tbody>
</table>
```
using the TEMPLATE attribute.</p>
<cfoutput>\HTMLCodeFormat(" CFMODULE
    Template="myTag.cfm"
    X=3
    attributeCollection=#attrCollection1#
    Y=4")</cfoutput>
<p>The result: <cfoutput>$result$</cfoutput></p>
﻿
<!--- Call the tag with CFMODULE with Name. --->
<!---
<CFMODULE
    Name="myTag"
    X="3"
    attributeCollection=#attrCollection1#
    Y="4">
--->
<!--- Show the code. --->
<HR size="2" color="#0000A0">
<p>Here is another way to invoke the custom tag, using the NAME attribute.</p>
<cfoutput>\HTMLCodeFormat(" CFMODULE
    NAME='myTag'
    X=3
    attributeCollection=#attrCollection1#
    Y=4")</cfoutput>
<p>The result: <cfoutput>$result$</cfoutput></p>
﻿
<!--- Call the tag using the shortcut notation. --->
<!---
<CF_myTag
    X="3"
    attributeCollection=#attrCollection1#
    Y="4">
--->
<!--- Show the code. --->
<p>Here is the shortcut to invoking the same tag.</p>
<cfoutput>\HTMLCodeFormat("<cf_mytag
    x = 3
    attributeCollection = #attrcollection1#
    y = 4")</cfoutput>
<p>The result: <cfoutput>$result$</cfoutput></p>
cfNTauthenticate

Description
Authenticates a user name and password against the Windows NT domain on which the ColdFusion server is running, and optionally retrieves the user’s groups.

Category
Security tags

Syntax
<cfNTauthenticate
    username="username"
    password="password"
    domain="nt_domain"
    result="result variable"
    listGroups = "yes" or "no"
    throwOnError = "yes" or "no">
</cfNTauthenticate>

See also
cflogin, cfloginuser, IsUserInRole, GetAuthUser

History
ColdFusion MX 7: Added this tag.

Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Req/Opt</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Required</td>
<td>User's login name.</td>
<td></td>
</tr>
<tr>
<td>password</td>
<td>Required</td>
<td>User's password.</td>
<td></td>
</tr>
<tr>
<td>domain</td>
<td>Required</td>
<td>Domain against which to authenticate the user. The ColdFusion J2EE server must be running on this domain.</td>
<td></td>
</tr>
<tr>
<td>result</td>
<td>Optional</td>
<td>cfNTauthenticate Name of the variable in which to return the results.</td>
<td></td>
</tr>
<tr>
<td>listGroups</td>
<td>Optional</td>
<td>No</td>
<td>Boolean value specifying whether to include a comma-delimited list of the user’s groups in the result structure.</td>
</tr>
<tr>
<td>throwOnError</td>
<td>Optional</td>
<td>No</td>
<td>Boolean value specifying whether to throw an exception if the validation fails. If this attribute is Yes, ColdFusion throws an error if the user name or password is invalid; the application must handle such errors in a try/catch block or ColdFusion error handler page.</td>
</tr>
</tbody>
</table>

Usage
Use this function to authenticate a user against a Windows NT domain and optionally get the user’s groups. This function does not work with the Microsoft Active Directory directory service, and does nothing on UNIX and Linux systems. You typically use this tag inside a cflogin tag to authenticate the user for a cfloginuser tag, as shown in the example.
Note: ColdFusion must run as a user that has the privilege to authenticate other users in the specified domain.

The structure specified in the result attribute contains the following information:

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>auth</td>
<td>Whether the user is authenticated:</td>
</tr>
<tr>
<td></td>
<td>• Yes</td>
</tr>
<tr>
<td></td>
<td>• No</td>
</tr>
<tr>
<td>groups</td>
<td>A comma-delimited list of the user’s groups in the specified domain.</td>
</tr>
<tr>
<td></td>
<td>The structure includes this field only if the listGroups attribute is Yes.</td>
</tr>
<tr>
<td>name</td>
<td>The user name; equals the tag’s name attribute.</td>
</tr>
<tr>
<td>status</td>
<td>The authentication status. One of the following:</td>
</tr>
<tr>
<td></td>
<td>• success</td>
</tr>
<tr>
<td></td>
<td>• UserNotInDirFailure: the user is not listed in the directory.</td>
</tr>
<tr>
<td></td>
<td>• AuthenticationFailure: the user is in the directory, but the password is not valid.</td>
</tr>
</tbody>
</table>

This tag provides two models for handling authentication: status checking and exception handling. If the throwOnError attribute is No, use the result variable’s auth and status fields to determine whether the user was authenticated and, if not, the reason for the failure. If the throwOnError attribute is Yes, ColdFusion throws an exception error if the user is not valid. In this case, use try/catch error handling. The catch block must handle any authentication failure.

Example

The following example uses the auth and status fields to determine whether the user is authenticated and the failure cause. It consists of three files that you put in the same directory:

- A main cfntauthexample.cfm page that displays the name if the user is authenticated and contains a logout link.
- A login form page that is displayed if the user is not logged in.
- The Application.cfm page, which contains all the login, authentication, and logout processing code.

For a full description of login processing, see ColdFusion MX Developer’s Guide. For information on how this example works, see the comments in the code.

Save the following page as cfntauthenticateexample.cfm. To run the example, request this page in your browser or IDE.

<!--- The Application.cfm page, which is processed each time a user requests this page, ensures that you log in first. --->
<cfoutput>
<h3>Welcome #GetAuthUser()#</h3>
<!--- A link to log out the user. --->
<a href="#CGI.script_name?logout=Yes">Log Out</a>
</cfoutput>
Save the following page as loginform.cfm:

```coldfusion
<cfif NOT IsDefined("cflogin")>
<cfinclude template="loginform.cfm">
<cfabort>
<cfelse>
<cfset theusername=trim(form.j_username)>
<cfset thepassword=trim(form.j_password)>
<cfset thedomain=trim(form.domain)>
<cfntauthenticate username="#theusername#" password="#thepassword#" domain="#thedomain#" result="authresult" listgroups="yes">
<cfif authresult.auth>
<cfloginuser name="#theusername#" password="#thepassword#" roles="authresult.groups">
<cfelse>
<cfoutput>
<cfif authresult.status IS "AuthenticationFailure">
<H2>The password for #theusername# is not correct<br>Please Try again</H2>
<cfelse>
<H2>The user name #theusername# is not valid<br></H2>
</cfif>
</cfif>
</cfoutput>
<cfif authresult.status IS "AuthenticationFailure">
</cfelse>
</cfif>
</cfabort>
</cfelse>
</cfntauthenticate>
</cfif>
```

Save the following page as Application.cfm:

```coldfusion
<!--- If this page is executing in response to the user clicking a logout link, log out the user. The cflogin tag code will then run. --->
<cfif IsDefined("URL.logout") AND URL.logout>
<cflogout>
</cfif>
</cfif>

<!--- The cflogin body code runs only if a user is not logged in. --->
<cflogin>
<!--- cflogin variable exists only if login credentials are available. --->
<cfif NOT IsDefined("cflogin")>
<!--- Show a login form that posts back to the page whose request initiated the login, and do not process the rest of this page. --->
<cfinclude template="loginform.cfm">
<cfabort>
<cfelse>
<!--- Trim any leading or trailing spaces from the username and password submitted by the form. --->
<cfset theusername=trim(form.j_username)>
<cfset thepassword=trim(form.j_password)>
<cfset thedomain=trim(form.domain)>
<cfntauthenticate username="#theusername#" password="#thepassword#" domain="#thedomain#" result="authresult" listgroups="yes">
<cfif authresult.auth>
<!--- Log user in to ColdFusion and set roles to the user's Groups. --->
<cfloginuser name="#theusername#" password="#thepassword#" roles="authresult.groups">
<cfelse>
<!--- The user was not authenticated. Display an error message and the login form. --->
<cfoutput>
<cfif authresult.status IS "AuthenticationFailure">
<!--- The user is valid, but not the password. --->
<H2>The password for #theusername# is not correct<br><br>Please Try again</H2>
<cfelse>
<!--- There is one other status value, invalid user name. --->
<H2>The user name #theusername# is not valid<br></H2>
</cfif>
</cfif>
</cfoutput>
<cfif authresult.status IS "AuthenticationFailure">
</cfelse>
</cfabort>
</cfif>
</cfntauthenticate>
</cfif>
```
Please Try again</H2>
</cfif>
</cfoutput>
<cfinclude template="loginform.cfm">
<cfabort>
</cfif>
</cfif>
</cfif>
</cflogon>
cfobject

Description

Creates a ColdFusion object, of a specified type.

Note: You can enable and disable this tag in the ColdFusion Administrator page, under ColdFusion Security, Sandbox Security.

Category

Extensibility tags

Syntax

The tag syntax depends on the object type. Some types use the type attribute; others do not. See the following sections:

• “cfobject: COM object” on page 313
• “cfobject: component object” on page 315
• “cfobject: CORBA object” on page 316
• “cfobject: Java or EJB object” on page 318
• “cfobject: web service object” on page 320

Note: On UNIX, this tag does not support COM objects.

See also

cfargument, cfcomponent,cffunction,cfinvoke,cfinvokeargument,cfproperty, cfreturn; “Using Java objects” in Chapter 37, “Integrating J2EE and Java Elements in CFML Applications,” in ColdFusion MX Developer’s Guide

History

ColdFusion MX:

• Changed instantiation behavior: this tag, and the CreateObject function, can now instantiate ColdFusion components (CFCs); you can use them within the cfscript tag.
• For CORBA object: changed the Naming Service separator format for addresses from a dot to a forward slash. For example, if “context=NameService”, for a class, use either of the following formats for the class parameter:
  ■ "Macromedia/Eng/CF"
  ■ "Macromedia.current/Eng.current/CF"
  (In earlier releases, the format was "Macromedia.Eng.CF".)
• For CORBA object: changed the locale attribute; it specifies the Java configuration that contains the properties file.
**cfobject: COM object**

**Description**

Creates and manipulates a Component Object Model (COM) object. Invokes a registered automation server object type.

For information on OLEView, and about COM and DCOM, see the Microsoft OLE Development website: [www.microsoft.com](http://www.microsoft.com).

To use this tag, you must provide the object’s program ID or filename, the methods and properties available through the IDispatch interface, and the arguments and return types of the object’s methods. For most COM objects, you can get this information with the OLEView utility.

*Note:* On UNIX, this tag does not support COM objects.

**Syntax**

```xml
<cfobject
  type = "com"
  action = "action"
  class = "program_ID"
  name = "text"
  context = "context"
  server = "server_name">
</cfobject>
```

**See also**

`ReleaseComObject`, `cfcollection`, `cfexecute`; “COM” in Chapter 17, “Developing Globalized Applications,” in *ColdFusion MX Developer’s Guide*

**Attributes**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Req/Opt</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>Optional</td>
<td>com</td>
<td>Object type:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>corba</td>
<td>• com</td>
</tr>
<tr>
<td></td>
<td></td>
<td>java</td>
<td>• corba</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• java</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(The other object types do not take the <code>type</code> attribute.)</td>
</tr>
<tr>
<td>action</td>
<td>Required</td>
<td></td>
<td>• create: instantiates a COM object (typically, a DLL) before invoking methods or properties.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• connect: connects to a COM object (typically, an EXE) running on server.</td>
</tr>
<tr>
<td>class</td>
<td>Required</td>
<td></td>
<td>Component ProgID for the object to invoke. When using Java stubs to connect to the COM object, the class must be the ProgID of the COM object.</td>
</tr>
<tr>
<td>name</td>
<td>Required</td>
<td></td>
<td>String; name for the instantiated component.</td>
</tr>
</tbody>
</table>
Example

```
<h3>cfobject (COM) Example</h3>

<!--- Create a COM object as an inproc server (DLL). (class = prog-id)--->
<cfobject action = "Create"
type = "COM"
class = Allaire.DocEx1.1
name = "obj">

<!--- Call a method. Methods that expect no arguments should be called using empty parentheses. --->
<cfset obj.Init()>

<!--- This is a collection object. It should support, at a minimum:
   Property : Count
   Method : Item(inarg, outarg)
   and a special property called _NewEnum
--->
<cfoutput>
   This object has #obj.Count# items.
   <br> <HR>
</cfoutput>

<!--- Get the 3rd object in the collection. --->
<cfset emp = obj.Item(3)>
<cfoutput>
   The last name in the third item is #emp.lastname#.
   <br> <HR>
</cfoutput>

<!---Loop over all the objects in the collection.--->
<p>Looping through all items in the collection:
<br>
<cfloop
collection = #obj#
   item = file2>
   <cfoutput>Last name: #file2.lastname# <br></cfoutput>
</cfloop>
```
**cfoBJECT: component object**

**Description**

Creates an instance of a ColdFusion component (CFC) object.

**Syntax**

```xml
<cfobject
  name = "variable name"
  component = "component name">
```

**See also**


**Attributes**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Req/Opt</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Required</td>
<td>String; name for the instantiated component. The name must not have a period as the first or last character.</td>
<td></td>
</tr>
<tr>
<td>component</td>
<td>Required</td>
<td>Name of component to instantiate.</td>
<td></td>
</tr>
</tbody>
</table>

**Usage**

When the `cfoBJECT` tag creates an instance of the CFC, ColdFusion executes any constructor code in the CFC; that is, it runs code that is not in the method definitions.

On UNIX systems, ColdFusion searches first for a file with a name that matches the specified component name, but is all lowercase. If it does not find the file, it looks for a filename that matches the component name exactly, with the identical character casing.

**Example**

```xml
<!--- Separate instantiation and method invocation: permits multiple invocations. --->
<cfobject
  name="quoteService"
  component="nasdaq.quote">
<cfinvoke
  component="#quoteService#"
  method="getLastTradePrice"
  symbol="macr"
  returnVariable="res">
<cfoutput>#$res$</cfoutput><br>
<cfinvoke
  component="#quoteService#"
  method="getLastTradePrice"
  symbol="mot"
  returnVariable="res">
<cfoutput>#$res$</cfoutput>
```
cfobject: CORBA object

Description

Calls methods on a registered CORBA object.

Syntax

```html
cfobject
type = "corba"
context = "context"
class = "file or naming service"
name = "text"
locale = "type-value arguments"
```

See also

cfcollection, cfexecute, cfindex, cfreport, cfsearch, cfwddx; “CORBA” in Chapter 17, “Developing Globalized Applications,” in ColdFusion MX Developer's Guide

History

See the History section of the main cfobject tag page.

Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Req/Opt</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>Optional</td>
<td></td>
<td>Object type:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• com</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• corba</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• java</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(The other object types do not take the type attribute.)</td>
</tr>
<tr>
<td>context</td>
<td>Required</td>
<td></td>
<td>• ior: ColdFusion uses Interoperable Object Reference (IOR) to access CORBA server.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• nameservice: ColdFusion uses naming service to access server. This option is valid only with the InitialContext of a VisiBroker Orb.</td>
</tr>
<tr>
<td>class</td>
<td>Required</td>
<td></td>
<td>• If context = &quot;ior&quot;: absolute path of file that contains string version of the Interoperable Object Reference (IOR). ColdFusion must be able to read file; it should be local to ColdFusion server or accessible on network.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• If context = &quot;nameservice&quot;: forward slash-delimited naming context for naming service. For example: Allaire//Doc/empobject.</td>
</tr>
<tr>
<td>name</td>
<td>Required</td>
<td></td>
<td>String; name for the instantiated component. An application uses it to reference the CORBA object’s methods and attributes.</td>
</tr>
<tr>
<td>locale</td>
<td>Optional</td>
<td></td>
<td>Sets arguments for a call to init_orb. Use of this attribute is specific to VisiBroker ORBs. It is available on C++, Version 3.2. The value must be in the form:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>locale = &quot; -ORBagentAddr 199.99.129.33 -ORBagentPort 19000&quot;</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Each type-value pair must start with a hyphen.</td>
</tr>
</tbody>
</table>
Usage

ColdFusion Enterprise version 4.0 and later supports CORBA through the Dynamic Invocation Interface (DII). To use `cfobject` with CORBA objects, you must provide the name of the file that contains a string-formatted version of the IOR, or the object's naming context in the naming service; and the object's attributes, method names, and method signatures.

User-defined types (for example, structures) are not supported.

Example

```cfc
<cfobject type = "corba"
  context = "ior"
  class = "c:\\myobject.ior"
  name = "GetName">
```
**cfobject: Java or EJB object**

**Description**

Creates and manipulates a Java and Enterprise Java Bean (EJB) object.

**Syntax**

```cfobject
  type = "Java"
  action = "Create"
  class = "Java class"
  name = "object name"
</cfobject>
```

**See also**

collection, cexecute, cindex, creport, cfsearch, cfwddx; “Using Java objects” in Chapter 37, “Integrating J2EE and Java Elements in CFML Applications,” in *ColdFusion MX Developer’s Guide*

**Attributes**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Req/Opt</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>Optional</td>
<td></td>
<td>Object type:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• com</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• corba</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• java</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(The other object types do not take the type attribute.)</td>
</tr>
<tr>
<td>action</td>
<td>Required</td>
<td></td>
<td>Create: Creates a Java or WebLogic Environment object.</td>
</tr>
<tr>
<td>class</td>
<td>Required</td>
<td></td>
<td>Java class.</td>
</tr>
<tr>
<td>name</td>
<td>Required</td>
<td></td>
<td>String; name for the instantiated component.</td>
</tr>
</tbody>
</table>

**Usage**

To call Java CFXs or Java objects, ColdFusion uses a Java Virtual Machine (JVM) that is embedded in the process. You can configure JVM loading, location and settings in the ColdFusion Administrator.

Any Java class available in the class path that is specified in the ColdFusion Administrator can be loaded and used from ColdFusion, using the `cfobject` tag.

To access Java methods and fields, do the following steps:

1. Call the `cfobject` tag, to load the class. See the example code.
2. Use the `init` method with appropriate arguments, to call a constructor. For example:
   ```cfset ret = myObj.init(arg1, arg2)```

Calling a public method on the object without first calling the `init` method results in an implicit call to the default constructor. Arguments and return values can be any Java type (simple, array, object). ColdFusion makes the conversions if strings are passed as arguments, but not if they are received as return values.

Overloaded methods are supported if the number of arguments is different.
**Calling EJBs**

To create and call EJB objects, use the `cfobject` tag. In the second example below, the WebLogic JNDI is used to register and find EJBHome instances.

**Example**

<!--- Example of a Java Object, this cfobject call loads the class MyClass but does not create an instance object. Static methods and fields are accessible after a call to cfobject. --->

```clobby
<cfobject
  action = "create"
  type = "java"
  class = "myclass"
  name = "myobj">
</cfobject>

<!--- Example of an EJB - The cfobject tag creates the Weblogic Environment object, which is used to get InitialContext. The context object is used to look up the EJBHome interface. The call to Create() results in getting an instance of stateless session EJB. --->

```clobby
<cfobject
  action = "create"
  type = "java"
  class = "weblogic/jndi/Environment"
  name = "wlEnv">
</cfobject>

<cfset ctx = wlEnv.getInitialContext()>
<cfset ejbHome = ctx.lookup("statelessSession.TraderHome")>
<cfset trader = ejbHome.Create()>
<cfset value = trader.shareValue(20, 55.45)>
<cfoutput>
  Share value = #value#
</cfoutput>
<cfset value = trader.remove()>
```
**cfobject: web service object**

**Description**

Creates a web service proxy object.

**Syntax**

```<cfobject
    webservice= "http://....?wsdl" or "name set in Administrator"
    name = "myobjectname">
```

**See also**

`cfcollection`, `cfexec`, `cfindex`, `cfreport`, `cfsearch`, `cfwddx`; “Consuming web services” in Chapter 36, “Using Web Services,” in *ColdFusion MX Developer’s Guide*

**Attributes**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Req/Opt</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>webservice</td>
<td>Required</td>
<td>URL to web service WSDL file:</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Absolute URL of web service.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Name (string) assigned in the Administrator to the web service.</td>
<td></td>
</tr>
<tr>
<td>name</td>
<td>Required</td>
<td>Local name for the web service. String.</td>
<td></td>
</tr>
</tbody>
</table>

**Usage**

Instantiates a proxy object for a web service. You can enter the absolute URL in this tag, or refer to a web service that is entered in the ColdFusion Administrator. To minimize potential code maintenance, enter the web service in the Administrator, and then refer to that name in this tag.
**cfobjectcache**

**Description**
Flushes the query cache.

**Category**
Database manipulation tags

**Syntax**
```<cfobjectcache action = "clear">```

**See also**
cfobject

**History**
ColdFusion 5: Added this tag.

**Attributes**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Req/Opt</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>action</td>
<td>Required</td>
<td>clear: clears queries from the cache in the Application scope.</td>
<td></td>
</tr>
</tbody>
</table>
**cfoutput**

**Description**
Displays output that can contain the results of processing ColdFusion variables and functions. Can loop over the results of a database query.

**Category**
Data output tags

**Syntax**
```
<cfoutput
    query = "query_name"
    group = "query_column"
    groupCaseSensitive = "yes" or "no"
    startRow = "start_row"
    maxRows = "max_rows_output">
</cfoutput>
```

**See also**
cfcol, cfcontent, cfdirectory, cftable

**History**
ColdFusion 4.5.0: Added the groupCaseSensitive attribute.

**Attributes**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Req/Opt</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>query</td>
<td>Optional</td>
<td>Name of cfquery from which to draw data for output section.</td>
<td></td>
</tr>
<tr>
<td>group</td>
<td>Optional</td>
<td>Query column to use to group sets of records. Eliminates adjacent duplicate rows when data is sorted. Use if you retrieved a record set ordered on one or more a query columns. For example, if a record set is ordered on &quot;Customer_ID&quot; in the cfquery tag, you can group the output on &quot;Customer_ID.&quot;</td>
<td></td>
</tr>
<tr>
<td>groupCaseSensitive</td>
<td>Optional</td>
<td>Yes</td>
<td>Boolean. Whether to consider the case in grouping rows.</td>
</tr>
<tr>
<td>startRow</td>
<td>Optional</td>
<td>1</td>
<td>Row from which to start output.</td>
</tr>
<tr>
<td>maxRows</td>
<td>Optional</td>
<td>displays all rows</td>
<td>Maximum number of rows to display.</td>
</tr>
</tbody>
</table>

**Usage**
In the cfoutput tag body, ColdFusion treats text that is surrounded by number signs (#) as a ColdFusion variable or function call. For example, the following code displays the text "Hello World!":

```
<cfset myVar="Hello World!">
<cfoutput>#{myVar}#</cfoutput>
```
When you specify a `query` attribute, this tag loops over the query rows and produces output for each row within the range specified by the `startRow` and `maxRows` values, and groups or eliminates duplicate entries as specified by the grouping attribute values, if any. It also sets the `query.currentRow` variable to the current row being processed.

If you nest `cfoutput` blocks that process a query, you specify the `query` and `group` attributes at the top-most level; you can specify a `group` attribute for each inner block except the innermost `cfoutput` block.

This tag requires an end tag.

**Example**

<!--- EXAMPLE: This example shows how cfoutput operates. --->
<!--- Run a sample query. --->
<cfquery name = "GetCourses" dataSource = "cfdocexamples">
  SELECT Dept_ID, CorName, CorLevel
  FROM courseList
  ORDER by Dept_ID, CorLevel, CorName
</cfquery>
<h3>cfoutput Example</h3>
<p>cfoutput tells ColdFusion Server to begin processing, and then to hand back control of page rendering to the web server. If you enclosed that expression in cfoutput, the result would be</p><cfoutput>cfoutput豸Datefomat(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>

<p>There are <cfoutput>#getCourses.recordCount#</cfoutput> total records in our query. Using the maxRows parameter, we are limiting our display to 4 rows.</p>

<cfquery name = "GetCourses"(dataSource = "cfdocexamples") maxRows = 4>
  #Dept_ID# #CorName# #CorLevel#<br>
</cfquery>

<p>EXAMPLE: The next example uses the group attribute to eliminate duplicate lines from a list of course levels taught in each department.</p>

<cfquery name = "GetCourses" dataSource = "cfdocexamples">
  SELECT Dept_ID, CorLevel
  FROM courseList
  ORDER by Dept_ID, CorLevel
</cfquery>

<p>cfoutput can also show the results of a more complex expression, such as getting the day of the week from today's date. We first extract the integer representing the Day of the Week from the server function `Now()` and then apply the result to the `DayOfWeekAsString` function:

<br>Today is #DayOfWeekAsString(DayOfWeek(Now()))#
Today is <cfoutput>#DayofWeekAsString(DayofWeek(Now()))#</cfoutput>

<cfquery datasource="cfdocexamples" name="empSalary">
SELECT Emp_ID, firstname, lastname, e.dept_id, salary, d.dept_name
FROM employee e, departmt d
WHERE e.dept_id = d.dept_id
ORDER BY d.dept_name
</cfquery>

<!--- Outer cfoutput. --->
<cfoutput query="empSalary" group="dept_id">
<h2>#dept_name#</h2>
<table width="95%" border="2" cellspacing="2" cellpadding="2">
<tr>
<th>Employee</th>
<th>Salary</th>
</tr>
<cfset deptTotal = 0>
<!--- Inner cfoutput. --->
<cfoutput>
<tr>
<td>#empSalary.lastname#, #empSalary.firstname#</td>
<td align="right">#DollarFormat(empSalary.salary)#</td>
</tr>
<cfset deptTotal = deptTotal + empSalary.salary>
<tr>
<td align="right">Total</td>
<td align="right">#DollarFormat(deptTotal)#</td>
</tr>
<cfset deptTotal = 0>
</table>
</cfoutput>
</cfoutput>
cfparam

Description
Tests for the existence of a parameter (that is, a variable), validates its data, and, if a default value is not assigned, optionally provides one.

Category
Variable manipulation tags

Syntax
<cfparam
  name = "param_name"
  type = "data_type"
  default = "value"
  max = "value"
  min = "value"
  pattern = "regular expression">

See also
cfcookie, cfregistry, cfsavecontent, cfschedule, cfset; “Validating data with the IsValid function and the cfparam tag” in Chapter 28, “Validating Data,” in ColdFusion MX Developer’s Guide

History
ColdFusion MX 7:
• Added min, max, and pattern attributes.
• Added creditcard, email, eurodate, float, integer, range, regex, regular_expression, ssn, social_security_number, time, URL, USdate, and zipcode attributes of the type attribute.
<table>
<thead>
<tr>
<th>Attribute</th>
<th>Req/Opt</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Required</td>
<td></td>
<td>Name of the parameter (variable) to test (such as &quot;Client.Email&quot; or &quot;Cookie.BackgroundColor&quot;). If omitted, and if the parameter does not exist, an error is thrown.</td>
</tr>
<tr>
<td>type</td>
<td>Optional</td>
<td>any</td>
<td>The valid format for the data; one of the following. For detailed information on validation algorithms, see &quot;Validating form data using hidden fields&quot; in Chapter 28, &quot;Validating Data,&quot; in ColdFusion MX Developer's Guide.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• any: any type of value.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• array: an array of values.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• binary: a binary value.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• boolean: a Boolean value: yes, no, true, false, or a number.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• creditcard: a 13-16 digit number conforming to the mod10 algorithm.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• date or time: a date-time value.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• email: a valid e-mail address.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• eurodate: a date-time value. Any date part must be in the format dd/mm/yy, The format can use /, -, or . characters as delimiters.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• float or numeric: a numeric value.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• guid: a Universally Unique Identifier of the form &quot;XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX&quot; where 'X' is a hexadecimal number.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• integer: an integer.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• query: a query object.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• regex or regular_expression: matches input against pattern attribute.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• ssn or social_security_number: a U.S. social security number.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• string: a string value or single character.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• struct: a structure.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• telephone: a standard U.S. telephone number.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• URL: an http, https, ftp, file, mailto, or news URL.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• UUID: a ColdFusion Universally Unique Identifier, formatted &quot;XXXXXXXX-XXXX-XXXX-XXXXXXXXXXXXX&quot;, where 'X' is a hexadecimal number. See CreateUUID on page 544.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• USdate: a U.S. date of the format mm/dd/yy, with 1-2 digit days and months, 1-4 digit years.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• variableName: a string formatted according to ColdFusion variable naming conventions.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• xml: XML objects and XML strings.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• zipcode: U.S. 5- or 9-digit format ZIP codes.</td>
</tr>
<tr>
<td>default</td>
<td>Optional</td>
<td></td>
<td>Value to set parameter to if it does not exist. Any expression used for the default attribute is evaluated, even if the parameter exists. The result is not assigned if the parameter exists, but if the expression has side effects, they still occur.</td>
</tr>
<tr>
<td>max</td>
<td>Optional</td>
<td></td>
<td>The maximum valid value; used only for range validation.</td>
</tr>
</tbody>
</table>
Usage

You can use this tag to make the following tests:

- To test whether a required variable exists, use this tag with only the name attribute. If it does not exist, ColdFusion MX stops processing the page and returns an error.
- To test whether a required variable exists, and that it is of the specified type, use this tag with the name and type attributes. If the variable does not exist or its value is not of the specified type, ColdFusion returns an error.
- To set a default value for the variable, use this tag with the name and default attributes. If the variable does not exist, it is created and set to the default attribute value. If the variable exists, processing continues; the value is not changed.

If you specify variableName for the type attribute, the parameter's value must be a string that is in ColdFusion variable name format; that is, starts with a letter, underscore (_), or Unicode currency symbol, and contains letters, numbers, underscores, periods, and Unicode currency symbols, only. ColdFusion does not check whether the parameter value corresponds to an existing ColdFusion variable.

Tip: To improve performance, avoid using the cfparam tag in ColdFusion functions, including in CFC methods. Instead, place the cfparam tags in the body of the CFML pages.

Example

<!--- This example shows how to use CFPARAM to define default values for page variables. -------->
<cfparam name = "storeTempVar" default = "my default value">
<cfparam name = "tempVar" default = "my default value">

<!--- Check if form.tempVar was passed. --->
<cfif IsDefined("form.tempVar") is "True">
  <!--- Check if form.tempVar is not blank. --->
  <cfif form.tempVar is not "">
    <!--- If not, set tempVar to value of form.tempVar --->
    <cfset tempVar = form.tempVar>
  </cfif>
</cfif>

<body>
<h3>cfparam Example</h3>
<p>cfparam is used to set default values so that a developer does not have to check for the existence of a variable using a function like IsDefined.</p>
<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 --->
<cfoutput>
</cfoutput>
<cfif tempVar is not #StoreTempVar# and tempVar is not "">
    <h3>The value of tempVar has changed: the new value is #tempVar#</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 = "submit" value = "submit">
</form>
</p>
cfpop

Description
Retrieves or deletes e-mail messages from a POP mail server.

Category
Internet Protocol tags

Syntax
<cfpop
  server = "servername"
  port = "port_number"
  username = "username"
  password = "password"
  action = "action"
  name = "queryname"
  messageNumber = "number"
  uid = "number"
  attachmentPath = "path"
  timeout = "seconds"
  maxRows = "number"
  startRow = "number"
  generateUniqueFilenames = "yes" or "no"
  debug = "yes" or "no">

See also
cfftp, cfhttp, cfldap, cfmail, cfmailparam, SetLocale; Chapter 39, “Sending and Receiving E-Mail” in ColdFusion MX Developer's Guide.

History
ColdFusion MX 6.1:
• Added support for multipart mail messages with Text and HTML parts.
• Changed the attachment name separator: the TAB character is now the separator between attachment names in the attachments and attachmentfiles query fields if a message has multiple attachments. This behavior is identical to ColdFusion 5 and earlier versions.

ColdFusion MX: Changed the attachment name separator: the comma separates names in the attachments and attachmentfiles query fields if a message has multiple attachments.

Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Req/Opt</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>server</td>
<td>Required</td>
<td></td>
<td>POP server identifier:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• A host name; for example, &quot;biff.upperlip.com&quot;.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• An IP address; for example, &quot;192.1.2.225&quot;.</td>
</tr>
<tr>
<td>port</td>
<td>Optional</td>
<td>110</td>
<td>POP port.</td>
</tr>
<tr>
<td>username</td>
<td>Optional</td>
<td></td>
<td>A user name.</td>
</tr>
<tr>
<td>Attribute</td>
<td>Req/Opt</td>
<td>Default</td>
<td>Description</td>
</tr>
<tr>
<td>------------</td>
<td>-------------</td>
<td>------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>password</td>
<td>Optional</td>
<td></td>
<td>Password that corresponds to <code>username</code>.</td>
</tr>
<tr>
<td>action</td>
<td>Optional</td>
<td>getHeaderOnly</td>
<td>• <code>getHeaderOnly</code>: returns message header information only</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• <code>getAll</code>: returns message header information, message text, and attachments if <code>attachmentPath</code> is specified</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• <code>delete</code>: deletes messages on POP server</td>
</tr>
<tr>
<td>name</td>
<td>Required if <code>action</code> = &quot;getAll&quot; or &quot;getHeaderOnly&quot;</td>
<td></td>
<td>Name for query object that contains the retrieved message information.</td>
</tr>
<tr>
<td>message</td>
<td>Number</td>
<td></td>
<td>Message number or comma-delimited list of message numbers to get or delete. Invalid message numbers are ignored. Ignored if <code>uid</code> is specified.</td>
</tr>
<tr>
<td>uid</td>
<td></td>
<td></td>
<td>UID or a comma-delimited list of UIDs to get or delete. Invalid UIDs are ignored.</td>
</tr>
<tr>
<td>attachment</td>
<td>Optional</td>
<td></td>
<td>If <code>action</code> = &quot;getAll&quot;, specifies a directory in which to save any attachments. If the directory does not exist, ColdFusion creates it. If you omit this attribute, ColdFusion does not save any attachments. If you specify a relative path, the path root is the ColdFusion temporary directory, which is returned by the <code>GetTempDirectory</code> function.</td>
</tr>
<tr>
<td>timeout</td>
<td>Optional</td>
<td>60</td>
<td>Maximum time, in seconds, to wait for mail processing.</td>
</tr>
<tr>
<td>maxRows</td>
<td>Optional</td>
<td>retrieves all available rows</td>
<td>Number of messages to return or delete, starting with the number in <code>startRow</code>. Ignored if <code>messageNumber</code> or <code>uid</code> is specified.</td>
</tr>
<tr>
<td>startRow</td>
<td>Optional</td>
<td>1</td>
<td>First row number to get or delete. Ignored if <code>messageNumber</code> or <code>uid</code> is specified.</td>
</tr>
</tbody>
</table>
Usage

The **cfpop** tag retrieves one or more mail messages from a POP server and populates a ColdFusion query object with the resulting messages, one message per row. Alternatively, it deletes one or more messages from the POP server.

**Note:** When the **cfpop** tag encounters malformed mail messages, it does not generate errors; instead, it returns empty fields.

To optimize performance, two retrieve options are available. Message header information is typically short, and therefore quick to transfer. Message text and attachments can be very long, and therefore take longer to process.

**cfpop query variables**

The following table describes the variables that provide information about the query that is returned by **cfpop**:

<table>
<thead>
<tr>
<th>Variable names</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>queryname.recordCount</td>
<td>Number of records returned by query</td>
</tr>
<tr>
<td>queryname.currentRow</td>
<td>Current row that <strong>cfoutput</strong> is processing</td>
</tr>
<tr>
<td>queryname.columnList</td>
<td>List of column names in query</td>
</tr>
<tr>
<td>queryname.UID</td>
<td>Unique identifier for the e-mail message file</td>
</tr>
</tbody>
</table>
Query message header and body columns

The following table lists the message header and body columns that are returned if `action = "getHeaderOnly"` or "getAll":

<table>
<thead>
<tr>
<th>Column name</th>
<th>getHeaderOnly returns</th>
<th>getAll returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>queryname.date</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>queryname.from</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>queryname.messagenumber</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>queryname.messageid</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>queryname.replyto</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>queryname.subject</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>queryname.cc</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>queryname.to</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>queryname.body</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>queryname.textBody</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>queryname.HTMLBody</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>queryname.header</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>queryname.attachments</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>queryname.attachmentfiles</td>
<td>no</td>
<td>yes</td>
</tr>
</tbody>
</table>

If the mail message includes a part with a Content-Type of text/plain, the `queryname.textBody` column contains the part’s message content. If the mail message includes a part with a Content-Type of text/HTML, the `queryname.HTMLBody` column contains the part’s message content. If no Content-Type matches these types, the columns are empty. The `queryname.Body` column always contains the first message body found.

The `queryname.attachments` column contains a tab-separated list of all the attachment names. The `queryname.attachmentfiles` column contains a tab-separated list of the locations of the attachment files. Use the `cffile` tag to delete these temporary files when you have processed them.

To create a ColdFusion date/time object from the date-time string that is extracted from a mail message in the `queryname.date` column, use the following table:

<table>
<thead>
<tr>
<th>Locale</th>
<th>How to create a ColdFusion date/time object from <code>queryname.date</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>English (US)</td>
<td>Use the <code>ParseDateTime</code> function. If you specify the <code>pop-conversion</code> attribute, the function adjusts the date/time object to UTC.</td>
</tr>
<tr>
<td>Other</td>
<td>Extract the date part of string; pass it to the <code>LSParseDateTime</code> function.</td>
</tr>
</tbody>
</table>

*Note: To set the default display format of date, time, number, and currency values, use the `SetLocale` function.*
For more information on `cfpop`, see Chapter 39, “Sending and Receiving E-Mail” in *ColdFusion MX Developer's Guide*.

**Example**

<!--- This view-only example shows the use of cfpop. --->

<h3>cfpop Example</h3>

<p>cfpop lets you retrieve and manipulate mail in a POP3 mailbox. This view-only example shows how to create one feature of a mail client to display the mail headers in a POP3 mailbox.

To execute this, uncomment this code and run with a mail-enabled CF Server.

<!---
<cfif IsDefined("form.server ")>
    <!--- Make sure server, username are not empty. --->
    <cfif form.server is not "" and form.username is not ">
        <cfpop server = "" username = "" password = "">
            action = "getHeaderOnly" name = "GetHeaders">
            <h3>Message Headers in Your Inbox</h3>
            <p>Number of Records:
                <cfoutput>#GetHeaders.recordCount#</cfoutput></p>
                <ul>
                    <cfoutput query = "GetHeaders">
                        <li>Row: #currentRow#: From: #From# -- Subject: #Subject#</li>
                    </cfoutput>
                </ul>
        </cfif>
    </cfif>
</cfif>

<form action = "cfpop.cfm " method = "post">
    Enter your mail server:
    <input type = "Text" name = "popserver">
    Enter your username:
    <input type = "Text" name = "username">
    Enter your password:
    <input type = "password" name = "pwd">
    <input type = "Submit" name = "get message headers">
</form>

---
cfprocessingdirective

Description

Provides the following information to ColdFusion about how to process the current page:

- Specifies whether to remove excess whitespace character from ColdFusion generated content in the tag body.
- Identifies the character encoding (character set) of the page contents.

Category

Data output tags, Page processing tags

Syntax

<cfprocessingdirective
    pageencoding = "page-encoding literal string" />
or
<cfprocessingdirective
    suppressWhiteSpace = "yes" or "no"
    pageEncoding = "page-encoding literal string">
    CFML tags
</cfprocessingdirective>

See also

cfcol, cfcontent, cfoutput, cfsetting, cfsilent, cftable, SetEncoding; Chapter 17, “Developing Globalized Applications” in ColdFusion MX Developer’s Guide

History

ColdFusion MX:

- Changed suppresswhitespace attribute value behavior: you can specify the suppresswhitespace attribute value as a string variable. (ColdFusion 5 supported setting it only as a constant.)
- Added the pageEncoding attribute.
Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Req/Opt</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>suppressWhiteSpace</td>
<td>Optional</td>
<td>Boolean</td>
<td>Boolean; whether to suppress white space characters within the <code>cfprocessingdirective</code> block that are generated by CFML tags and often do not affect HTML appearance. Does not affect any white space in HTML code.</td>
</tr>
<tr>
<td>pageEncoding</td>
<td>Optional</td>
<td>Character encoding</td>
<td>A string literal; cannot be a variable. Identifies the character encoding of the current CFML page. This attribute affects the entire page, not just the <code>cfprocessing</code> tag body. The value may be enclosed in single- or double-quotation marks, or none. The following list includes commonly used values: utf-8, iso-8859-1, windows-1252, us-ascii, shift_jis, iso-2022-jp, euc-jp, euc-kr, big5, euc-cn, utf-16. For more information on character encodings, see <a href="http://www.w3.org/International/O-charset.html">www.w3.org/International/O-charset.html</a>.</td>
</tr>
</tbody>
</table>

Usage

The `cfprocessingdirective` tag has limitations that depend on the attribute you use. For this reason, Macromedia recommends that you include either the `pageEncoding` or `suppressWhiteSpace` attribute in a `cfprocessingdirective` tag, not both. To specify both values, use separate tags.

In a ColdFusion component (.cfc file), the `cfprocessingdirective` tag must follow the `cfcomponent` tag.

If you use the `pageEncoding` attribute, the following rules apply:

- You must put the tag within the first 4096 bytes of a page. It can be positioned after a `cfsetting` or `cfsilent` tag.
- If you use the tag on a page that includes other pages by using the `cfinclude` or `cfmodule` tags, custom tag invocation, and so on, the tag has no effect on the included pages.
• You cannot embed the tag within conditional logic, because the pageEncoding attribute is evaluated when ColdFusion compiles a page (not when it executes the page). For example, the following code has no effect at execution time, because the cfprocessingdirective tag has already been evaluated:

```html
<cfif dynEncoding is not "dynamic encoding is not possible">
  <cfprocessingdirective pageencoding=#dynEncoding#>
</cfprocessingdirective>
</cfif>
```

• If you have multiple cfprocessingdirective tags in one page that specify the pageEncoding attribute, they must all specify the same value; if not, ColdFusion throws an error.

• If you specify only the pageencoding attribute, do not use a separate end tag.

• ColdFusion accepts character encoding names that are supported by the Java platform. If an invalid name is specified, ColdFusion throws an InvalidEncodingSpecification exception.

• If a page has a byte order mark (BOM), and a pageencoding attribute specifies an encoding that differs from the BOM, ColdFusion generates an error.

The following rules apply to the suppressWhiteSpace attribute:

• You can specify the suppresswhitespace attribute value as a constant or a variable. To use a variable: define the variable (for example, whitespaceSetting), assign it the value yes or no, and code a statement such as the following:

```html
<!--- ColdFusion allows suppression option to be set at runtime --->
<cfprocessingdirective suppresswhitespace=#whitespaceSetting#>
  code to whose output the setting is applied
</cfprocessingdirective>
```

• The suppresswhitespace attribute only affects code that you put between the cfprocessingdirective begin tag and the cfprocessingdirective end tag.

The following example shows the use of a nested cfprocessingdirective tag. The outer tag suppresses unnecessary whitespace during computation of a large table; the inner tag retains whitespace, to output a preformatted table.

**Example**

```html
<cfprocessingdirective suppressWhiteSpace = "Yes">
  <!--- CFML code --->
  <cfprocessingdirective suppressWhiteSpace = "No">
    <cfoutput>#table_data#</cfoutput>
  </cfprocessingdirective>
</cfprocessingdirective>
```

The following example shows the use of the pageencoding attribute:

```html
<cfprocessingdirective pageencoding = "shift_jis">
**cfprocparam**

**Description**

Defines stored procedure parameters. This tag is nested within a `cfstoredproc` tag.

**Category**

*Database manipulation tags*

**Syntax**

```
<cfprocparam
  type = "in" or "out" or "inout"
  variable = "variable name"
  value = "parameter value"
  CFSQLType = "parameter datatype"
  maxLength = "length"
  scale = "decimal places"
  null = "yes" or "no">
```

**See also**

`cfinsert`, `cfprocresult`, `cfquery`, `cfqueryparam`, `cfstoredproc`, `cftransaction`, `cfupdate`; “Optimizing ColdFusion applications” in Chapter 13, “Designing and Optimizing a ColdFusion Application,” in *ColdFusion MX Developer’s Guide*

**History**

ColdFusion MX:

- The `maxrows` attribute is obsolete.
- Changed the `dbvarname` attribute behavior: it is now ignored for all drivers. ColdFusion MX uses JDBC 2.2 and does not support named parameters.
- Changed the `maxLength` attribute behavior: it now applies to IN and INOUT parameter values.

**Attributes**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Req/Opt</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
</table>
| type      | Optional| in      | • in: the parameter is used to send data to the database system only. Passes the parameter by value.  
|           |         |         | • out: the parameter is used to receive data from the database system only. Passes the parameter as a bound variable.  
|           |         |         | • inout: the parameter is used to send and receive data. Passes the parameter as a bound variable. |
| variable  | Required if `type = "OUT"` or "INOUT" |          | ColdFusion variable name; references the value that the output parameter has after the stored procedure is called. This is ignored for IN parameters. |
| value     | Required if `type = "IN"` |          | Value that ColdFusion passes to the stored procedure. This is optional for INOUT parameters. |
Usage

Use this tag to identify stored procedure parameters and their data types. Code one cfprocparam tag for each parameter. The parameters that you code vary based on parameter type and DBMS. ColdFusion MX supports positional parameters only and you must code cfprocparam tags in the same order as the associated parameters in the stored procedure definition.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Req/Opt</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CFSQLEType</td>
<td>Required</td>
<td>SQL type to which the parameter (any type) is bound. ColdFusion supports the following values, where the last element of the name corresponds to the SQL data type. Different database systems might support different subsets of this list. See your DBMS documentation for information on supported parameter types.</td>
<td></td>
</tr>
<tr>
<td>maxLength</td>
<td>Optional</td>
<td>0</td>
<td>Maximum length of a string or character IN or INOUT value attribute. A maxLength of 0 allows any length. The maxLength attribute is not required when specifying type=out.</td>
</tr>
<tr>
<td>scale</td>
<td>Optional</td>
<td>0</td>
<td>Number of decimal places in numeric parameter. A scale of 0 limits the value to an integer.</td>
</tr>
<tr>
<td>null</td>
<td>Optional</td>
<td>No</td>
<td>Whether the parameter is passed in as a null value. Not used with OUT type parameters.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Yes: tag ignores the value attribute.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• No</td>
</tr>
</tbody>
</table>

For a mapping of ColdFusion SQL data types to JDBC data types, See also cfqueryparam.
Output variables are stored in the ColdFusion variable specified by the `variable` attribute.

You cannot use the `cfprocparam` tag for Oracle 8 and 9 reference cursors. Instead, use the `cfprocresult` tag.

**Example**

The following example shows how to invoke an Oracle 8 PL/SQL stored procedure. It makes use of Oracle 8 support of the Reference Cursor type.

The following package, `Foo_Data`, houses a procedure `refcurproc` that declares output parameters as Reference Cursor:

- Parameter `pParam1` returns the rows in the EMP table
- Parameter `pParam2` returns the rows in the DEPT table

The procedure declares one input parameter as an integer, and one output parameter as a two-byte char varying type. Before the `cfstoredproc` tag can call this procedure, it must be created, compiled, and bound in the RDBMS environment.

```plaintext
CREATE OR REPLACE PACKAGE Foo_Data AS
    TYPE EmpTyp IS REF CURSOR RETURN Emp%ROWTYPE;
    TYPE DeptTyp IS REF CURSOR RETURN Dept%ROWTYPE;
    PROCEDURE refcurproc(pParam1 in out EmpTyp, pParam2 in out DeptTyp,
                           pParam3 in integer, pParam4 out varchar2);
END foo_data;

CREATE OR REPLACE PACKAGE BODY Foo_Data AS
    PROCEDURE RefCurProc(pParam1 in out EmpTyp,
                          pParam2 in out DeptTyp,
                          pParam3 in integer,
                          pParam4 out varchar2) IS
        BEGIN
            OPEN pParam1 FOR select * from emp;
            OPEN pParam2 FOR select * from dept;
            IF pParam3 = 1
                THEN
                    pParam4 := 'hello';
                ELSE
                    pParam4 := 'goodbye';
                END IF;
        END RefCurProc;
END Foo_Data;
```

The following CFML example shows how to invoke the `RefCurProc` procedure using `cfstoredproc`, `cfprocparam`, and `cfprocresult`:

```cfml
<cfstoredproc  procedure = "foo_data.refcurproc"
    dataSource = "oracle8i"
    username = "scott"
    password = "tiger"
    returnCode = "No">
    <cfprocparam type = "Out" CFSQLType = "CF_SQL_REFCURSOR"
                  variable = "param1">
    <cfprocparam type = "Out" CFSQLType = "CF_SQL_REFCURSOR"
                  variable = "param2">
</cfstoredproc>
```
<cfprocparam type = "IN" CFSQLType = "CF_SQL_INTEGER" value = "1">
<cfprocparam type = "OUT" CFSQLType = "CF_SQL_VARCHAR"
variable = "FOO">
<cfprocresult name = "rs1">
<cfprocresult name = "rs2" resultSet = "2">
</cfstoredproc>

<b>The first result set:</b><br>
<hr>
<cftable query = "rs1" colHeaders HTMLTable border = "1">
<cfcol header = "EMPNO" text = "#EMPNO#">
<cfcol header = "EMPLOYEE name" text = "#ENAME#">
<cfcol header = "JOB" text = "#JOB#">
<cfcol header = "SALARY" text = "#SAL#">
<cfcol header = "DEPT NUMBER" text = "#DEPTNO#">
</cftable>
<hr>

<b>The second result set:</b><br>
<hr>
<cftable query = "rs2" colHeaders HTMLTable border = "1">
<cfcol header = "DEPT name" text = "#DNAME#">
<cfcol header = "DEPT NUMBER" text = "#DEPTNO#">
</cftable>
<hr>

<b>The output parameter is:</b>'#FOO#'
</cfoutput>
cfprocresult

Description

Associates a query object with a result set returned by a stored procedure. Other ColdFusion tags, such as cfoutput and cftable, use this query object to access the result set. This tag is nested within a cfstoredproc tag.

Category

Database manipulation tags

Syntax

```
<cfprocresult
    name = "query_name"
    resultSet = "1-n"
    maxRows = "maxrows">
```

See also
cfinsert, cfprocparam, cfquery, cfqueryparam, cfstoredproc, cftransaction, cfupdate;
“Optimizing database use” in Chapter 13, “Designing and Optimizing a ColdFusion Application,” in ColdFusion MX Developer’s Guide

Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Req/Opt</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Required</td>
<td></td>
<td>Name for the query result set.</td>
</tr>
<tr>
<td>resultSet</td>
<td>Optional</td>
<td>1</td>
<td>Names one result set, if stored procedure returns more than one.</td>
</tr>
<tr>
<td>maxRows</td>
<td>Optional</td>
<td>-1(All)</td>
<td>Maximum number of rows returned in result set.</td>
</tr>
</tbody>
</table>

Usage

To enable access to data returned by the stored procedure, specify one or more cfprocresult tags. If the stored procedure returns more than one result set, use the resultSet attribute to specify which of the stored procedure’s result sets to return.

The resultSet attribute must be unique within the scope of the cfstoredproc tag. If you specify a result set twice, the second occurrence overwrites the first.

CFML supports Oracle 8 and 9 Reference Cursor type, which passes a parameter by reference. Parameters that are passed this way can be allocated and deallocated from memory within the execution of one application. To use reference cursors in packages or stored procedures, use the cfprocresult tag. This causes the ColdFusion JDBC database driver to put Oracle reference cursors into a result set. (You cannot use this method with Oracle’s ThinClient JDBC drivers.)

Example

```
<!--- This example executes a Sybase stored procedure that returns three result sets, two of which we want. The stored procedure returns status code and one output parameter, which we display. We use named notation for parameters. --->
<cfprocresult procedure = "foo_proc"
```

```
dataSource = "MY_SYBASE_TEST" username = "sa"
password = "" dbServer = "scup" dbName = "pubs2"
returnCode = "Yes" debug = "Yes">
<!--- cfprocresult tags ---->
<cfprocresult name = RS1>
<cfprocresult name = RS3 resultSet = 3>
<!--- cfprocparam tags ---->
<cfprocparam type = "IN"
  CFSQLType = CF_SQL_INTEGER
  value = "1" dbVarName = @param1>
<cfprocparam type = "OUT" CFSQLType = CF_SQL_DATE
  variable = FOO dbVarName = @param2>
<!--- Close the cfstoredproc tag. ---->
</cfstoredproc>
<cfoutput>
The output param value: '#foo#'</cfoutput>
<h3>The Results Information</h3>
<cfoutput query = RS1>#name#, #DATE_COL#<br>
</cfoutput>
<p>
<cfoutput query = RS3>#col1#, #col2#, #col3#<br>
</cfoutput>
<p>
The return code for the stored procedure is: '#cfstoredproc.statusCode#'</p>
...
**cfproperty**

**Description**
Defines properties of a ColdFusion component (CFC). Used to create complex data types for web services. The attributes of this tag are exposed as component metadata and are subject to inheritance rules.

**Category**
Extensibility tags

**Syntax**
```
<cfproperty
    name="name"
    type="type"
    required="boolean"
    default="default value"
    displayname="descriptive name"
    hint="extended description"
>
```

**See also**
- cfargument, cfcomponent, cffunction, cfinvoke, cfinvokeargument, cfobject, cfreturn;

**History**
ColdFusion MX: Added this tag.
Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Req/Opt</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Required</td>
<td></td>
<td>A string; a property name. Must be a static value.</td>
</tr>
<tr>
<td>type</td>
<td>Optional</td>
<td>any</td>
<td>A string; identifies the property data type:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• any</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• array</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• binary</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• boolean</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• date</td>
</tr>
<tr>
<td></td>
<td></td>
<td>guid</td>
<td>the argument must be a UUID or GUID of the form xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx where each x is a character representing a hexadecimal number (0-9A-F).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>numeric</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>query</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>string</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>struct</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>uuid</td>
<td>The argument must be a ColdFusion UUID of the form xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx where each x is a character representing a hexadecimal number (0-9A-F).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>variableName</td>
<td>a string formatted according to ColdFusion variable naming conventions.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>a component name</td>
<td>if the type attribute value is not one of the preceding items, ColdFusion treats it as the name of a ColdFusion component. When The function executes, it generates an error if the argument that is passed in is not a CFC with the specified name.</td>
</tr>
<tr>
<td>required</td>
<td>Optional</td>
<td>no</td>
<td>Whether the parameter is required:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• yes</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• no</td>
</tr>
<tr>
<td>default</td>
<td>Optional</td>
<td></td>
<td>If no property value is set when the component is used for a web service, specifies a default value.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>If this attribute is present, the required attribute must be set to <em>no</em> or not specified.</td>
</tr>
<tr>
<td>displayname</td>
<td>Optional</td>
<td></td>
<td>A value to be displayed when using introspection to show information about the CFC. The value appears in parentheses following the property name.</td>
</tr>
<tr>
<td>hint</td>
<td>Optional</td>
<td></td>
<td>Text to be displayed when using introspection to show information about the CFC. This attribute can be useful for describing the purpose of the parameter.</td>
</tr>
</tbody>
</table>

Usage

You must position cfproperty tags at the beginning of a component, above executable code and function definitions.

If a component is not used as a web service, The cfproperty only provides metadata information when the component is viewed using introspection; for example, by opening the CFC file directly in the browser. It does not define variables or set values that you can then use in your component.
For web services that you create in ColdFusion, the `cfproperty` tag defines complex variables used by the web service.

**Example**

The following code defines a component in the file `address.cfc` that contains properties that represent a street address:

```cfcomponent
  <cfproperty name="Number" type="numeric">
  <cfproperty name="Street" type="string">
  <cfproperty name="City" type="string">
  <cfproperty name="State" type="string">
  <cfproperty name="Country" type="string">
</cfcomponent>

This component represents a complex data type that can be used in a component that is exported as a web service, such as the following:

```cfcomponent
  <cffunction name="echoAddress" returnType="address" access="remote">
    <cfargument name="input" type="address">
    <cfreturn arguments.input>
  </cffunction>
</cfcomponent>
**cfquery**

**Description**

Passes queries or SQL statements to a data source.


**Category**

Database manipulation tags

**Syntax**

```xml
<cfquery
    name = "query_name"
    dataSource = "ds_name"
    dbtype = "query"
    username = "username"
    password = "password"
    maxRows = "number"
    blockFactor = "blocksize"
    timeout = "seconds"
    cachedAfter = "date"
    cachedWithin = "timespan"

    Either of the following:
    debug = "yes" or "no"
    or:
    debug
    >
    result = "result_name"

</cfquery>
```

**See also**

`cfinsert, cfprocparam, cfprocresult, cfqueryparam, cfstoredproc, cftransaction, cfupdate`; “Optimizing database use” in Chapter 13, “Designing and Optimizing a ColdFusion Application,” and Chapters 19–22 in *ColdFusion MX Developer’s Guide*

**History**

ColdFusion MX 7:

- Added the `result` attribute for specifying an alternate name for the structure that holds the result variables.
- Added result variables for the SQL statement executed (`sql`), the number of records returned (`recordcount`), whether the query was cached (`cached`), an array of `cfqueryparam` values (`sqlparameters`), and the list of columns in the returned query (`columnlist`).
ColdFusion MX:

- Changed Query of Queries behavior: it now supports a larger subset of standard SQL.
- Changed dot notation support: ColdFusion now supports dot notation within a record set name. ColdFusion interprets such a name as a structure.
- Deprecated the connectString, dbName, dbServer, provider, providerDSN, and sql attributes, and all values of the dbtype attribute except query. They do not work, and might cause an error, in releases later than ColdFusion 5.
- No longer supports native drivers. It now uses JDBC (and ODBC-JDBC bridge) for database connectivity.

Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Req/Opt</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Required</td>
<td>Name of query. Used in page to reference query record set. Must begin with a letter. Can include letters, numbers, and underscores.</td>
<td></td>
</tr>
<tr>
<td>dataSource</td>
<td>Required unless dbtype=query.</td>
<td>Name of data source from which query gets data. You must specify either dbtype or dataSource.</td>
<td></td>
</tr>
<tr>
<td>dbtype</td>
<td>Optional</td>
<td>Use this value to specify the results of a query as input. You must specify either dbtype or dataSource.</td>
<td></td>
</tr>
<tr>
<td>username</td>
<td>Optional</td>
<td>Overrides username in data source setup.</td>
<td></td>
</tr>
<tr>
<td>password</td>
<td>Optional</td>
<td>Overrides password in data source setup.</td>
<td></td>
</tr>
<tr>
<td>maxRows</td>
<td>Optional</td>
<td>Maximum number of rows to return in record set.</td>
<td></td>
</tr>
<tr>
<td>blockSize</td>
<td>Optional</td>
<td>Maximum rows to get at a time from server. Range: 1 - 100. Might not be supported by some database systems.</td>
<td></td>
</tr>
<tr>
<td>timeout</td>
<td>Optional</td>
<td>Maximum number of seconds that each action of a query is permitted to execute before returning an error. The cumulative time may exceed this value. For JDBC statements, ColdFusion sets this attribute. For other drivers, check driver documentation.</td>
<td></td>
</tr>
<tr>
<td>cachedAfter</td>
<td>Optional</td>
<td>Date value (for example, April 16, 1999, 4-16-99). If date of original query is after this date, ColdFusion uses cached query data. To use cached data, current query must use same SQL statement, data source, query name, user name, password. A date/time object is in the range 100 AD-9999 AD. When specifying a date value as a string, you must enclose it in quotation marks.</td>
<td></td>
</tr>
</tbody>
</table>
Usage

Use this tag to execute a SQL statement against a ColdFusion data source. Although you can use the `cfquery` tag to execute any SQL Data Definition Language (DDL) or Data Manipulation Language (DML) statement, you typically use it to execute a SQL SELECT statement.

**Note:** To call a stored procedure, use the `cfstoredproc` tag.

This tag creates a query object, providing this information in query variables:

<table>
<thead>
<tr>
<th>Variable name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>query_name.currentRow</code></td>
<td>Current row of query that <code>cfoutput</code> is processing.</td>
</tr>
<tr>
<td><code>query_name.columnList</code></td>
<td>Comma-delimited list of the query columns.</td>
</tr>
<tr>
<td><code>query_name.RecordCount</code></td>
<td>Number of records (rows) returned from the query.</td>
</tr>
</tbody>
</table>

The `cfquery` tag also returns the following result variables in a structure. You can access these variables with a prefix of the name you specified in the `result` attribute. For example, if you assign the name `myResult` to the `result` attribute, you would retrieve the name of the SQL statement that was executed by accessing `#myResult.sql#`. The `result` attribute provides a way for functions or CFCs that are called from multiple pages, possibly at the same time, to avoid overwriting results of one call with another.

<table>
<thead>
<tr>
<th>Variable name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>result_name.sql</code></td>
<td>The SQL statement that was executed.</td>
</tr>
<tr>
<td><code>result_name.recordcount</code></td>
<td>Current row of query that <code>cfoutput</code> is processing.</td>
</tr>
<tr>
<td><code>result_name.cached</code></td>
<td>True if the query was cached; False otherwise.</td>
</tr>
<tr>
<td><code>result_name.sqlparameters</code></td>
<td>An ordered Array of <code>cfqueryparam</code> values.</td>
</tr>
</tbody>
</table>

Attribute | Req/Opt | Default | Description                                                                 |
-----------|---------|---------|-----------------------------------------------------------------------------|
`cachedWithin` | Optional | Timespan, using the `CreateTimeSpan` function. If original query date falls within the time span, cached query data is used. `CreateTimeSpan` defines a period from the present, back. Takes effect only if query caching is enabled in the Administrator. To use cached data, the current query must use the same SQL statement, data source, query name, user name, and password. |
`debug` | Optional; value and equals sign may be omitted | • Yes, or if omitted: if debugging is enabled, but the Administrator Database Activity option is not enabled, displays SQL submitted to datasource and number of records returned by query. • No: if the Administrator Database Activity option is enabled, suppresses display. |
`result` | Optional | Specifies a name for the structure in which `cfquery` returns the result variables. For more information, see the Usage section. |

**cachedWithin**

Optional  Timespan, using the `CreateTimeSpan` function. If original query date falls within the time span, cached query data is used. `CreateTimeSpan` defines a period from the present, back. Takes effect only if query caching is enabled in the Administrator. To use cached data, the current query must use the same SQL statement, data source, query name, user name, and password.
You can cache query results and execute stored procedures. For information about this and about displaying `cfquery` output, see *ColdFusion MX Developer’s Guide*.

Because the `timeout` attribute only affects the maximum time for each suboperation of a query, the cumulative time may exceed its value. To set a timeout for a page that might get a very large result set, set the Administrator > Server Settings > Timeout Requests option to an appropriate value or use the RequestTimeout attribute of the `cfsetting` tag (for example, `<cfsetting requestTimeout="300">`).

The Caching page of the ColdFusion MX Administrator specifies the maximum number of cached queries. Setting this value to 0 disables query caching.

You cannot use ColdFusion reserved words as query names.

You cannot use SQL reserved words as variable or column names in a Query of Queries, unless they are escaped. The escape character is the bracket `[]`; for example:

```
SELECT [count] FROM MYTABLE.
```

For a list of reserved keywords in ColdFusion MX, see “Escaping reserved keywords” in Chapter 22, “Using Query of Queries” in *ColdFusion MX Developer’s Guide*.

**Example**

```xml
<!--- This example shows the use of CreateTimeSpan with CFQUERY ------->
<!--- to cache a record set. Define startrow and maxrows to ----->
<!--- facilitate 'next N' style browsing. ---->  
<cfparam name="MaxRows" default="10">  
<cfparam name="StartRow" default="1">  
<!--------------------------------------------------------------------  
Query database for information if cached database information has  
not been updated in the last six hours; otherwise, use cached data.  
--------------------------------------------------------------------->
<cfquery  
name="GetParks" datasource="cfdocexamples"  
cachedwithin="#CreateTimeSpan(0, 6, 0, 0)#">  
SELECT PARKNAME, REGION, STATE  
FROM Parks  
ORDER BY ParkName, State  
</cfquery>  
<!--- Build HTML table to display query. ------------------------->  
<table cellpadding="1" cellspacing="1">  
<tr>  
<td bgcolor="#f0f0f0">  
</td>  
<td bgcolor="#f0f0f0">  
<b><i>Park Name</i></b>  
</td>  
<td bgcolor="#f0f0f0">  
</td>  
</tr>  
</table>  
```

<table>
<thead>
<tr>
<th>Variable name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>result_name.columnList</code></td>
<td>Comma-delimited list of the query columns.</td>
</tr>
<tr>
<td><code>result_name.ExecutionTime</code></td>
<td>Cumulative time required to process the query.</td>
</tr>
</tbody>
</table>
```
<!--- Output the query and define the startrow and maxrows parameters. Use the query variable CurrentCount to keep track of the row you are displaying. ------>
<cfoutput query="GetParks" startrow="#StartRow#" maxrows="#MaxRows#">
<tr>
<td valign="top" bgcolor="ffffed">
<b>#GetParks.CurrentRow#</b>
</td>
<td valign="top">
<font size="-1">#ParkName#</font>
</td>
<td valign="top">
<font size="-1">#Region#</font>
</td>
<td valign="top">
<font size="-1">#State#</font>
</tr>
</cfoutput>
<!--- If the total number of records is less than or equal to the total number of rows, then offer a link to the same page, with the startrow value incremented by maxrows (in the case of this example, incremented by 10). --------->
<tr colspan="4">
<cfif (StartRow + MaxRows) LTE GetParks.RecordCount>
<cfoutput><a href="#CGI.SCRIPT_NAME#?startrow=#Evaluate(StartRow + MaxRows)#">See next #MaxRows# rows</a></cfoutput>
</cfif>
</tr>
</table>
```
cfqueryparam

Description
Verifies the data type of a query parameter and, for DBMSs that support bind variables, enables ColdFusion to use bind variables in the SQL statement. Bind variable usage enhances performance when executing a cfquery statement multiple times.

This tag is nested within a cfquery tag, embedded in a query SQL statement. If you specify optional parameters, this tag performs data validation.


Category
Database manipulation tags

Syntax
<cfquery
   name = "query_name"
   dataSource = "ds_name"
   ...other attributes...
   SQL STATEMENT column_name =
   <cfqueryparam value = "parameter value"
      CFSQLType = "parameter type"
      maxLength = "maximum parameter length"
      scale = "number of decimal places"
      null = "yes" or "no"
      list = "yes" or "no"
      separator = "separator character">
   AND/OR ...additional criteria of the WHERE clause...
</cfquery>

See also
cfinsert, cfprocparam, cfprocresult, cfquery, cfstoredproc, cftransaction, cfupdate;
## Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Req/Opt</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>Required</td>
<td>Value</td>
<td>Value that ColdFusion passes to the right of the comparison operator in a <em>where</em> clause. If <code>CFSQLType</code> is a date or time option, ensure that the date value uses your DBMS-specific date format. Use the <code>CreateODBCDateTime</code> or <code>DateFormat</code> and <code>TimeFormat</code> functions to format the date value.</td>
</tr>
<tr>
<td><code>CFSQLType</code></td>
<td>Optional</td>
<td><code>CF_SQL_CHAR</code></td>
<td>SQL type that parameter (any type) is bound to:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• <code>CF_SQL_BIGINT</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• <code>CF_SQL_BIT</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• <code>CF_SQL_CHAR</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• <code>CF_SQL_BLOB</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• <code>CF_SQL_CLOB</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• <code>CF_SQL_DATE</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• <code>CF_SQL_DECIMAL</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• <code>CF_SQL_DOUBLE</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• <code>CF_SQL_FLOAT</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• <code>CF_SQL_IDSTAMP</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• <code>CF_SQL_INTEGER</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• <code>CF_SQL_LONGVARCHAR</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• <code>CF_SQL_MONEY</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• <code>CF_SQL_MONEY4</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• <code>CF_SQL_NUMERIC</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• <code>CF_SQL_REAL</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• <code>CF_SQL_REAL</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• <code>CF_SQL_REFCURSOR</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• <code>CF_SQL_SMALLINT</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• <code>CF_SQL_TIME</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• <code>CF_SQL_TIMESTAMP</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• <code>CF_SQL_TINYINT</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• <code>CF_SQL_VARCHAR</code></td>
</tr>
<tr>
<td>maxLength</td>
<td>Optional</td>
<td>Length of string in <code>value</code> attribute</td>
<td>Maximum length of parameter. Ensures that the length check is done by ColdFusion before the string is sent to the DBMS, thereby helping to prevent the submission of malicious strings.</td>
</tr>
<tr>
<td>scale</td>
<td>Optional</td>
<td>0</td>
<td>Number of decimal places in parameter. Applies to <code>CF_SQL_NUMERIC</code> and <code>CF_SQL_DECIMAL</code>.</td>
</tr>
<tr>
<td>null</td>
<td>Optional</td>
<td>No</td>
<td>Whether parameter is passed as a null value:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Yes: tag ignores the <code>value</code> attribute.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• No</td>
</tr>
<tr>
<td>list</td>
<td>Optional</td>
<td>No</td>
<td>Whether the <code>value</code> attribute value is a delimited list:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Yes</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• No</td>
</tr>
<tr>
<td>separator</td>
<td>Required, if you specify a list in <code>value</code> attribute</td>
<td>, (comma)</td>
<td>Character that separates values in list, in <code>value</code> attribute.</td>
</tr>
</tbody>
</table>
Usage

Use the `cfqueryparam` tag in any SQL statement (for example, SELECT, INSERT, UPDATE, and DELETE) that uses ColdFusion variables.

You cannot use the `cfquery cachedAfter` or `cachedWithin` attributes with `cfqueryparam`.

For maximum validation of string data, specify the `maxlength` attribute.

This tag does the following:

• Allows the use of SQL bind parameters, which improves performance.
• Ensures that variable data matches the specified SQL type.
• Allows long text fields to be updated from a SQL statement.
• Escapes string variables in single-quotiation marks.

To benefit from the enhanced performance of bind variables, you must use `cfqueryparam` for all ColdFusion variables, and your DBMS must support bind variables. If a DBMS does not support bind parameters, ColdFusion validates and substitutes the validated parameter value back into the string. If validation fails, it returns an error message.

The validation rules are as follows:

• For these types, a data value can be converted to a numeric value: `CF_SQL_SMALLINT`, `CF_SQL_INTEGER`, `CF_SQL_REAL`, `CF_SQL_FLOAT`, `CF_SQL_DOUBLE`, `CF_SQL_TINYINT`, `CF_SQL_MONEY`, `CF_SQL_MONEY4`, `CF_SQL_DECIMAL`, `CF_SQL_NUMERIC`, and `CF_SQL_BIGINT`
• For these types, a data value can be converted to a date supported by the target data source: `CF_SQL_DATE`, `CF_SQL_TIME`, `CF_SQL_TIMESTAMP`
• For all other types, if the `maxLength` attribute is used, a data value cannot exceed the maximum length specified.

ColdFusion debug output shows the bind variables as question marks and lists the values beneath the query, in order of usage.

**Note:** To insert an empty string into a Microsoft Access table using the SequelLink ODBC Socket or SequelLink Access driver, the `CFSQLType` attribute must specify `CF_SQL_LONGVARCHAR`.

The following table shows the mapping of ColdFusion SQL data types with JDBC SQL types and those of the listed database management systems:

<table>
<thead>
<tr>
<th>ColdFusion</th>
<th>JDBC</th>
<th>DB2</th>
<th>Informix</th>
<th>Oracle</th>
<th>MSSQL</th>
</tr>
</thead>
<tbody>
<tr>
<td>CF_SQL_ARRAY</td>
<td>ARRAY</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CF_SQL_BIGINT</td>
<td>BIGINT</td>
<td>Bigint</td>
<td>int8, serial8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CF_SQL_BINARY</td>
<td>BINARY</td>
<td>Char for Bit Data</td>
<td>binary timestamp</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CF_SQL_BIT</td>
<td>BIT</td>
<td>boolean</td>
<td>bit</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CF_SQL_BLOB</td>
<td>BLOB</td>
<td>Blob</td>
<td>blob</td>
<td>blob, bfile</td>
<td></td>
</tr>
<tr>
<td>ColdFusion</td>
<td>JDBC</td>
<td>DB2</td>
<td>Informix</td>
<td>Oracle</td>
<td>MSSQL</td>
</tr>
<tr>
<td>--------------------</td>
<td>---------------</td>
<td>--------------</td>
<td>----------------</td>
<td>---------------</td>
<td>---------------</td>
</tr>
<tr>
<td>CF_SQL_CHAR</td>
<td>CHAR</td>
<td>Char</td>
<td>char, nchar</td>
<td>char, nchar</td>
<td>char, nchar, unique identifier</td>
</tr>
<tr>
<td>CF_SQL_CLOB</td>
<td>CLOB</td>
<td>Clob</td>
<td>clob</td>
<td>clob, nclob</td>
<td></td>
</tr>
<tr>
<td>CF_SQL_DATE</td>
<td>DATE</td>
<td>Date</td>
<td>date, datetime, year to day</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CF_SQL_DECIMAL</td>
<td>DECIMAL</td>
<td>Decimal</td>
<td>decimal, money</td>
<td>number</td>
<td>decimal, money, small money</td>
</tr>
<tr>
<td>CF_SQL_DISTINCT</td>
<td>DISTINCT</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CF_SQL_DOUBLE</td>
<td>DOUBLE</td>
<td>Double</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CF_SQL_FLOAT</td>
<td>FLOAT</td>
<td>Float</td>
<td>float</td>
<td>number</td>
<td>float</td>
</tr>
<tr>
<td>CF_SQL_IDSTAMP</td>
<td>CHAR</td>
<td>Char</td>
<td>char, nchar</td>
<td>char, nchar</td>
<td>char, nchar, unique identifier</td>
</tr>
<tr>
<td>CF_SQL_INTEGER</td>
<td>INTEGER</td>
<td>Integer</td>
<td>integer, serial</td>
<td></td>
<td>int</td>
</tr>
<tr>
<td>CF_SQL_LONGVARBINARY</td>
<td>LONGVARBINARY</td>
<td>Long Varchar</td>
<td>byte</td>
<td>long raw</td>
<td>image</td>
</tr>
<tr>
<td>CF_SQL_LONGVARCHAR</td>
<td>LONGVARCHAR</td>
<td>Long Varchar</td>
<td>text</td>
<td>long</td>
<td>text, ntext</td>
</tr>
<tr>
<td>CF_SQL_MONEY</td>
<td>DOUBLE</td>
<td>Double</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CF_SQL_MONEY4</td>
<td>DOUBLE</td>
<td>Double</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CF_SQL_NULL</td>
<td>NULL</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CF_SQL_NUMERIC</td>
<td>NUMERIC</td>
<td>Numeric</td>
<td>numeric</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CF_SQL_OTHER</td>
<td>OTHER</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CF_SQL_REAL</td>
<td>REAL</td>
<td>Real</td>
<td>smallfloat</td>
<td>real</td>
<td></td>
</tr>
<tr>
<td>CF_SQL_REFCURSOR</td>
<td>REF</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CF_SQL_SMALLINT</td>
<td>SMALLINT</td>
<td>Smallint</td>
<td>smallint</td>
<td>smallint</td>
<td></td>
</tr>
<tr>
<td>CF_SQL_STRUCT</td>
<td>STRUCT</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CF_SQL_TIME</td>
<td>TIME</td>
<td>Time</td>
<td>datetime hour to second</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Example

<!--- This example shows cfqueryparam with VALID input in Course_ID. --->
<h3>cfqueryparam Example</h3>
<cfset Course_ID = 12>
<cfquery name = "getFirst" dataSource = "cfdocexamples">
    SELECT *
    FROM courses
    WHERE Course_ID = <cfqueryPARAM value = "#Course_ID#" CFSQLType = "CF_SQL_INTEGER">
</cfquery>
<cfoutput query = "getFirst">
    <p>Course Number: #Course_ID#<br> Description: #descript#</p>
</cfoutput>

<!--- This example shows the use of CFQUERYPARAM when INVALID string data is in Course_ID. ---->
<p>This example throws an error because the value passed in the CFQUERYPARAM tag exceeds the MAXLENGTH attribute</p>
<cfset LastName="Peterson; DELETE employees WHERE LastName='Peterson'">
<!------- Note that for string input you must specify the MAXLENGTH attribute for validation. -------------------------------------------------->  
<cfquery name="getFirst" datasource="cfdocexamples">
    SELECT *
    FROM employees
    WHERE LastName=<cfqueryparam value="#LastName#" cfsqltype="CF_SQL_VARCHAR" maxlength="17">
</cfquery>
<cfoutput query="getFirst"> <p>
    Course Number: #FirstName# #LastName#
    Description: #Department#</p>
</cfoutput>
cfregistry

**Description**

This tag is deprecated for the UNIX platform.

Reads, writes, and deletes keys and values in the system registry. Provides persistent storage of client variables.

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

**Category**

*Other tags, Variable manipulation tags*

**Syntax**

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

- `cfregistry action = "getAll"` on page 357
- `cfregistry action = "get"` on page 359
- `cfregistry action = "set"` on page 360
- `cfregistry action = "delete"` on page 361

**See also**


**History**

ColdFusion MX:

- Deprecated this tag on the UNIX platform. It might not work, and might cause an error, in later releases.
- Changed how persistent data is stored: ColdFusion now stores most persistent data outside the system registry, in XML files.
cfregistry action = "getAll"

Description

Returns all registry keys and values defined in a branch. You can access the values as you would any record set.

Syntax

```xml
<cfregistry
    action = "getAll"
    branch = "branch"
    type = "data type"
    name = "query name"
    sort = "criteria">
</cfregistry>
```

See also

Chapter 15, “Using Persistent Data and Locking,” in *ColdFusion MX Developer's Guide*

Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Req/Opt</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>action</td>
<td>Required</td>
<td>Always getall.</td>
<td></td>
</tr>
<tr>
<td>branch</td>
<td>Required</td>
<td>Name of a registry branch.</td>
<td></td>
</tr>
<tr>
<td>type</td>
<td>Optional</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• string: returns string values.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• dWord: returns DWord values.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• key: returns keys.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• any: returns keys and values.</td>
<td></td>
</tr>
<tr>
<td>name</td>
<td>Required</td>
<td>Name of record set to contain returned keys and values.</td>
<td></td>
</tr>
<tr>
<td>sort</td>
<td>Optional</td>
<td>ASC</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Sorts query column data (case-insensitive). Sorts on Entry, Type, and Value columns as text. Specify a combination of columns from query output, in a comma-delimited list. For example:</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>sort = &quot;value desc, entry asc&quot;</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• asc: ascending (a to z) sort order.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• desc: descending (z to a) sort order.</td>
<td></td>
</tr>
</tbody>
</table>

Usage

This tag returns #entry#, #type#, and #value# in a record set that you can access through tags such as `cfoutput`. To fully qualify these variables, use the record set name, as specified in the `name` attribute.

If #type# is a key, #value# is an empty string.

If you specify `type= "any"`, `getAll` also returns binary registry values. For binary values, the #type# variable contains UNSUPPORTED and #value# is blank.

Example

```xml
<!--- This example uses cfregistry with the getAll action. --->
<cfregistry action = "getAll"
    branch = "HKEY_LOCAL_MACHINE\Software\Microsoft\Java VM"
    type = "Any" name = "RegQuery">
```

---

**cfregistry** 357
<p><h1>cfregistry action = "getAll"</h1></p>
<cfregistry action = "getAll">
<cfquery name = "RegQuery">
<cfcol header = "Entry" width = "35" text = "#RegQuery.Entry#">
<cfcol header = "Type" width = "10" text = "#RegQuery.Type#">
<cfcol header = "Value" width = "35" text = "#RegQuery.Value#">
</cfquery>
</cfregistry>
**cfregistry action = "get"**

**Description**

Accesses a registry value and stores it in a ColdFusion variable.

**Syntax**

```cfc
<cfregistry
action = "get"
branch = "branch"
entry = "key or value"
variable = "variable"
type = "data type">
```

**See also**

Chapter 15, “Using Persistent Data and Locking,” in *ColdFusion MX Developer’s Guide*

**Attributes**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Req/Opt</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>action</td>
<td>Required</td>
<td>Always get.</td>
<td></td>
</tr>
<tr>
<td>branch</td>
<td>Required</td>
<td>Name of a registry branch.</td>
<td></td>
</tr>
<tr>
<td>entry</td>
<td>Required</td>
<td>Registry value to access.</td>
<td></td>
</tr>
<tr>
<td>variable</td>
<td>Required</td>
<td>Variable into which to put value.</td>
<td></td>
</tr>
<tr>
<td>type</td>
<td>Optional</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• string: returns string value.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• dWord: returns DWord value.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• key: returns key’s default value.</td>
<td></td>
</tr>
</tbody>
</table>

**Usage**

If the value does not exist, the `cfregistry` tag does not create an entry.

**Example**

```cfc
<!--- This example uses cfregistry with the get action. --->
<cfregistry action = "get"
branch = "HKEY_LOCAL_MACHINE\Software\Microsoft\Java VM"
entry = "ClassPath" type = "String" variable = "RegValue">
<h1>cfregistry action = "get"</h1>
<cfoutput>
<p>Java ClassPath value is #RegValue#</p>
</cfoutput>
```
**cfregistry action = "set"**

**Description**

Adds a registry key, adds a value, or updates a value.

**Syntax**

```xml
<cfregistry
    action = "set"
    branch = "branch"
    entry = "key or value"
    type = "value type"
    value = "data">
```

**See also**

Chapter 15, “Using Persistent Data and Locking,” in ColdFusion MX Developer’s Guide

**Attributes**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Req/Opt</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>action</td>
<td>Required</td>
<td>Always set.</td>
<td></td>
</tr>
<tr>
<td>branch</td>
<td>Required</td>
<td>Name of a registry branch.</td>
<td></td>
</tr>
<tr>
<td>entry</td>
<td>Required</td>
<td>Key or value to set.</td>
<td></td>
</tr>
<tr>
<td>type</td>
<td>Optional</td>
<td>• string: sets a string value (default).&lt;br&gt;• dWord: sets a DWord value.&lt;br&gt;• key: creates a key.</td>
<td></td>
</tr>
<tr>
<td>value</td>
<td>Optional</td>
<td>Value data to set. If you omit this attribute, the <code>cfregistry</code> tag creates default value, as follows:&lt;br&gt;• string: creates an empty string: &quot;&quot;.&lt;br&gt;• dWord: creates a value of 0 (zero).</td>
<td></td>
</tr>
</tbody>
</table>

**Usage**

If it does not exist, the `cfregistry` tag creates the key or value.

**Example**

```xml
<!--- This example uses the cfregistry set action to modify registry value data. --->
<!--- Normally you pass in a filename instead of setting one here. --->
<cfset FileName = "dummy.cfm">
<cfregistry action = "set"
    branch = "HKEY_LOCAL_MACHINE\Software\cflangref"
    entry = "LastCFM01" type = "String" value = "#FileName#">
<h1>cfregistry action = "set"</h1>
```
**cfregistry action = "delete"**

**Description**

Deletes a registry key or value.

**Syntax**

```plaintext
<cfregistry
 action = "delete"
 branch = "branch"
 entry = "keyorvalue">
```

**See also**

Chapter 15, “Using Persistent Data and Locking,” in *ColdFusion MX Developer’s Guide*

**Attributes**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Req/Opt</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>action</td>
<td>Required</td>
<td>Always delete.</td>
<td></td>
</tr>
<tr>
<td>branch</td>
<td>Required</td>
<td></td>
<td>• For key deletion: name of registry key to delete. Do not specify the entry attribute.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• For value deletion: name of registry branch that contains value to delete. You must specify the entry attribute.</td>
</tr>
<tr>
<td>entry</td>
<td>Required for value deletion</td>
<td>Value to delete.</td>
<td></td>
</tr>
</tbody>
</table>

**Usage**

If you delete a key, the `cfregistry` tag also deletes values and subkeys defined beneath it.

**Example**

```plaintext
<cfregistry action = "delete"
 branch = "HKEY_LOCAL_MACHINE\Software\cflangref\tempkey"
 entry = "LastCFM01">
<h1>cfregistry action = "delete"</h1>
```
cfreport

Description

Used to do either of the following:

- Execute a report definition created with the ColdFusion Report Builder, displaying it in PDF, FlashPaper, or Excel format. You can optionally save this report to a file.
- Run a predefined Crystal Reports report. Applies only to Windows systems. Uses the CFCRYSTAL.exe file to generate reports. Sets parameters in the Crystal Reports engine according to its attribute values.

Category

Data output tags

Syntax

<!--- Syntax 1 - Use this syntax with the ColdFusion Report Builder. --->
<cfreport
    template = "report definition filename"
    format = "PDF or FlashPaper" or "excel"
    name = "cf variable"
    filename = "output filename"
    query = "query variable"
    overwrite = "yes" or "no"
    encryption = "128-bit" or "40-bit" or "none"
    ownerpassword = "password"
    userpassword = "password"
    permissions = "permission list"
>
    cfreportparam tags [optional]
</cfreport>

OR

<!--- Syntax 2 - Use this syntax with Crystal Reports. --->
<cfreport
    report = "report_path"
    dataSource = "ds_name"
    type = "type"
    timeout = "number of seconds"
    orderBy = "result_order"
    username = "username"
    password = "password"
    formula = "formula"
>
</cfreport>

See also

cfcollection, cfdocument, cfdocumentitem, cfdocumentsection, cfexecute, cfindex, cfobject, cfreportparam, cfsearch, cfwddx; Chapter 32, “Creating Reports for Printing,” in ColdFusion MX Developer’s Guide; Report Builder online Help
History

ColdFusion MX 7: Added support for the ColdFusion Report Builder.

ColdFusion MX: Changed data source connection behavior: Crystal Reports now establishes an independent connection to the data source. The connection is not subject to any ColdFusion data source-specific restrictions. For example, the Crystal Reports server can access a data source, regardless of whether it is disabled in the ColdFusion MX Administrator.

Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Applies to</th>
<th>Req/Opt</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>template</td>
<td>Report Builder</td>
<td>Req</td>
<td></td>
<td>Specifies the path to the report definition file, relative to the web root.</td>
</tr>
<tr>
<td>format</td>
<td>Report Builder</td>
<td>Req</td>
<td></td>
<td>Specify one of the following: PDF, FlashPaper, Excel</td>
</tr>
<tr>
<td>name</td>
<td>Report Builder</td>
<td>Opt</td>
<td></td>
<td>The name of the ColdFusion variable that will hold the report output. You cannot specify both name and filename.</td>
</tr>
<tr>
<td>filename</td>
<td>Report Builder</td>
<td>Opt</td>
<td></td>
<td>The filename to contain the report. You cannot specify both name and filename.</td>
</tr>
<tr>
<td>query</td>
<td>Report Builder</td>
<td>Opt</td>
<td></td>
<td>The name of the query that contains input data for the report. If you omit this parameter, the report definition obtains data from the internal SQL or from cfreportparam items.</td>
</tr>
<tr>
<td>overwrite</td>
<td>Report Builder</td>
<td>Opt</td>
<td>No</td>
<td>Specifies whether to overwrite files that have the same name as that specified in the filename attribute. Specify Yes or No.</td>
</tr>
<tr>
<td>encryption</td>
<td>Report Builder</td>
<td>Opt</td>
<td>None</td>
<td>(format=&quot;PDF&quot; only) Specifies whether the output is encrypted. Specify one of the following: 128-bit, 40-bit, none</td>
</tr>
<tr>
<td>ownerpassword</td>
<td>Report Builder</td>
<td>Opt</td>
<td></td>
<td>(format=&quot;PDF&quot; only) Specifies an owner password.</td>
</tr>
<tr>
<td>userpassword</td>
<td>Report Builder</td>
<td>Opt</td>
<td></td>
<td>(format=&quot;PDF&quot; only) Specifies a user password.</td>
</tr>
</tbody>
</table>
**Usage**

Use this tag to generate a report using a report definition created in either ColdFusion Report Builder or in Crystal Reports. (For more information on using the ColdFusion Report Builder, display the online help by opening the Report Builder and pressing F1.)
**Note:** The Excel report output format type provides limited support for the formatting options available in ColdFusion MX 7 Reporting. Images and charts are not supported and numeric data containing formatting (commas, percents, currency, and so on) appear as plain text in Excel. The Excel output format supports simple reports only and Macromedia recommends that you give careful design and layout consideration to reports designed for Excel output.

This tag requires an end tag.

**Example**

<!--- Example 1: This example shows the use of cfreport for the ColdFusion Report Builder. --->

```coldfusion
<cfquery name="northwindemployees" datasource="localnorthwind">
    SELECT EmployeeID, LastName, FirstName, Title, City, Region, Country
    FROM Employees
    ORDER BY Country, City
</cfquery>

<CFREPORT format="PDF" template="FifthReport.cfr" query="#northwindemployees#" />
```

---

<!--- Example 2: This view-only example shows the use of cfreport for Crystal Reports. --->

```coldfusion
<h3>cfreport Tag</h3>
<p>cfreport lets reports from the Crystal Reports Professional report writer display through a ColdFusion interface. To run, the tag requires the name of the report. cfreport can also pass information to the report file displayed, to change the output conditions.</p>
<p>This example would run a report called "monthlysales.rpt" and pass it an optional filter condition to show only the information for a subset of the report.</p>

<cfreport report="/reports/monthlysales.rpt">
    [Departments.Department] = 'International'
</cfreport>

<p>Substitute your report files and filters for this code. cfreport can put Crystal Reports into web pages.</p>
**cfreportparam**

**Description**
Passes input parameters to a ColdFusion Report Builder report definition. Allowed inside `cfreport` tag bodies only.

**Category**
Data output tags

**Syntax**
```
<cfreportparam
  name = "data name"
  value = "data value">
```

**See also**
`cfreport`; Chapter 32, “Creating Reports for Printing,” in *ColdFusion MX Developer’s Guide*; Report Builder online Help

**History**
ColdFusion MX 7: Added this tag.

**Attributes**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Req/Opt</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Required</td>
<td>Variable name for data that is passed. The ColdFusion Report Builder report definition must include an input parameter that matches this name.</td>
<td></td>
</tr>
<tr>
<td>value</td>
<td>Required</td>
<td>Value of the data that is sent.</td>
<td></td>
</tr>
</tbody>
</table>

**Usage**
Specifies an input parameter to a ColdFusion Report Builder report definition. The ColdFusion Report Builder report definition must include an input parameter that matches the `name` attribute.

You can use `cfreportparam` to pass data to a report definition, as an alternative to the `cfreport` query attribute.

**Example**
```
<cfquery name="coursedept" datasource="cfdocexamples">
  SELECT Departments.Dept_ID as dDept_ID, Departments.Dept_Name, CourseList.Course_ID, CourseList.Dept_ID as cDept_ID, CourseList.CorNumber, CourseList.CorName, CourseList.CorLevel FROM Departments, CourseList WHERE Departments.Dept_ID = CourseList.Dept_ID ORDER BY CourseList.Dept_ID
</cfquery>
```
<cfreport format="PDF" template="FourthReport.cfr" query="#coursedept#" overwrite="yes">
  <cfreportparam NAME="ReportTime" VALUE="#DateFormat(Now())#, #TimeFormat(Now())#" />
</cfreport>
cfrerthrow

Description
Rethrows the currently active exception. Preserves the exception’s cfcatch.type and cfcatch.tagContext variable values.

Category
Exception handling tags, Extensibility tags

Syntax
<cfrethrow>

See also
cferror, cfthrow, cftry; “Handling runtime exceptions with ColdFusion tags” in Chapter 14, “Handling Errors,” in ColdFusion MX Developer’s Guide

Usage
Use this tag within a cfcatch block. This tag is useful in error handling code, if the error handler cannot handle an error that it catches. For example, if cfcatch type = "any" gets a DATABASE exception, and the code is designed to handle only CFX exceptions, the handler raises the exceptions again, with details intact, so that a higher-level handler can process the error information. If you used the cfthrow tag, the type and details of the original exception would be lost.

Example
<h3>cfrethrow Example</h3>
<!--- Rethrow a DATABASE exception. --->
<cftry>
  <cftry>
    <cfquery name = "GetMessages" dataSource = "cfdocexamples">
      SELECT  *
      FROM   Messages
    </cfquery>
    <cfcatch type = "DATABASE">
      <!--- If database signalled a 50555 error, ignore; otherwise, rethrow exception. --->
      <cfif cfcatch.sqlstate neq 50555>
        <cfrethrow>
      </cfif>
    </cfcatch>
  </cftry>
  <cfcatch>
    <h3>Sorry, this request can't be completed</h3>
    <h4>Catch variables</h4>
    <cfoutput>
      <cfloop collection = #cfcatch# item = "c">
        <br>
        <cfif IsSimpleValue(cfcatch[c])> #c# = #cfcatch[c]#
      </cfif>
    </cfloop>
  </cfcatch>
</cftry>
**cfreturn**

**Description**

Returns result values from a component method. Contains an expression returned as result of the function.

**Return value**

An expression; the result of the function from which this tag is called.

**Category**

Extensibility tags

**Syntax**

```xml
<cfreturn expr>
```

**See also**

`cfargument`, `cfcomponent`, `cffunction`, `cfinvoke`, `cfinvokeargument`, `cfobject`, `cfproperty`; “Creating reports with the ColdFusion MX 7 reporting” in Chapter 32, “Creating Reports for Printing,” in *ColdFusion MX Developer's Guide*

**History**

ColdFusion MX: Added this tag.

**Attributes**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Req/Opt</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>expr</td>
<td>Required</td>
<td></td>
<td>Function result; value of any type.</td>
</tr>
</tbody>
</table>

**Usage**

This tag is equivalent to a `return` statement within a `cfscript` tag. It accepts one return variable argument. To return more than one value, populate a structure with name-value-pairs, and return the structure with this tag.

To access the result value from this tag, you use the variable scope that is the value of the `cfinvoke` tag `returnVariable` attribute.

You can code a maximum of one `cfreturn` tag within a function.

For example code, see Chapter 10, “Building and Using ColdFusion Components” in *ColdFusion MX Developer's Guide*.

**Example**

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

Description
Saves the generated content of the cfsavecontent tag, including the results of evaluating expressions and executing custom tags, in the specified variable.

Category
Variable manipulation tags

Syntax
<cfcontent
variable = "variable name">
the content
</cfcontent>

See also
“Caching parts of ColdFusion pages” in Chapter 13, “Designing and Optimizing a ColdFusion Application,” in ColdFusion MX Developer's Guide

Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Req/Opt</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>variable</td>
<td>Required</td>
<td></td>
<td>Name of the variable in which to save the generated content of the tag.</td>
</tr>
</tbody>
</table>

Usage
This tag requires an end tag.

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

Example
The following example uses a custom tag to generate a report and saves the report in the variable CONTENT. It replaces all instances of the word "report" with the phrase "MyCompany Quarterly Report" and outputs the result.

```coldfusion
<cfcontent variable="content">
  <CF_OutputBigReport>
  </cfcontent>
<cfoutput>
  #replace(content, "report", "MyCompany Quarterly Report", "all")#
</cfoutput>
```
cfschedule

Description
Provides a programmatic interface to the ColdFusion scheduling engine. Can run a CFML page at scheduled intervals, with the option to write the page output to a static HTML page. This feature enables you to schedule pages that publish data, such as reports, without waiting while a database transaction is performed to populate the page.

Category
Variable manipulation tags

Syntax
```xml
<cfschedule
    action = "update"
    task = "taskname"
    operation = "HTTPRequest"
    file = "filename"
    path = "path_to_file"
    startDate = "date"
    startTime = "time"
    url = "URL"
    port = "port_number"
    publish = "yes" or "no"
    endDate = "date"
    endTime = "time"
    interval = "seconds"
    requestTimeOut = "seconds"
    username = "username"
    password = "password"
    proxyServer = "hostname"
    proxyPort = "port_number"
    proxyUser = "username"
    proxyPassword = "password"
    resolveURL = "yes" or "no"
    </cfschedule>

<cfschedule
    action = "delete"
    task = "TaskName">

<cfschedule
    action = "run"
    task = "TaskName">

See also
cffield, cfparam, cfregistry, cfsavecontent, cfset
History

ColdFusion MX 6.1: Changed the way intervals are calculated. The day length now reflects changes between standard and daylight saving times. The month length is now the calendar month length, not four weeks. The scheduler handles leap years correctly.

Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Req/Opt</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
</table>
| action         | Required|         | • delete: deletes the specified task.
|                |         |         | • update: updates an existing task or creates a new task, if one with the name specified by the task attribute does not exist.
| task           | Required|         | Name of the task. |
| operation      | Required if action = "update" |         | Operation that the scheduler performs. Must be HTTPRequest. |
| file           | Required if publish = "Yes" |         | Name of the file in which to store the published output of the scheduled task. |
| path           | Required if publish = "Yes" |         | Path to the directory in which to put the published file. |
| startDate      | Required if action = "update" |         | Date on which to first run the scheduled task. |
| startTime      | Required if action = "update" |         | Time at which to run the scheduled task starts. |
| url            | Required if action = "update" |         | URL of the page to execute. |
| port           | Optional | 80 | Port to use on the server that is specified by the url parameter. If resolveURL = "yes", retrieved document URLs that specify a port number are automatically resolved, to preserve links in the retrieved document. A port value in the url attribute overrides this value. |
| publish        | Optional | No | • Yes: saves the result to a file. 
|                |         |     | • No |
| endDate        | Optional |         | Date when scheduled task ends. |
| endTime        | Optional |         | Time when scheduled task ends (seconds). |
| interval       | Required if action = "update" |         | Interval at which task is scheduled: 
|                |         |         | • number of seconds 
|                |         |         | • once 
|                |         |         | • daily 
| start           |         |         | • weekly 
|                |         |         | • monthly |
| requestTimeOut | Optional |         | Can be used to extend the default timeout period. |
| username       | Optional |         | Username, if URL is protected. |
This tag and the ColdFusion MX Administrator Scheduled task page schedule ColdFusion tasks. Tasks that you add or change using this tag are visible in the Administrator. You can disable this tag in the Administrator Sandbox/Resource security page. This tag's success or failure status is written to the schedule.log file in the cf_root/logs directory (cf_webapp_root/WEB-INF/cfusion/logs in the multiserver and J2EE configurations).

When you create a task, you specify the URL of the ColdFusion page to execute, the date, time and frequency of execution, and whether to publish the task output to a HTML file. If the output is published, you specify the output file path and file.

If you schedule a job to run monthly on any date in the range 28-31, the scheduler does the following:

- If you schedule a monthly job to run on the last day of a month, the scheduled job will run on the last day of each month. For example, if you schedule a monthly job to start on January 31, it will run on January 31, February 28 or 29, March 31, April 30, and so on.
- If you schedule a monthly job to run on the 29th or 30th of the month, the job will run on the specified day of each month for 30 or 31-day months, and the last day of February. For example, if you schedule a monthly job to start on January 30, the job will run on January 30, February 28 or 29, March 30, April 30, and so on.

If you schedule a job to run once, the starting time is in the past, and the task has not yet run, it runs immediately. If you schedule a recurring job with a start time in the past, ColdFusion schedules the job to run on the next closest interval in the future.

The Scheduler configuration file, cf_root\lib\neo-cron.xml contains all scheduled events, as individual entries.

Example

```cfschedule Example```

```<!--- This read-only example schedules a task. To run the example, remove the comments around the code and change the startDate, startTime, url, file, and path attributes to appropriate values. --->

```cfschedule action = "update"
  task = "TaskName"
```
operation = "HTTPRequest"
url = "http://127.0.0.1/playpen/history.cfm"
startDate = "8/7/03"
startTime = "12:25 PM"
interval = "3600"
resolveURL = "Yes"
publish = "Yes"
file = "sample.html"
path = "c:\inetpub\wwwroot\playpen"
requestTimeOut = "600">
--->
**cfscript**

**Description**

Encloses a code block that contains `cfscript` statements.

**Category**

Application framework tags, Other tags

**Syntax**

```cfc
<cfscript>
  cfscript code here
</cfscript>
```

**See also**

`cfinvoke`, `cfmodule`, `createobject`; Chapter 6, “Extending ColdFusion Pages with CFML Scripting” in *ColdFusion MX Developer’s Guide*

**History**

ColdFusion MX:

• Changed how to invoke component methods: this tag can now invoke component methods, using the `createobject` function
• Changed use of reserved words: you cannot use ColdFusion reserved words within this tag
• Added the `try` and `catch` statements.

**Usage**

Performs processing in CFScript. This tag uses ColdFusion functions, expressions, and operators. You can read and write ColdFusion variables within this tag.

For a detailed description of the CFScript scripting language, including documentation of CFScript statements and the CFScript equivalents of CFML tags, see Chapter 6, “Extending ColdFusion Pages with CFML Scripting” in *ColdFusion MX Developer’s Guide*.

You can use this tag to enclose a series of assignment statements that would otherwise require `cfset` statements.

**Caution:** If you code a `cftry/cfcatch` block within this tag using an exception’s Java class name, you must provide the fully qualified class name.

You cannot use some ColdFusion reserved words in this tag. You cannot put a user-defined function whose name begins with any of these strings within this tag:

- `cf`
- `cf_`
- `_cf`
- `coldfusion`
- `coldfusion_`
- `_coldfusion`
You cannot use the `elseif` construct within a `cfscript` tag. You can use code such as the following:

```
else if ( condition )
{
...
}
```

### Exception handling with the `cfscript` tag

To handle exceptions with this tag, use `try` and `catch` statements, which are equivalent to the `cftry` and `cfcatch` tags. For each `try` statement, you must have a `catch` statement. In the `catch` block, the variable `exceptionVariable` contains the exception type. This variable is the equivalent of the `cfcatch` tag built-in variable `cfcatch.Type`. For more information, see Chapter 6, “Extending ColdFusion Pages with CFML Scripting” in ColdFusion MX Developer’s Guide.

### Invoking ColdFusion components with the `cfscript` tag

CFScript invokes component methods using the `CreateObject` function.

The following example shows how to invoke a component object with the `cfscript` tag, using ordered arguments:

```
<cfscript>
quote = CreateObject( "component", "nasdaq.quote" ) ;
<!--- Invocation using ordered arguments. --->
res = quote.getLastTradePrice( "macr" ) ;
</cfscript>
```

The following example shows how to use an attribute collection within the `cfscript` tag to pass parameters when invoking a component object. An attribute collection is a structure in which each key corresponds to a parameter name and each value is the parameter value passed for the corresponding key.

```
<cfscript>
   stArgs = structNew();
   stArgs.translationmode = "en_es";
   stArgs.sourceData= "Hello world, friend";
</cfscript>
```

```
<cfinvoke
   webservice = "http://www.xmethods.net/sd/2001/BabelFishService.wsdl"
   method     = "BabelFish"
   argumentCollection = "#stArgs#"
   returnVariable = "varName">
<cfoutput>#varName#</cfoutput>
```

In this example, the structure is created in a `cfscript` block, but you can use any ColdFusion method to create the structure.
Consuming web services with the cfscript tag

The following example shows how to consume a web service with the `cfscript` tag. You use the `CreateObject` function to connect to the web service.

```
<cfscript>
    ws = CreateObject("webservice", "http://www.xmethods.net/sd/2001/BabelFishService.wsdl");
    xlatstring = ws.BabelFish("en_es", "Hello world, friend");
    writeoutput(xlatstring);
</cfscript>
```

For more information, see Chapter 36, “Using Web Services” in *ColdFusion MX Developer’s Guide*.

Example

```
<p>This simple example shows variable declaration and manipulation.
<cfif IsDefined("form.myValue")>
    <cfif IsNumeric(form.myValue)>
        <cfset x = form.myValue>
        <cfscript>
            y = x;
            z = 2 * y;
            StringVar = form.myString;
        </cfscript>
        <cfoutput><p>twice #x# is #z#.
            <p>Your string value was: <b><I>#StringVar#</i></b></cfoutput>
    </cfif>
</cfelse>
```
**cfsearch**

**Description**

Searches one or more Verity collections.

A collection must be created and indexed before this tag can return search results.

A collection can be created in these ways:

- With the `cfcollection` tag
- In the ColdFusion MX Administrator
- Using a native Verity indexing tool, such as Vspider or MKVDK. For more information on Vspider and MKVDK, see Chapter 8, “Introducing Verity and Verity Tools” in *Configuring and Administering ColdFusion MX*.

If you use a native Verity tool to create a collection, it must be registered. A collection can be registered with ColdFusion in the following ways:

- With the `cfcollection` tag
- In the ColdFusion MX Administrator

A collection can be indexed in the following ways:

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

For more information, see Chapter 24, “Building a Search Interface” in *ColdFusion MX Developer’s Guide*.

**Category**

Extensibility tags

**Syntax**

```cfm
<cfsearch
  name = "search_name"
  collection = "collection_name"
  category = "category[,category2,...]"
  categoryTree = "tree_location"
  status = ""
  type = "criteria"
  criteria = "search_expression"
  maxRows = "number"
  startRow = "row_number"
  suggestions = "suggestion_option"
  contextPassages = "number_of_passages"
  contextBytes = "number_of_bytes"
  contextHighlightBegin = "html_string"
  contextHighlightEnd = "html_string"
  previousCriteria = "criteria"
  language = "language">
```

```html
</cfsearch>
```
See also
cfcollection, cfexecute, cfindex, cfobject, cfreport, cfwddx

History
ColdFusion MX 7:
- Added category, categoryTree, status, suggestions, contextPassages, contextBytes, contextHighlightBegin, contextHighlightEnd, and previousCriteria attributes.
- Added author, category, categoryTree, context, rank, size, recordsSearched, and type result columns.
- Added information on the status structure and its associated keys.
- Removed references to a separate K2 server and k2server.ini file.
- Removed references to unregistered collections.
- Removed references to external collections. ColdFusion MX now manages all collections through the Verity Search service.
- Changed cflock recommendation. It is no longer a best practice to surround the cfsearch tag with a cflock tag.

ColdFusion MX:
- Deprecated the external attribute. It might not work, and might cause an error, in later releases. (ColdFusion stores this information about each collection; it automatically detects whether a collection is internal or external.) This tag supports absolute (also known as fully qualified) collection pathnames and mapped collection names.
- Changed query result behavior: the cfindex tag can index the query results from a cfsearch operation.
- Changed Verity operations behavior: ColdFusion supports Verity operations on Acrobat PDF files.
- Changed multiple collection behavior: this tag can search multiple collections. In a multiple collection search, you cannot combine collections that are registered with K2Server and registered in another way.
- Changed acceptable collection naming: this tag accepts collection names that include spaces.
- Changed the following support: this tag supports Verity 2.6.1 and the LinguistX and ICU locales.
- Changed thrown exceptions: this tag can throw the SEARCHENGINE exception.

Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Req/Opt</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Required</td>
<td></td>
<td>Name of the search query.</td>
</tr>
<tr>
<td>collection</td>
<td>Required</td>
<td></td>
<td>One or more collection names. You can specify more than one collection unless you are performing a category search (that is, specifying category or categoryTree).</td>
</tr>
<tr>
<td>Attribute</td>
<td>Req/Opt</td>
<td>Default</td>
<td>Description</td>
</tr>
<tr>
<td>-------------</td>
<td>---------</td>
<td>---------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>category</td>
<td>Optional</td>
<td></td>
<td>A list of categories, separated by commas, to which the search is limited. If specified, and the collection does not have categories enabled, ColdFusion throws an exception.</td>
</tr>
<tr>
<td>categoryTree</td>
<td>Optional</td>
<td></td>
<td>The location in a hierarchical category tree at which to start the search. ColdFusion searches at and below this level. If specified, and the collection does not have categories enabled, ColdFusion throws an exception. Can be used in addition to the category attribute.</td>
</tr>
<tr>
<td>status</td>
<td>Optional</td>
<td></td>
<td>Specifies the name of the structure variable into which ColdFusion places search information, including alternative criteria suggestions (spelling corrections). For a list of keys in this structure, see “Status structure keys” on page 384.</td>
</tr>
<tr>
<td>type</td>
<td>Optional</td>
<td>simple</td>
<td>Used to specify the parser that Verity uses to process the criteria.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• simple: STEM and MANY operators are implicitly used.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• explicit: operators must be invoked explicitly. Also known as Bool_Plus.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• internet: for documents that are mostly WYSIWIG (what-you-see-is-what-you-get) documents. Also known as Internet_advanced.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• internet_basic: minimizes search time.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• natural: specifies the Query By Example (QBE) parser. Also known as FreeText.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>For more information, see Chapter 25, “Using Verity Search Expressions” in ColdFusion MX Developer’s Guide. Also see your Verity documentation.</td>
</tr>
<tr>
<td>criteria</td>
<td>Optional</td>
<td></td>
<td>Search criteria. Follows the syntax rules of the type attribute. If you pass a mixed-case entry in this attribute, the search is case-sensitive. If you pass all uppercase or all lowercase, the search is case-insensitive. Follow Verity syntax and delimiter character rules; see Chapter 25, “Using Verity Search Expressions” in ColdFusion MX Developer’s Guide.</td>
</tr>
<tr>
<td>maxRows</td>
<td>Optional</td>
<td>All</td>
<td>Maximum number of rows to return in query results.</td>
</tr>
<tr>
<td>startRow</td>
<td>Optional</td>
<td>1</td>
<td>First row number to get.</td>
</tr>
</tbody>
</table>
The **cfsearch** tag returns a query object whose columns you can reference in a **cfoutput** tag. For example, the following code specifies a search for the exact terms "filming" or "filmed":

```cfc
<cfsearch
  name = "mySearch"
  collection = "myCollection"
  criteria = '<WILDCARD>`film{ing,ed}`'
  type = "explicit"
  startrow = 1
  maxrows = "100">
<cfdump var = "#mySearch#>
```
In this example, the single-quotation mark (‘) and backtick (´) characters are used as delimiters; for more information, see Chapter 25, “Using Verity Search Expressions” in *ColdFusion MX Developer’s Guide*.

To optimize search performance, always specify the `maxrows` attribute, setting it to a value that matches your application’s needs. A value less than 300 helps to ensure optimal performance.

Macromedia does not recommend using the `cflock` tag with this tag; Verity provides the locking function. Using the `cflock` tag slows search performance.

**The cfsearch tag result columns**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>context</td>
<td>A context summary containing the search terms, highlighted in bold (by default). This is enabled if you set the <code>contextpassages</code> attribute to a number greater than zero.</td>
</tr>
<tr>
<td>url</td>
<td>Value of <code>URLpath</code> attribute in the <code>cfindex</code> tag used to populate a collection.</td>
</tr>
<tr>
<td>key</td>
<td>Value of the <code>key</code> attribute in the <code>cfindex</code> tag used to populate a collection.</td>
</tr>
<tr>
<td>title</td>
<td>Value of <code>title</code> attribute in <code>cfindex</code> tag used to populate the collection, including PDF and Office document titles. If a title is not extracted from the document, the tag uses the <code>cfindex title</code> attribute value for each row.</td>
</tr>
<tr>
<td>score</td>
<td>Relevancy score of document based on search criteria</td>
</tr>
<tr>
<td>custom1, custom2, custom3, custom4</td>
<td>Value of custom fields in <code>cfindex</code> tag used to populate a collection.</td>
</tr>
<tr>
<td>size</td>
<td>The number of bytes in the index document.</td>
</tr>
<tr>
<td>rank</td>
<td>The rank of this document in the search results.</td>
</tr>
<tr>
<td>author</td>
<td>Extracted from the HTML, Office, and PDF documents when available.</td>
</tr>
<tr>
<td>type</td>
<td>The MIME type of the document.</td>
</tr>
<tr>
<td>category</td>
<td>A list of the categories that were specified for this document when it was indexed.</td>
</tr>
<tr>
<td>categoryTree</td>
<td>A hierarchical category tree, or serial list of categories, that was specified for this document when it was indexed. Only a single tree is returned.</td>
</tr>
<tr>
<td>summary</td>
<td>Contents of automatic summary generated by <code>cfindex</code>.</td>
</tr>
<tr>
<td>recordCount</td>
<td>Number of records returned in record set.</td>
</tr>
<tr>
<td>currentRow</td>
<td>Current row that <code>cfoutput</code> is processing.</td>
</tr>
<tr>
<td>columnList</td>
<td>List of column names within record set.</td>
</tr>
<tr>
<td>recordsSearched</td>
<td>Number of records searched. This is the same value for each row in the record set. Corresponds to the <code>searched</code> key in the status structure.</td>
</tr>
</tbody>
</table>
To permit application users to search Verity collections for nonstandard strings, words, or characters (for example, "AB23.45.67" or "--->") that would otherwise cause an error, you can create a text file that lists these elements and defines their formats for Verity. Name the file style.lex and put copies of the file in these directories:

- **Windows:**
  - `cf_root\verity\k2\common\style` (typically, `cf_root = C:\CFusionMX7`)
  - `cf_root\verity\Data\stylesets\ColdFusionK2`

- **UNIX:**
  - `cf_root/verity/k2/common/style` (typically, `cf_root = /opt/coldfusionmx7`)
  - `cf_root/verity/Data/stylesets/ColdFusionK2`

In the multiserver and J2EE configurations, you install Verity in a separate directory. **Note:** To search for a character such as an angle bracket (< or >), you must use a `criteria` attribute value such as "&lt:" or "&gt:". The bracket characters are reserved in Verity, and using a backslash to escape the character (`criteria="\<"`) does not work in this context. For more information, see Chapter 25, "Using Verity Search Expressions" in ColdFusion MX Developer’s Guide.

**Example**

```xml
<!-#1 (TYPE=SIMPLE) ----------------------------->
<cfsearch
  name="name"
  collection="snippets.syntax.snippets"
  criteria="example"
  maxrows = "100">
<p>
<cfoutput>Search Result total = #name.RecordCount#</cfoutput><br>
<cfoutput>
  url=#name.url#<br>
  key=#name.key#<br>
</cfoutput>
</cfsearch>
```
title=#name.title#<br>
score=#name.score#<br>
custom1=#name.custom1#<br>
custom2=#name.custom2#<br>
summary=#name.summary#<br>
recordcount=#name.recordcount#<br>
currentrow=#name.currentrow#<br>
columnlist=#name.columnlist#<br>
recordssearched=#name.recordssearched#<br>
</cfoutput>
<cfdump var = #name#>
<br>
<!--- #2 (TYPE=EXPLICIT) ----------------------------->
<cfsearch
name = "snippets"
collection = "snippets"
criteria = '<wildcard>`film{ing,ed}`'
type="explicit"
startrow=1
maxrows = "100">
<cfoutput
query="snippets">
url=#url#<br>
key=#key#<br>
title=#title#<br>
score=#score#<br>
custom1=#custom1#<br>
custom2=#custom2#<br>
summary=#summary#<br>
recordcount=#recordcount#<br>
currentrow=#currentrow#<br>
columnlist=#columnlist#<br>
recordssearched=#recordssearched#<br>
</cfoutput>
<cfdump var = #snippets#>
<br>
<!--- #3 (search by CF key) ----------------------------->
<cfsearch
name = "book"
collection = "custom_book"
criteria = "cf_key=bookid2"
maxrows = "100">
<cfoutput
url=#book.url#<br>
key=#book.key#<br>
title=#book.title#<br>
score=#book.score#<br>
custom1=#book.custom1#<br>
custom2=#book.custom2#<br>
summary=#book.summary#<br>
recordcount=#book.recordcount#<br>
currentrow=book.currentrow<br>
columnlist=book.columnlist<br>
recordssearched=book.recordssearched<br>
</cfoutput>
<cfdump var = book>
cfselect

Description

Constructs a drop-down list box form control. Used within a cfform tag. You can populate the list from a query, or by using the HTML option tag.

Category

Forms tags

Syntax

```
<cfselect
    name = "name"
    label = "label"
    style = "style specification"
    size = "integer"
    required = "yes" or "no"
    message = "text"
    onError = "text"
    multiple = "yes" or "no"
    query = "queryname"
    value = "text"
    display = "text">
    group = "query column name"
    queryPosition = "above" or "below"
    selected = "value or list"
    onKeyUp = "JavaScript or ActionScript"
    onKeyDown = "JavaScript or ActionScript"
    onMouseUp = "JavaScript or ActionScript"
    onMouseDown = "JavaScript or ActionScript"
    onClick = "JavaScript or ActionScript"
    visible = "Yes" or "No"
    enabled = "Yes" or "No"
    tooltip = "tip text"
    height = "number of pixels" Flash only
    width = "number of pixels" Flash only
</cfselect>
```

zero or more HTML option tags

See also

cfapplet, cfcalendar, cfform, cfformgroup, cfformitem, cfgrid, cfform, cftextarea, cftree; Chapter 26, “Introduction to Retrieving and Formatting Data,” in ColdFusion MX Developer's Guide

History

ColdFusion MX 7:

- Added support for specifying multiple values to the selected attribute.
- Deprecated the passthrough attribute. The tag now supports all HTML select tag attributes directly.
• Added on-prefixed attributes.
• Added enabled, group, height, label, queryPosition, tooltip, visible, and width attributes.

Attributes

The following table lists attributes that ColdFusion uses directly. The tag also supports all HTML select tag attributes that are not on this list, and passes them directly to the browser.

Note: Attributes that are marked as Flash only are not handled by the skins provided with ColdFusion MX. They are, however, included in the generated XML.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Req/Opt; Format</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Required; All</td>
<td></td>
<td>Name of the select form element.</td>
</tr>
<tr>
<td>label</td>
<td>Optional; Flash and XML</td>
<td></td>
<td>Label to put next to the control on a Flash or XML-format form.</td>
</tr>
<tr>
<td>style</td>
<td>Optional; All</td>
<td></td>
<td>In HTML or XML format forms, ColdFusion passes the style attribute to the browser or XML. In Flash format, must be a style specification in CSS format, with the same syntax and contents as used in Macromedia Flex for the corresponding Flash element.</td>
</tr>
<tr>
<td>size</td>
<td>Optional; All</td>
<td>1</td>
<td>Number of entries to display at one time. The default, 1, displays a drop-down list. Any other value displays a list box with size number of entries visible at one time.</td>
</tr>
<tr>
<td>required</td>
<td>Optional; All</td>
<td>No</td>
<td>• Yes: a list element must be selected when the form is submitted. Note: This attribute has no effect if you omit the size attribute or set it to 1, because the browser always submits the displayed item. You can work around this issue: format forms by having an initial option tag with value=&quot;&quot; (note the space character between the quotation marks). • No</td>
</tr>
<tr>
<td>message</td>
<td>Optional; All</td>
<td></td>
<td>Message to display if required = &quot;Yes&quot; and no selection is made.</td>
</tr>
<tr>
<td>onError</td>
<td>Optional; HTML and XML</td>
<td></td>
<td>Custom JavaScript function to execute if validation fails.</td>
</tr>
<tr>
<td>multiple</td>
<td>Optional; All</td>
<td>No</td>
<td>• Yes: allows selecting multiple elements in drop-down list. • No</td>
</tr>
<tr>
<td>query</td>
<td>Optional; All</td>
<td></td>
<td>Name of query to populate drop-down list.</td>
</tr>
<tr>
<td>value</td>
<td>Optional; All</td>
<td></td>
<td>Query column to use for the value of each list element. Used with the query attribute.</td>
</tr>
<tr>
<td>Attribute</td>
<td>Req/Opt; Format</td>
<td>Default Description</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>-------------------------</td>
<td>----------------------------------------------------------</td>
<td>----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>display</td>
<td>Optional; All</td>
<td>Value of value attribute</td>
<td>Query column to use for the display label of each list element. Used with the query attribute.</td>
</tr>
<tr>
<td>group</td>
<td>Optional; HTML and XML</td>
<td></td>
<td>Query column to use to group the items in the drop-down list into a two-level hierarchical list.</td>
</tr>
<tr>
<td>queryPosition</td>
<td>Optional; All above</td>
<td></td>
<td>If you populate the options list with a query and use HTML option child tags to specify additional entries, determines the location of the items from the query relative to the items from the option tags: above: puts the query items above the options items. below: puts the query items below the options items.</td>
</tr>
<tr>
<td>selected</td>
<td>Optional; All</td>
<td></td>
<td>One or more option values to preselect in the selection list. To specify multiple values, use a comma-delimited list. This attribute applies only if selection list items are generated from a query. The cfformpreservedata attribute value can override this value.</td>
</tr>
<tr>
<td>onKeyUp</td>
<td>Optional; All</td>
<td></td>
<td>JavaScript (HTML/XML) or ActionScript (Flash) to run when the user releases a keyboard key in the control.</td>
</tr>
<tr>
<td>onKeyPress</td>
<td>Optional; All</td>
<td></td>
<td>JavaScript (HTML/XML) or ActionScript (Flash) ActionScript to run when the user presses a keyboard key in the control.</td>
</tr>
<tr>
<td>onMouseUp</td>
<td>Optional; All</td>
<td></td>
<td>JavaScript (HTML/XML) or ActionScript (Flash) to run when the user presses a mouse button in the control.</td>
</tr>
<tr>
<td>onMouseDown</td>
<td>Optional; All</td>
<td></td>
<td>JavaScript (HTML/XML) or ActionScript (Flash) to run when the user releases a mouse button in the control.</td>
</tr>
<tr>
<td>onChange</td>
<td>Optional; All</td>
<td></td>
<td>JavaScript (HTML/XML) or ActionScript (Flash) to run when the control changes due to user action.</td>
</tr>
<tr>
<td>onClick</td>
<td>Optional; HTML and XML</td>
<td></td>
<td>JavaScript to run when the user clicks the control.</td>
</tr>
<tr>
<td>enabled</td>
<td>Optional; Flash Yes</td>
<td>Boolean value specifying whether the control is enabled. A disabled control appears in light gray. The inverse of the disabled attribute.</td>
<td></td>
</tr>
<tr>
<td>visible</td>
<td>Optional; Flash Yes</td>
<td>Boolean value specifying whether to show the control. Space that would be occupied by an invisible control is blank.</td>
<td></td>
</tr>
<tr>
<td>tooltip</td>
<td>Optional; Flash</td>
<td>Text to display when the mouse pointer hovers over the control.</td>
<td></td>
</tr>
</tbody>
</table>
Note: Attributes that are marked as Flash only are not handled by the skins provided with ColdFusion MX. They are, however, included in the generated XML.

Usage

This tag requires an end tag and can include HTML option and optgroup child tags.

To ensure that a selected list box item persists across postbacks, use the cfform preserveData attribute with a list generated from a query. (This strategy works only with data that is populated from a query.)

If the cfform preserveData attribute is true and the form posts back to the same page, and if the control is populated by a query, the posted selection(s) for the cfselect control are used instead of the Selected attribute. For controls that are populated with regular HTML option tags, the developer must dynamically add the Selected attribute to the appropriate option tag(s).

The group option generates a query using SQL GROUP BY syntax and places the value column entries from each group in an indented list under the group column’s field value. This option generates an HTML optgroup tag for each entry in the group column.

Close each HTML option tag in the cfselect tag body with a </option> end tag. If you do not do so, and you specify queryPosition="below", the first item from the query might not appear in the list.

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

For more information, see the cfform tag entry.

Example

The following example lets you select one or more employee names from a list of all employees, grouped by departments, and displays the selected names and the employee's email addresses. It includes an option to get data for all employees.

```cfc
<!--- Get the employee names from the database. --->
<!--- Use SQL to create a Name field with first and last names. --->
<cfquery name = "GetAllEmployees" dataSource = "cfdocexamples"
cachedwithin="#createTimeSpan(0,1,0,0)#">
    SELECT Emp_ID, EMail, Phone, Department, FirstName, LastName, FirstName + ' ' + lastName as Name
    FROM Employees
    GROUP BY Department, Emp_ID, EMail, Phone, FirstName, LastName, FirstName + lastName
</cfquery>

<h2>cfselect Example</h2>
<!-- The cfif statement is true if the form was submitted. -->
Show the selected names. --->
<cfif IsDefined("form.employeeid")>
<!--- The form was submitted. --->
<h4>You Selected the following employees</h4>
<cfif form.employeeid IS ">
<!--- Select All option was selected. Show all employees. --->
<cfoutput query="GetAllEmployees">
#name#<br>
Email: #email#<br><br>
</cfoutput>
<cfelse>
<!--- Use a query of queries to get the data for the selected users.
Form.employeeid is a comma-delimited list of selected employee IDs. --->
<cfquery name = "GetSelectedEmployees" dbtype="query">
SELECT Emp_ID, EMail, Phone, Department, FirstName, LastName, FirstName 
+ ' ' +lastName as Name 
FROM   GetAllEmployees 
WHERE Emp_ID in (#form.employeeid#)
</cfquery>
<!--- Display the names and e-mail addresses from the query. --->
<cfoutput query="GetSelectedEmployees">
#firstName# #lastName#<br>
Email: #email#<br>
<br>
</cfoutput>
</cfif>
</cfif>

<!--- The cfform tag posts back to the current page. --->
<h3>Select one or more employees</h3>
<cfform action="#CGI.SCRIPT_NAME#">
<!--- Use cfselect to present the query's LastName column, grouped by department.
Allow Multiple selections.--->
<cfselect
    name = "employeeid"
    size = "15"
    multiple="yes"
    required = "Yes"
    message = "Select one or more employee names"
    query = "GetAllEmployees"
    group="Department"
    display ="name"
    value = "emp_id"
    queryPosition="Below">
<!--- Add an option to select all employees. --->
<option value = """>Select All</option>
</cfselect><br><br>
<input type="Submit">
</cfform>
cfservlet

Description
This tag is deprecated. Executes a Java servlet on a JRun engine.

To access servlets that run on the same server as ColdFusion, use code such as the following, in which *path* specifies a servlet, JSP, or anything else:

GetPageContext().include(path)
GetPageContext().forward(path)

For more information, see the JSP PageContext API or the Servlet RequestDispatcher API.

History
ColdFusion MX: Deprecated this tag. It might not work, and it might cause an error, in later releases.
cfservletparam

**Description**

This tag is deprecated.

A child tag of the `cfservlet` tag. Passes data to a servlet. Each `cfservletparam` tag within the `cfservlet` block passes a separate item of data to the servlet.

To access servlets that run on the same server as ColdFusion, use code such as the following, in which `path` specifies a servlet, JSP, or anything else:

```java
GetPageContext().include(path)
GetPageContext().forward(path)
```

For more information, see the JSP PageContext API or the Servlet RequestDispatcher API.

**History**

ColdFusion MX: Deprecated this tag. It might not work, and it might cause an error, in later releases.
**cfset**

**Description**
Sets a value in ColdFusion. Used to create a variable, if it does not exist, and assign it a value. Also used to call functions.

**Category**
Variable manipulation tags

**Syntax**
<cfset
    var variable_name = expression
>

**See also**
cfcookie, cfparam, cfregistry, cfsavecontent, cfschedule; Chapter 2, “Elements of CFML,” in ColdFusion MX Developer’s Guide

**Attributes**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Req/Opt</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>var</td>
<td>Optional</td>
<td></td>
<td>A keyword. Does not take a value. Identifies the variable as being local to a function. The variable only exists for the time of the current invocation of the function.</td>
</tr>
<tr>
<td>variable_name</td>
<td>Required</td>
<td></td>
<td>A variable.</td>
</tr>
</tbody>
</table>

**Usage**
The following sections provide detailed descriptions of some of the uses for the cfset tag.

**Calling functions**
When you use the cfset tag to call a function, you do not need to assign the function return value to a variable if the function does not return a value or you do not need to use the value returned by the function. For example, the following line is a valid ColdFusion cfset tag for deleting the MyVariable variable from the Application scope:
<cfset StructDelete(Application, "MyVariable")>

**Arrays**
The following example assigns a new array to the variable months:
<cfset months = ArrayNew(1)>

This example creates a variable Array_Length that resolves to the length of the array Scores:
<cfset Array_Length = ArrayLen(Scores)>

This example assigns, to index position two in the array months, the value February:
<cfset months[2] = "February">
Dynamic variable names

In this example, the variable name is itself a variable:

```cfset myvariable = "current_value">
<cfset "#myvariable#" = 5>
```

Function local variables

The `var` keyword specifies that the variable being defined is only available inside the body of a function that you define using the `cffunction` tag. The variable value that is set in one invocation of the function is not available in any other invocation of the function. The `var` keyword is the equivalent of the `var` statement in CFScript. The following rules apply to the `var` keyword:

- Any `cfset` tag that uses the `var` keyword must be inside the body of a `cffunction` tag. If you use the `var` keyword in a `cfset` tag outside a `cffunction` tag body, ColdFusion displays an error message.
- You must place all `cfset` tags that use the `var` keyword at the beginning of the `cffunction` tag body, before any other ColdFusion tags.

The following example shows how to use the new keyword:

```cffunction name="myFunct">
    <cfset var myVar = "This is a test">
    <cfreturn myVar & " Message."/>
</cffunction>
<cfoutput>#myFunct()#</cfoutput>
```

In this example, the variable `myVar` exists only when the function `myFunct` executes, and it is not available elsewhere on the ColdFusion page.

COM objects

In this example, a COM object is created. A `cfset` defines a value for each method or property in the COM object interface. The last `cfset` creates a variable to store the return value from the COM object's `SendMail` method.

```cfobject action = "Create"
    name = "Mailer"
    class = "SMTPsvg.Mailer">
<cfset MAILER.FromName = form.fromname>
<cfset MAILER.RemoteHost = RemoteHost>
<cfset MAILER.FromAddress = form.fromemail>
<cfset MAILER.AddRecipient("form.fromname", "form.fromemail")>
<cfset MAILER.Subject = "Testing cfobject">
<cfset MAILER.BodyText = "form.msgbody">
<cfset MAILER.SMTPLog = "logfile">
<cfset success = MAILER.SendMail()>
<cfoutput> #success# </cfoutput>
```

```
Example

<!-- This example shows how to use cfset. -->
<cfquery name="GetMessages" dataSource="cfdocexamples">
    SELECT *
    FROM Messages
</cfquery>

<h3>cfset Example</h3>
<p>cfset sets and reassigns values to local or global variables within a page.</p>
<cfset NumRecords = GetMessages.recordCount>
<p>For example, the variable NumRecords has been declared on this page to hold the number of records returned from query (</cfoutput>#NumRecords#</cfoutput>).</p>
<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>
    "hey, you, get off of my cloud"
</cfoutput>" 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>
<p>cfset can also be used to collect environmental variables, such as the time, the IP address of the user, or another function or expression.</p>
<cfset the_date = #DateFormat(Now())# & " " & #TimeFormat(Now())#>
<cfset user_ip = CGI.REMOTE_ADDR>
<cfset complex_expr = (23 MOD 12) * 3>
<cfset str_example = Reverse(Left(GetMessages.body, 35))>
<cfoutput>
<ul>
<li>The date: #the_date#
<li>User IP Address: #user_ip#
<li>Complex Expression ((23 MOD 12) * 3): #complex_expr#
<li>String Manipulation (the first 35 characters of the body of the first message in our query)
    <br><b>Reversed</b>: #str_example#
    <br><b>Normal</b>: #Reverse(str_example)#
</ul></cfoutput>
**cfsetting**

**Description**

Controls aspects of page processing, such as the output of HTML code in pages.

**Category**

Page processing tags, Variable manipulation tags

**Syntax**

```xml
<cfsetting
   enableCFoutputOnly = "yes" or "no"
   showDebugOutput = "yes" or "no"
   requestTimeOut = "value in seconds" >
```

**See also**

cfcache, cfflush, cfheader, cftimeout, cfinclude, cfprocessingdirective, cfsilent;
"Controlling debugging output with the cfsetting tag" in Chapter 18, “Debugging and Troubleshooting Applications,” in *ColdFusion MX Developer’s Guide*

**History**

ColdFusion MX 6.1: Changed behavior: if the tag has a body, ColdFusion executes its contents.

ColdFusion MX:

- Added the requestTimeOut attribute.
- The catchExceptionsByPattern attribute is obsolete. It does not work, and causes an error, in releases later than ColdFusion 5.
- Changed exception handling: the structured exception manager searches for the best-fit cfcatch handler. (In earlier releases, an exception was handled by the first cfcatch block that could handle an exception of its type.)

**Attributes**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Req/Opt</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>enableCFoutputOnly</td>
<td>Required</td>
<td></td>
<td>Yes: blocks output of HTML that is outside cfoutput tags, No: displays HTML that is outside cfoutput tags.</td>
</tr>
<tr>
<td>showDebugOutput</td>
<td>Optional</td>
<td>Yes</td>
<td>Yes: if debugging is enabled in the Administrator, displays debugging information. No: suppresses debugging information that would otherwise display at the end of the generated page.</td>
</tr>
<tr>
<td>requestTimeout</td>
<td>Optional</td>
<td></td>
<td>Integer; number of seconds. Time limit, after which ColdFusion processes the page as an unresponsive thread. Overrides the timeout set in the ColdFusion Administrator.</td>
</tr>
</tbody>
</table>
Usage

The `cfsetting requestTimeout` attribute replaces the use of `requestTimeOut` within a URL. To enforce a page timeout, detect the URL variable and use code such as the following to change the page timeout:

```cfsetting RequestTimeout = "#URL.RequestTimeout#"
```

You can use this tag to manage whitespace in ColdFusion output pages.

If you nest `cfsetting` tags: to make HTML output visible, you must match each `enableCFoutputOnly = "Yes"` statement with an `enableCFoutputOnly = "No"` statement. For example, after five `enableCFoutputOnly = "Yes"` statements, to enable HTML output, you must have five corresponding `enableCFoutputOnly = "No"` statements.

If HTML output is enabled (no matter how many `enableCFoutputOnly = "No"` statements have been processed) the first `enableCFoutputOnly = "Yes"` statement blocks output.

If the debugging service is enabled and `showDebugOutput = "Yes"`, the `IsDebugMode` function returns `Yes`; otherwise, `No`.

**Note:** Releases after ColdFusion MX allow a `</cfsetting>` end tag; however, this end tag does not affect processing. The `cfsetting` attributes affect code inside and outside the `cfsetting` tag body. ColdFusion MX ignored code between `cfsetting` start and end tags.

Example

```p
CFSETTING is used to control the output of HTML code in ColdFusion pages. This tag can be used to minimize the amount of generated whitespace.

<cfsetting enableCFoutputOnly = "Yes">
 This text is not shown
</cfsetting>
<cfsetting enableCFoutputOnly = "No">
 <p>This text is shown
</cfsetting>
<cfsetting enableCFoutputOnly = "Yes">
 <cfoutput>
  <p>Text within cfoutput is always shown
 </cfoutput>
</cfsetting>
```
cfsilent

Description
Suppresses output produced by CFML within a tag’s scope.

Category
Data output tags, Page processing tags

Syntax
<cfsilent>
...
</cfsilent>

See also
cfcache, cfflush, cfheader, cfhtmlhead, cfinclude, cfsetting; Chapter 9, “Writing and Calling User-Defined Functions,” in ColdFusion MX Developer’s Guide

Usage
This tag requires an end tag.

Example
<h3>cfsilent</h3>

<cfsilent>
  <cfset a = 100>
  <cfset b = 99>
  <cfset c = b-a>
  <cfoutput>Inside cfsilent block<br>b-a = #c#</cfoutput><br>
</cfsilent>

<p>Even information within cfoutput tags does not display within a cfsilent block.<br>
<cfoutput>
  b-a = #c#
</cfoutput>
</p>
cfslider

Description

Puts a slider control, for selecting a numeric value from a range, in a ColdFusion form. The slider moves over the slider groove. As the user moves the slider, the current value displays. Used within a cfiform tag. Not supported with Flash forms.

Category

Forms tags

Syntax

```xml
<cfslider
    name = "name"
    label = "text"
    range = "min_value, max_value"
    scale = "uinteger"
    value = "integer"
    onValidate = "script_name"
    message = "text"
    onError = "text"
    height = "integer"
    width = "integer"
    vSpace = "integer"
    hSpace = "integer"
    align = "alignment"
    lookAndFeel = "motif" or "windows" or "metal"
    vertical = "yes" or "no"
    bgColor = "color"
    textColor = "color"
    font = "font_name"
    fontSize = "integer"
    italic = "yes" or "no"
    bold = "yes" or "no"
    notSupported = "text">
</cfslider>
```

See also

cfapplet, cfcalendar, cfiform, cfiformgroup, cfiformitem, cfgrid, cfinput, cfselect, cftextarea, cftree; Chapter 26, "Introduction to Retrieving and Formatting Data,” and Chapter 27, “Building Dynamic Forms with cfiform Tags,” in ColdFusion MX Developer’s Guide

History

ColdFusion MX: Deprecated the img, imgStyle, grooveColor, refreshLabel, tickmarklabels, tickmarkmajor, tickmarkminor, and tickmarkimages attributes. They might not work, and might cause an error, in later releases.
## Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Req/Opt</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Required</td>
<td></td>
<td>Name of cfslider control.</td>
</tr>
<tr>
<td>label</td>
<td>Optional</td>
<td></td>
<td>Label to display with control; for example, &quot;Volume&quot;. This displays: &quot;Volume %value%&quot;. To reference the value, use &quot;%value%&quot;. If % is omitted, slider value displays directly after label.</td>
</tr>
<tr>
<td>range</td>
<td>Optional</td>
<td>&quot;0,100&quot;</td>
<td>Numeric slider range values. Separate values with a comma.</td>
</tr>
<tr>
<td>scale</td>
<td>Optional</td>
<td></td>
<td>Unsigned integer. Defines slider scale, within range. For example, if range = &quot;0,1000&quot; and scale = &quot;100&quot;, the display values are: 0, 100, 200, 300, ... Signed and unsigned integers in ColdFusion are in the range -2,147,483,648 to 2,147,483,647.</td>
</tr>
<tr>
<td>value</td>
<td>Optional</td>
<td>Minimum in range</td>
<td>Starting slider setting. Must be within the range values.</td>
</tr>
<tr>
<td>onValidate</td>
<td>Optional</td>
<td></td>
<td>Custom JavaScript function to validate user input; in this case, a change to the default slider value. Specify only the function name.</td>
</tr>
<tr>
<td>message</td>
<td>Optional</td>
<td></td>
<td>Message text to appear if validation fails.</td>
</tr>
<tr>
<td>.onError</td>
<td>Optional</td>
<td></td>
<td>Custom JavaScript function to execute if validation fails. Specify only the function name.</td>
</tr>
<tr>
<td>height</td>
<td>Optional</td>
<td>40</td>
<td>Slider control height, in pixels.</td>
</tr>
<tr>
<td>width</td>
<td>Optional</td>
<td></td>
<td>Slider control width, in pixels.</td>
</tr>
<tr>
<td>vSpace</td>
<td>Optional</td>
<td></td>
<td>Vertical spacing above and below slider, in pixels.</td>
</tr>
<tr>
<td>hSpace</td>
<td>Optional</td>
<td></td>
<td>Horizontal spacing to left and right of slider, in pixels.</td>
</tr>
<tr>
<td>align</td>
<td>Optional</td>
<td></td>
<td>Alignment of slider: • top • left • bottom • baseline • texttop • absbottom • middle • absmiddle • right</td>
</tr>
<tr>
<td>lookAndFeel</td>
<td>Optional</td>
<td>Windows</td>
<td>• motif: renders slider using Motif style. • windows: renders slider using Windows style. • metal: renders slider using Java Swing style. If platform does not support choice, the tag defaults to the platform's default style.</td>
</tr>
</tbody>
</table>
## Usage

This tag requires the client to download a Java applet. Using this tag may be slightly slower than using an HTML form element to display the same information. Also, if the client does not have an up-to-date Java plugin installed, the system might also have to download an updated Java plugin to display the tag.

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

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Req/Opt</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
</table>
| vertical   | Optional| No      | - Yes: renders slider in browser vertically. You must set width and height attributes; ColdFusion does not automatically swap width and height values.
|            |         |         | - No: renders slider horizontally. |
| bgColor    | Optional|         | Background color of slider label. For a hexadecimal value, use the form: bgColor = "####xxxxxx", where x = 0-9 or A-F; use two number signs or none. |
|            |         |         | - Any color, in hexadecimal format |
|            |         |         | - black |
|            |         |         | - red |
|            |         |         | - blue |
|            |         |         | - magenta |
|            |         |         | - cyan |
|            |         |         | - orange |
|            |         |         | - darkgray |
|            |         |         | - pink |
|            |         |         | - gray |
|            |         |         | - white |
|            |         |         | - lightgray |
|            |         |         | - yellow |
| textColor  | Optional|         | Options: same as for bgColor attribute. |
| font       | Optional|         | Font name for label text. |
| fontSize   | Optional|         | Font size for label text, in points. |
| italic     | Optional| No      | - Yes: label text in italics. |
|            |         |         | - No: normal text. |
| bold       | Optional| No      | - Yes: label text in bold. |
|            |         |         | - No: medium text. |
| notSupported | Optional|         | Text to display if a page that contains a Java applet-based cfform control is opened by a browser that does not support Java or has Java support disabled. For example: "<b>Browser must support Java to view ColdFusion Java Applets</b>"
|            |         |         | Default message: <b>Browser must support Java to view ColdFusion Java Applets</b>
If the following conditions are true, a user’s selection from query data that populates this tag’s options continues to display after the user submits the form:

- The `cfform preserveData` attribute is set to "Yes".
- The `cfform action` attribute posts to the same page as the form itself (this is the default), or the action page has a form that contains controls with the same names as corresponding controls on the user entry form.

For more information, see the `cfform` tag entry.

**Example**

```html
<!--- This example shows how to use cfslider within cfform. --->
<h3>cfslider Example</h3>
<p>cfslider, used within a cfform, can provide functionality to Java-enabled browsers. Try moving the slider back and forth to see the real-time value change. Then, submit the form to show how cfslider passes its value on to a new page.
</p>
<cfif isdefined("form.mySlider") is true>
<h3>You slid to a value of <cfoutput>#mySlider#</cfoutput></h3>
Try again!
</cfif>
<cfform action = "cfslider.cfm">
<cfslider name = "mySlider" value = "12"
label = "Actual Slider Value"
range = "1,100" align = "BASELINE"
message = "Slide the bar to get a value between 1 and 100"
height = "50" width = "150" font = "Verdana"
bgColor = "Silver" bold = "No"
italic = "Yes" refreshLabel = "Yes"> 100
<p><input type = "Submit" name = "" value = "Show the Result"/>
</cfform>
```
cfstoredproc

Description

Executes a stored procedure in a server database. It specifies database connection information and identifies the stored procedure.

Category

Database manipulation tags

Syntax

```xml
<cfstoredproc
    procedure = "procedure name"
    dataSource = "ds_name"
    username = "username"
    password = "password"
    blockFactor = "blocksize"
    debug = "yes" or "no"
    returnCode = "yes" or "no">
    result = "result_name"
</cfstoredproc>
```

See also

`cfinsert`, `cfqueryparam`, `cfproccparam`, `cfprocresult`, `cftransaction`, `cfquery`, `cfupdate`;

“Optimizing database use” in Chapter 13, “Designing and Optimizing a ColdFusion Application,” in *ColdFusion MX Developer’s Guide*

History

ColdFusion MX 7: Added the result attribute.

ColdFusion MX: Deprecated the connectString, dbName, dbServer, dbtype, provider, and providerDSN attributes. They do not work, and might cause an error, in releases later than ColdFusion 5. (ColdFusion MX uses Type 4 JDBC drivers.)

Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Req/Opt</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>procedure</td>
<td>Required</td>
<td></td>
<td>Name of stored procedure on database server.</td>
</tr>
<tr>
<td>dataSource</td>
<td>Required</td>
<td></td>
<td>Name of data source that points to database that contains stored procedure.</td>
</tr>
<tr>
<td>username</td>
<td>Optional</td>
<td></td>
<td>Overrides username in data source setup.</td>
</tr>
<tr>
<td>password</td>
<td>Optional</td>
<td></td>
<td>Overrides password in data source setup.</td>
</tr>
<tr>
<td>blockFactor</td>
<td>Optional</td>
<td>1</td>
<td>Maximum number of rows to get at a time from server. Range is 1 to 100.</td>
</tr>
</tbody>
</table>
| debug     | Optional | No      | • Yes: lists debug information on each statement.  
|           |         |         | • No |
Usage

Use this tag to call a database stored procedure. Within this tag, you code `cfprocresult` and `cfprocparam` tags as follows:

- `cfprocresult` If the stored procedure returns one or more result sets, code one `cfprocresult` tag per result set.

- `cfprocparam` If the stored procedure uses input or output parameters, code one `cfprocparam` tag per parameter, ensuring that you include every parameter in the stored procedure definition. Additionally, you must code `cfprocparam` tags in the same order as the parameters in the stored procedure definition.

If you set `returnCode = "Yes"`, this tag sets the variable `prefix.statusCode`, which holds the status code for a stored procedure. Status code values vary by DBMS. For the meaning of code values, see your DBMS documentation.

This tag sets the variable `prefix.ExecutionTime`, which contains the execution time of the stored procedure, in milliseconds.

The value of `prefix` is either `cfstoredproc` or the value specified by the `result` attribute, if it is set. The `result` attribute provides a way for stored procedures that are called from multiple pages, possibly at the same time, to avoid overwriting the results of one call with another. If you set the `result` attribute to `myResult`, for example, you would access `ExecutionTime` as `myResult.ExecutionTime`. Otherwise, you would access it as `cfstoredproc.ExecutionTime`.

Before implementing this tag, ensure that you understand stored procedures and their usage.

The following examples use a Sybase stored procedure; for an example of an Oracle 8 or 9 stored procedure, see `cfprocparam`.

Example

<!--- This view-only example executes a Sybase stored procedure that returns three result sets, two of which we want. The stored procedure returns the status code and one output parameter, which we display. We use named notation for the parameters. --->

```
<cfstoredproc procedure = "foo_proc"
    dataSource = "MY_SYBASE_TEST" username = "sa"
    password = "" dbServer = "scup" dbName = "pubs2"
    returnCode = "Yes" debug = "Yes">
  <cfprocresult name = RS1>
  <cfprocresult name = RS3 resultSet = 3>
```
<cfproccparam type = "IN" CFSQLType = CF_SQL_INTEGER
    value = "1" dbVarName = @param1>
<cfproccparam type = "OUT" CFSQLType = CF_SQL_DATE
    variable = FOO dbVarName = @param2>
</cfstoredproc>

---

<!---
<cfoutput>The output param value: '#foo#'<br></cfoutput>
<h3>The Results Information</h3>
<cfoutput query = RS1>#name#,#DATE_COL#<br></cfoutput><p>
<cfoutput>
<hr>
<p>Record Count: #RS1.recordCount#  Columns: #RS1.columnList#<hr>
</cfoutput>
<cfoutput query = RS3>#col1#,#col2#,#col3#<br>
</cfoutput><p>
<cfoutput>
<hr>
<p>Record Count: #RS3.recordCount#  Columns: #RS3.columnList#<hr>
The return code for the stored procedure is: '#cfstoredproc.statusCode#'<br>
</cfoutput>
---->
cfswitch

Description

Evaluates a passed expression and passes control to the cfcase tag that matches the expression result. You can, optionally, code a cfdefaultcase tag, which receives control if there is no matching cfcase tag value.

Category

Flow-control tags

Syntax

```html
<cfswitch
   expression = "expression">
   one or more cfcase tags
   zero or one cfdefaultcase tags
</cfswitch>
```

See also

cfcase, cfdefaultcase, cfabort, cfloop, cfbreak, cfexecute, cfexit, cfi, cflocation, cfrethrow, cfthrow, cftry; "cfswitch, cfcase, and cfdefaultcase" in Chapter 2, “Elements of CFML,” in ColdFusion MX Developer’s Guide

History

ColdFusion MX: Changed cfdefaultcase tag placement requirements: you can put the cfdefaultcase tag at any position within a cfswitch statement; it is not required to be the last item.

Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Req/Opt</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>expression</td>
<td>Required</td>
<td></td>
<td>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.</td>
</tr>
</tbody>
</table>

Usage

This tag requires an end tag. All code within this tag must be within a cfcase or cfdefaultcase tag. Otherwise, ColdFusion throws an error.

Use this tag followed by one or more cfcase tags. Optionally, include a cfdefaultcase tag. This tag selects the matching alternative from the cfcase and cfdefaultcase tags, jumps to the matching tag, and executes the code between the cfcase start and end tags.

The cfswitch tag provides better performance than a series of cfif/cfelseif tags, and the code is easier to read.
Example

<!--- This example shows the use of cfswitch and cfcase to exercise a case statement in CFML. --->
<cfquery name = "GetEmployees" dataSource = "cfdocexamples">
  SELECT Emp_ID, FirstName, LastName, EMail, Phone, Department 
  FROM   Employees
</cfquery>

<h3>cfswitch Example</h3>

<!--- By outputting the query and using cfswitch, we classify the output without using a cfloop construct. --->
<p>Each time the case is fulfilled, the specific information is printed; if the case is not fulfilled, the default case is output</p>
<cfoutput query="GetEmployees">
<cfswitch expression="#Trim(Department)#">
  <cfcase value="Sales">
    #FirstName# #LastName# is in <b>sales</b><br><br>
  </cfcase>
  <cfcase value="Accounting">
    #FirstName# #LastName# is in <b>accounting</b><br><br>
  </cfcase>
  <cfcase value="Administration">
    #FirstName# #LastName# is in <b>administration</b><br><br>
  </cfcase>
  <cfdefaultcase>
    #FirstName# #LastName# is not in Sales, Accounting, or Administration.<br><br>
  </cfdefaultcase>
</cfswitch>
</cfoutput>
cftable

Description

Builds a table in a ColdFusion page. This tag renders data as preformatted text, or, with the HTMLTable attribute, in an HTML table. If you don’t want to write HTML table tag code, or if your data can be presented as preformatted text, use this tag.

Preformatted text (defined in HTML with the <pre> and </pre> tags) displays text in a fixed-width font. It displays white space and line breaks exactly as they are written within the pre tags. For more information, see an HTML reference guide.

To define table column and row characteristics, use the cfcol tag within this tag.

Category

Data output tags

Syntax

```coldfusion
<cftable
    query = "query_name"
    maxRows = "maxrows_table"
    colSpacing = "number_of_spaces"
    headerLines = "number_of_lines"
    HTMLTable
    border
    colHeaders
    startRow = "row_number">
    ...
</cftable>
```

See also

cfcol, cfcontent, cflog, cfoutput, cfprocessingdirective, cftable; “Retrieving data” in Chapter 20, “Accessing and Retrieving Data,” in ColdFusion MX Developer’s Guide

Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Req/Opt</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>query</td>
<td>Required</td>
<td></td>
<td>Name of cfquery from which to draw data.</td>
</tr>
<tr>
<td>maxRows</td>
<td>Optional</td>
<td></td>
<td>Maximum number of rows to display in the table.</td>
</tr>
<tr>
<td>colSpacing</td>
<td>Optional</td>
<td>2</td>
<td>Number of spaces between columns.</td>
</tr>
<tr>
<td>headerLines</td>
<td>Optional</td>
<td>2</td>
<td>Number of lines to use for table header (the default leaves one line between header and first row of table).</td>
</tr>
<tr>
<td>HTMLTable</td>
<td>Optional</td>
<td></td>
<td>Renders data in an HTML 3.0 table. If you use this attribute (regardless of its value), ColdFusion renders data in an HTML table.</td>
</tr>
<tr>
<td>border</td>
<td>Optional</td>
<td></td>
<td>Displays border around table. If you use this attribute (regardless of its value), ColdFusion displays a border around the table. Use this only if you use the HTMLTable attribute.</td>
</tr>
</tbody>
</table>
### Usage

This tag aligns table data, sets column widths, and defines column heads.

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

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

### Example

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

<html>
<body>
<h3>cftable Example</h3>
<!--- Note use of HTMLTable attribute to display cftable as an HTML table, rather than as PRE formatted information. --->
<cftable query = "GetEmployees" startRow = "1" colSpacing = "3" HTMLTable = "true">
    <!--- Each cfcol tag sets width of a column in table, and specifies header information and text/CFML with which to fill cell. --->
    <cfcol header = "<b>ID</b>" align = "Left" width = 2 text = "+\Emp_ID+">
    <cfcol header = "<b>Name/Email</b>" align = "Left" width = 15 text = "<a href = 'mailto:\Email'>\FirstName #LastName</a>"
    <cfcol header = "<b>Phone Number</b>" align = "Center" width = 15 text = "\Phone">
</cftable>
</body>
</html>
```
**cftextarea**

**Description**

Puts a multiline text entry box in a *cfform* tag and controls its display characteristics.

**Category**

*Forms tags*

**Syntax**

```
<cftextarea
    name = "name"
    label = "text"
    required = "yes" or "no"
    style = "style specification"
    validate = "data type"
    validateAt= one or more of "onBlur", "onServer", "onSubmit"
    message = "text"
    range = "min_value, max_value"
    maxLength = "number"
    pattern = "regexp"
    onValidate = "script name"
    onError = "script name"
    disabled = "true" "false" or no attribute value
    value = "text"
    onKeyUp = "JavaScript or ActionScript"
    onKeyDown = "JavaScript or ActionScript"
    onMouseUp = "JavaScript or ActionScript"
    onMouseDown = "JavaScript or ActionScript"
    onChange = "JavaScript or ActionScript"
    onClick = "JavaScript or ActionScript"
    visible = "Yes" or "No"
    enabled = "Yes" or "No"
    tooltip = "Tip text"
    height = "number of pixels" Flash only
    width = "number of pixels" Flash only
>
    text
</cftextarea>
```

**See also**

*cfapplet*, *cfcalendar*, *cfform*, *cfformgroup*, *cfformitem*, *cfgrid*, *cfinput*, *cfselect*, *cfslider*, *cftree*; Chapter 26, “Introduction to Retrieving and Formatting Data,” in *ColdFusion MX Developer’s Guide*

**History**

ColdFusion MX 7: Added this tag.

**Attributes**

The following table lists attributes that ColdFusion uses directly. In HTML format, the tag also supports all HTML *textarea* tag attributes that are not on this list, and passes them directly to the browser.
Note: Attributes that are marked as Flash only are not handled by the skins provided with ColdFusion MX. They are, however, included in the generated XML.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Req/Opt; Format</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Required; All</td>
<td>Name of the cftextinput control.</td>
<td></td>
</tr>
<tr>
<td>label</td>
<td>Optional; Flash and XML</td>
<td>Label to put beside the control on a form.</td>
<td></td>
</tr>
<tr>
<td>style</td>
<td>Optional; All</td>
<td>In HTML or XML format forms, ColdFusion passes the style attribute to the browser or XML. In Flash format forms, must be a style specification in CSS format, with the same syntax and contents as used in Macromedia Flex for the corresponding Flash element.</td>
<td></td>
</tr>
<tr>
<td>required</td>
<td>Optional; All</td>
<td>no</td>
<td>• yes: the field must contain text. • no: the field can be empty.</td>
</tr>
<tr>
<td>validate</td>
<td>Optional; All</td>
<td>The type or types of validation to do. Available validation types and algorithms depend on the format. For details, see the Usage section of the cfinput tag reference.</td>
<td></td>
</tr>
<tr>
<td>validateAt</td>
<td>Optional; onSubmit HTML and XML</td>
<td>How to do the validation; one or more of the following: • onSubmit • onServer • onBlur For Flash format forms, onSubmit and onBlur are identical; validation is done when the user submits the form. For multiple values, use a comma-delimited list. For details, see the Usage section of the cfinput tag reference.</td>
<td></td>
</tr>
<tr>
<td>message</td>
<td>Optional; All</td>
<td>Message text to display if validation fails.</td>
<td></td>
</tr>
<tr>
<td>range</td>
<td>Optional; All</td>
<td>Minimum and maximum allowed numeric values. ColdFusion uses this attribute only if you specify range in the validate attribute. If you specify a single number or a single number followed by a comma, it is treated as a minimum, with no maximum. If you specify a comma followed by a number, the maximum is set to the specified number, with no minimum.</td>
<td></td>
</tr>
<tr>
<td>maxLength</td>
<td>Optional; All</td>
<td>The maximum length of text that can be entered. ColdFusion uses this attribute only if you specify maxLength in the validate attribute.</td>
<td></td>
</tr>
<tr>
<td>Attribute</td>
<td>Req/Opt: Format</td>
<td>Default</td>
<td>Description</td>
</tr>
<tr>
<td>-----------</td>
<td>----------------</td>
<td>---------</td>
<td>-------------</td>
</tr>
<tr>
<td>pattern</td>
<td>Required if validate = regular_expression HTML and XML</td>
<td>JavaScript regular expression pattern to validate input. Omit leading and trailing slashes. ColdFusion uses this attribute only if you specify regex in the validate attribute. For examples and syntax, see Chapter 27, &quot;Building Dynamic Forms with cfform Tags&quot; in ColdFusion MX Developer's Guide.</td>
<td></td>
</tr>
<tr>
<td>onValidate</td>
<td>Optional; HTML and XML</td>
<td>Custom JavaScript function to validate user input. The JavaScript DOM form object, input object, and input object value are passed to routine, which should return True if validation succeeds, False otherwise. If you specify this attribute, ColdFusion ignores the validate attribute.</td>
<td></td>
</tr>
<tr>
<td>onError</td>
<td>Optional; HTML and XML</td>
<td>Custom JavaScript function to execute if validation fails.</td>
<td></td>
</tr>
<tr>
<td>disabled</td>
<td>Optional; All disabled</td>
<td>Disables user input, making the control read-only. To disable input, specify disabled without an attribute, or disabled=&quot;Yes&quot; (or any ColdFusion positive Boolean value, such as True). To enable input, omit the attribute or specify disabled=&quot;No&quot; (or any ColdFusion negative Boolean value, such as False).</td>
<td></td>
</tr>
</tbody>
</table>
| value     | Optional; All | Initial value to display in text control. You can specify an initial value as an attribute or in the tag body, but not in both places. If you specify the value as an attribute, you must put the closing cftextarea tag immediately after the opening cftextarea tag, with no spaces or line feeds between, or place a closing slash at the end of the opening cftextarea tag; for example <cftextarea name="description" value="Enter a description."/>
| bind      | Optional; Flash | A Flex bind expression that populates the field with information from other form fields. For details, see Usage. |
| onKeyUp    | Optional; All | JavaScript (HTML/XML) or ActionScript (Flash) to run when the user releases a keyboard key in the control. |
| onKeyDown  | Optional; All | JavaScript (HTML/XML) or ActionScript (Flash) to run when the user presses a keyboard key in the control. |
| onMouseUp  | Optional; All | JavaScript (HTML/XML) or ActionScript (Flash) to run when the user presses a mouse button in the control. |
| onMouseDown| Optional; All | JavaScript (HTML/XML) or ActionScript (Flash) to run when the user releases a mouse button in the control. |
| onChange  | Optional; All | JavaScript (HTML/XML) or ActionScript (Flash) to run when the control changes due to user action. |
| onClick   | Optional; All | JavaScript (HTML/XML) to run when the user clicks the control. Not supported for Flash forms. |
Note: Attributes that are marked as Flash only are not handled by the skins provided with ColdFusion MX. They are, however, included in the generated XML.

Usage
For this tag to work properly in HTML format, the browser must be JavaScript-enabled.

If you put text in the tag body, the control displays all text characters between the cftextarea opening and closing tags; therefore, if you use line feeds or white space to format your source text, they appear in the control.

If the cfform preserveData attribute is "yes", and the form posts back to the same page, the posted value (not the value of the value attribute) of the cftextinput control is used.

Validation
For a detailed description of the validation attribute and the types of validation supported by ColdFusion, see the Usage section of the cfinput tag reference. For more details on ColdFusion MX validation techniques, see Chapter 28, “Validating Data” in ColdFusion MX Developer’s Guide.

Flash form data binding
The bind attribute lets you populate form fields using the contents of other form fields. To specify text from another field in a cftextarea bind attribute, use the following format:

\{sourceTagName.text\}

For example, the following line uses the value of the text that the user enters in the from the userName field in the greeting in the comment text box. The user can change or replace this message with a typed entry.

```xml
<cfformitem type="text">
  Enter your name here</cfformitem>
<cftextarea name="userName" height="20" Width="500"/>
<cftextarea name="comment" html height="300" Width="500"
```
Example

This example has two cftextarea controls. When you submit the form, ColdFusion copies the text from the first control into the second. The onBlur maxlength validation prevents you from entering more than 100 characters. The > character that closes the cftextarea opening tag, the text in the tag body, and the cftextarea closing tag are on a single line to ensure that only the desired text displays, but the line is split in this example for formatting purposes.

<h3>cftextarea Example</h3>
<cfparam name="text2" default="">
<cfif isdefined("form.textarea1") AND (form.textarea1 NEQ ")
  <cfset text2=form.textarea1>
</cfif>

<cfform name="form1">
  <cftextarea name="textarea1" wrap="virtual" rows="5" cols="25" validate="maxlength" validateAt="onBlur" maxlength="100">
    Replace this text. Maximum length is 100 Characters, and this text is currently 99 characters long.</cftextarea>
  <cftextarea name="textarea2" wrap="virtual" rows="5" cols="50" value="#text2#" />
  <input type="submit" value="submit field" />
</cfform>
cftextinput

Description

Puts a single-line text entry box in a cfform tag and controls its display characteristics.

This tag is deprecated, and is not supported in XML format forms. In its place, you should use a cfinput or cftextarea tag and use a cascading style sheet (CSS) to specify the text style characteristics.

History

ColdFusion MX 7: This tag is deprecated. In later releases it might not work, and might cause an error.

ColdFusion MX 6.1: Changed the validate = "creditcard" option requirements: the text entry must have 13-16 digits.
**cfthrow**

**Description**

Throws a developer-specified exception, which can be caught with a `cfcatch` tag that has any of the following `type` attribute options:

- `type = "custom_type"`
- `type = "Application"
- `type = "Any"

**Category**

Exception handling tags, Flow-control tags

**Syntax 1**

```xml
<cfthrow
type = "exception_type"
message = "message"
detail = "detail_description"
errorCode = "error_code"
extendedInfo = "additional_information"
object = "java_except_object">
```

**Syntax 2**

```xml
<cfthrow
object = #object_name#>
```

**See also**

cferror, cfrethrow, cftry, onError; Chapter 14, “Handling Errors” in ColdFusion MX Developer’s Guide

**History**

ColdFusion MX: Changed thrown exceptions: this tag can throw ColdFusion component method exceptions.

**Attributes**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Req/Opt</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>Optional</td>
<td>Application</td>
<td>• A custom type</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Application</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Do not enter another predefined type; types are not generated by ColdFusion applications. If you specify Application, you need not specify a type for <code>cfcatch</code>.</td>
</tr>
<tr>
<td>message</td>
<td>Optional</td>
<td></td>
<td>Message that describes exception event.</td>
</tr>
<tr>
<td>detail</td>
<td>Optional</td>
<td></td>
<td>Description of the event. ColdFusion appends error position to description; server uses this parameter if an error is not caught by your code.</td>
</tr>
<tr>
<td>errorCode</td>
<td>Optional</td>
<td></td>
<td>A custom error code that you supply.</td>
</tr>
</tbody>
</table>
Usage

Use this tag within a `cftry` block, to throw an error. The `cfcatch` block can access accompanying information, as follows:

- **Message**, with `cfcatch.message`
- **Detail**, with `cfcatch.detail`
- **Error code**, with `cfcatch.errorcode`

To get more information, use `cfcatch.tagContext`. This array shows where control switches from one page to another in the tag stack (for example, `cfinclude`, `cfmodule`).

To display the information displayed by `tagContext`: in the ColdFusion MX Administrator, Debugging page, select Enable CFML Stack Trace.

**Using the object parameter** To use this tag with the `object` parameter, you must first use a `cfobject` tag that specifies a valid Java exception class. For example, the following `cfobject` tag defines an object, `obj`, of the exception class `myException` (which you must create in Java):

```coldfusion
<cfobject
type="java"
action="create"
class="myException"
name="obj">
```

If your exception class has constructors that take parameters, such as a message, you can use the special `init` method to invoke the constructor, as in the following line. If you do not need to specify any constructor attributes, you can omit this step.

```coldfusion
<cfset obj.init("You must save your work before preceding")>
```

You can then use the, the `cfthrow` statement to throw the exception as follows:

```coldfusion
<cfthrow object=#obj#>
```

For more information on using Java objects in ColdFusion, see Chapter 37, “Integrating J2EE and Java Elements in CFML Applications” in *ColdFusion MX Developer’s Guide*.

**Example 1**

```coldfusion
<h3>cfthrow Example</h3>
<!--- Open a cftry block. --->
<cftry>
<!--- Define a condition upon which to throw the error. --->
<cfif NOT IsDefined("URL.myID")>
<!--- throw the error --->
<cfthrow message = "ID is not defined">
</cfif>
</cftry>
```
Example2

The following example shows how to throw an exception from a component method:

```coldfusion
<cfcomponent>
  <cffunction name="getEmp" required="yes">
    <cfargument name="lastName" required="yes">
      <cfquery name="empQuery" datasource="cfdocexamples">
        SELECT LASTNAME, FIRSTNAME, EMAIL
        FROM tblEmployees
        WHERE LASTNAME LIKE '#arguments.lastName#'
      </cfquery>
      <cfif empQuery.recordcount LT 1>
        <cfthrow type="noQueryResult"
          message="No results were found. Please try again."/>
      </cfif>
      <cfelse>
        <cfreturn empQuery>
      </cfelse>
  </cffunction>
</cfcomponent>
```

For an explanation of the example and more information, see Chapter 10, “Building and Using ColdFusion Components” in ColdFusion MX Developer’s Guide.
cftimer

Description
Displays execution time for a specified section of CFML code. ColdFusion MX displays the timing information along with any output produced by the timed code.

Note: To permit this tag to execute, you must enable the Timer Information option under Debugging Settings in the ColdFusion MX Administrator.

Category
Debugging tags

Syntax

```<cftimer
   label= "text"
   type = "inline" or "outline" or "comment" or "debug" >
   CFML statement(s)
</cftimer>```

See also
`cfdump, cftrace`; Chapter 18, “Debugging and Troubleshooting Applications” in *ColdFusion MX Developer’s Guide*

History
ColdFusion MX 7: Added this tag.

Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Req/Opt</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>label</td>
<td>Optional</td>
<td>&quot; &quot;</td>
<td>Label to display with timing information.</td>
</tr>
</tbody>
</table>
| type      | Optional | debug   | • inline: displays timing information inline, following the resulting HTML.
|           |          |         | • outline: displays timing information and also displays a line around the output produced by the timed code. The browser must support the FIELDSET tag to display the outline.
|           |          |         | • comment: displays timing information in an HTML comment in the format <!-- label: elapsed-time ms -->. The default label is cftimer.
|           |          |         | • debug: displays timing information in the debug output under the heading CFTimer Times. |

Usage
Use this tag to determine how long it takes for a block of code to execute. You can nest `cftimer` tags.

This tag is useful for debugging CFML code during application development. In production, you can leave `cftimer` tags in your code as long as you have disabled the debugging option in the ColdFusion MX Administrator.
Example 1

...  

<!--- type="inline" --->

<cftimer label="Query and Loop Time Inline" type="inline">
  <cfquery name="empquery" datasource="cfdocexamples">
    select * 
    from Employees 
  </cfquery>

  <cfloop query="empquery">
    <cfoutput>#lastname#, #firstname#</cfoutput><br>
  </cfloop>
</cftimer>

<hr><br>

<!--- type="outline" --->

<cftimer label="Query and CFOUTPUT Time with Outline" type="outline">
  <cfquery name="coursequery" datasource="cfdocexamples">
    select * 
    from CourseList 
  </cfquery>

  <table border="1" width="100%">
    <cfoutput query="coursequery">
      <tr>
        <td>#Course_ID#</td> 
        <td>#CorName#</td> 
        <td>#CorLevel#</td>
      </tr>
    </cfoutput>
  </table>
</cftimer>

<hr><br>

<!--- type="comment" --->

<cftimer label="Query and CFOUTPUT Time in Comment" type="comment">
  <cfquery name="parkquery" datasource="cfdocexamples">
    select * 
    from Parks 
  </cfquery>

  <p>Select View &gt; Source to see timing information</p>

  <table border="1" width="100%">
    <cfoutput query="parkquery">
      <tr>
        <td>#Parkname#</td>
      </tr>
    </cfoutput>
  </table>
</cftimer>

<hr><br>

<!--- type="debug" --->

<cftimer label="Query and CFOUTPUT Time in Debug Output" type="debug">
  <cfquery name="deptquery" datasource="cfdocexamples">
    select * 
  </cfquery>

  <p>Select View &gt; Source to see timing information</p>

  <table border="1" width="100%">
    <cfoutput query="deptquery">
      <tr>
        <td>#DeptName#</td> 
        <td>#DeptID#</td> 
        <td>#DeptDesc#</td> 
      </tr>
    </cfoutput>
  </table>
</cftimer>
from Departments
</cfquery>

<p>Scroll down to CFTimer Times heading to see timing information</p>
<table border="1" width="100%">
<tr>
<td>#Dept_ID#</td>
<td>#Dept_Name#</td>
</tr>
</table>
</cftimer>
**cftrace**

**Description**

Displays and logs debugging data about the state of an application at the time the `cftrace` tag executes. Tracks runtime logic flow, variable values, and execution time. Displays output at the end of the request or in the debugging section at the end of the request; or, in Dreamweaver MX and later, in the Server Debug tab of the Results window.

ColdFusion logs `cftrace` output to the file logs\cftrace.log, in the ColdFusion installation directory.

**Note:** To permit this tag to execute, you must enable debugging in the ColdFusion MX Administrator. Optionally, to report trace summaries, enable the Trace section.

**Category**

Debugging tags, Variable manipulation tags

**Syntax**

```xml
<cftrace
    abort = "Yes" or "No"
    category = "string"
    inline = "Yes" or "No"
    text = "string"
    type = "format"
    var = "variable_name"
</cftrace>
```

**See also**

cfdump, cferror, cfrethrow, cftimer, cftry; Chapter 18, “Debugging and Troubleshooting Applications” in ColdFusion MX Developer’s Guide

**History**

ColdFusion MX: Added this tag.

**Attributes**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Req/Opt</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
</table>
| abort     | Optional| No      | • Yes: calls `cfabort` tag when the tag is executed.  
|           |         |         | • No        |
| category  | Optional|         | User-defined string for identifying trace groups. |
| inline    | Optional| No      | • Yes: displays trace code in line on the page in the location of the `cftrace` tag, in addition to the debugging information output.  
|           |         |         | • No        |
| text      | Optional|         | User-defined string, which can include simple variables, but not complex variables such as arrays. Outputs to the `cflog` text attribute. |
Usage
You cannot put application code within this tag. (This avoids problems that can occur if you
disable debugging.)

This tag is useful for debugging CFML code during application development.

You can display cftrace tag output in the following ways:

• As a section in the debugging output

• In-line in an application page, and as a section in debugging output. If you specify in-line
  tracing, ColdFusion flushes all output up to the cftrace tag, and displays the trace output
  when it encounters the tag.

This is an example of a log file entry:

"Information"."web-4"."04/08/02"."23:21:30".  ."[30 ms (1st trace)]
[C:\CFusionMX7\wwwroot\generic.cfm @ line: 9] -
  [thisPage = /generic.cfm]  "
"Information"."web-0"."04/08/02"."23:58:58".  ."[5187 ms (10)]
[C:\CFusionMX7\wwwroot\generic.cfm @ line: 14] - [category]
  [thisPage = /generic.cfm] [ABORTED] thisPage "

For a complex variable, ColdFusion lists the variable name and the number of elements in the
object; it does not log the contents of the variable.

Example

The following example traces a FORM variable that is evaluated by a cfif block:

```cftrace var="FORM.variable"
  text="doing equivalency check for FORM.variable"
  category="form_vars"
  inline="true">
  <cfif isDefined("FORM.variable") AND #FORM.variable# EQ 1>
    <h1>Congratulations, you're a winner!</h1>
  </cfifelse>
  <h1>Sorry, you lost!</h1>
</cftrace>
cftransaction

Description
For enterprise database management systems that support transaction processing, instructs the
database management system to treat multiple database operations as a single transaction.
Provides database commit and rollback processing. See the documentation for your database
management system to determine whether it supports SQL transaction processing.

Category
Database manipulation tags

Syntax
<cftransaction
    action = "begin" or "commit" or "rollback"
    isolation = "read_uncommitted" or "read_committed" or
               "repeatable_read" >
</cftransaction>

See also
cfinsert, cfprocparam, cfprocresult, cfquery, cfqueryparam, cfstoredproc, cfupdate;
“Commits, rollbacks, and transactions” in Chapter 19, “Introduction to Databases and SQL,” in
ColdFusion MX Developer’s Guide

Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Req/Opt</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>action</td>
<td>Optional</td>
<td>begin</td>
<td>• begin: the start of the block of code to execute.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• commit: commits a pending transaction.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• rollback: rolls back a pending transaction.</td>
</tr>
<tr>
<td>isolation</td>
<td>Optional</td>
<td></td>
<td>ODBC lock type:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• read_uncommitted: reads without regard for other transactions</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>that are taking place. Sometimes called a ‘dirty read’ because data that is read can be in a transitional state and therefore not accurate.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• read_committed: uses shared locks to assure that no other transaction modifies rows that this transaction uses.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• repeatable_read: same as read_committed, except that rows in the recordset are exclusively locked until the transaction completes. Due to high overhead, Macromedia does not recommend this isolation level for normal database access.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• serializable: places an exclusive lock on every data table in use for the duration of the transaction. Causes access to the database to be single-threaded and is therefore not recommended for normal database access.</td>
</tr>
</tbody>
</table>
Usage

If you do not specify a value for the action attribute, automatic transaction processing proceeds as follows:

- If the cfquery operations within the transaction block complete without an error, the transaction is committed.
- If a cfquery tag generates an error within a cftransaction block, all cfquery operations in the transaction roll back.

If you do not specify a value for the isolation attribute, ColdFusion uses the default isolation level for the associated database.

Using CFML error handling and the action attribute, however, you can also explicitly control whether a transaction is committed or rolled back, based on the success or failure of the database query. Within a transaction block, you can do the following:

- Commit a database transaction by nesting the `<cftransaction action = "commit"/>` tag within the block
- Roll back a transaction by nesting the `<cftransaction action = "rollback"/>` tag within the block

(In these examples, the slash is alternate syntax that is the equivalent of an end tag.)

Within a transaction block, you can write queries to more than one database, but you must commit or roll back a transaction to one database before writing a query to another.

To control how the database engine performs locking during the transaction, use the isolation attribute.

Example

`<cftransaction>`
  `<cfquery name='makeNewCourse' datasource='Snippets'>`
      INSERT INTO Courses
        (Number, Descript)
      VALUES
        ('#myNumber#', '#myDescription#')
  `</cfquery>`
  `<cfquery name='insertNewCourseToList' datasource='Snippets'>`
      INSERT INTO CourseList
        (CorNumber, CorDesc, Dept_ID, CorName, CorLevel, LastUpdate)
      VALUES
        ('#myNumber#', '#myDescription#', '#myDepartment#', '#myDescription#', '#myCorLevel#', #Now()#)
  `</cfquery>`
`</cftransaction>`
cftree

Description

Inserts a tree control in a form. Validates user selections. Used within a cfform tag block. You can use a ColdFusion query to supply data to the tree.

Category

Forms tags

Syntax

```xml
<cftree name = "name"
  format="applet", "flash", xml, or "object"
  required = "yes" or "no"
  delimiter = "delimiter"
  completePath = "yes" or "no"
  appendKey = "yes" or "no"
  highlightHref = "yes" or "no"
  onValidate = "script_name"
  message = "text"
  onError = "text"
  lookAndFeel = "motif" or "windows" or "metal"
  font = "font"
  font = "font"
  fontSize = "size"
  italic = "yes" or "no"
  bold = "yes" or "no"
  height = "integer"
  width = "integer"
  vSpace = "integer"
  hSpace = "integer"
  align = "alignment"
  border = "yes" or "no"
  hScroll = "yes" or "no"
  vScroll = "yes" or "no"
  style= "style specification"
  enabled = "Yes" or "No"
  visible = "Yes" or "No"
  tooltip = "tip text"
  onChange = "ActionScript"
  notSupported = "text">
  </cftree>
```

See also

cfapplet, cfcalendar, cfform, cfformgroup, cfformitem, cfgrid, cfinput, cfselect, cfslider, cftextarea, cftreeitem; “Working with action pages” in Chapter 26, “Introduction to Retrieving and Formatting Data,” and “Building tree controls with the cftree tag” in Chapter 27, “Building Dynamic Forms with cfform Tags,” in ColdFusion MX Developer’s Guide
History
ColdFusion MX 7:
• Added the format attribute and support for generating Flash and XML and object output.
• Added enabled, onChange, style, tooltip, and visible attributes (Flash format only).
ColdFusion MX: Changed behavior: ColdFusion renders a tree control regardless of whether there are any treeitems within it.

Attributes

Note: In XML format, ColdFusion MX passes all attributes to the XML. The supplied XSLT skins do not handle or display XML format trees, but do display applet and Flash format trees.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Req/Opt Format</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Required; All</td>
<td></td>
<td>Name for tree control.</td>
</tr>
<tr>
<td>format</td>
<td>Optional; applet</td>
<td>All</td>
<td>• applet: displays the tree using a Java applet in the browser.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• flash: displays the tree using a Flash control</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• object: returns the tree as a ColdFusion structure with the name specified by the name attribute. For details of the structure contents, see &quot;object format&quot;.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• xml: generates an XML representation of the tree.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>In XML format forms, includes the generated XML in the form.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>In HTML format forms, puts the XML in a string variable with the name specified by the name attribute.</td>
</tr>
<tr>
<td>required</td>
<td>Optional; No</td>
<td></td>
<td>• Yes: user must select an item in the tree control.</td>
</tr>
<tr>
<td></td>
<td>Applet, Flash</td>
<td></td>
<td>• No</td>
</tr>
<tr>
<td>delimiter</td>
<td>Optional; \</td>
<td></td>
<td>Character to separate elements in the Forms.treename.path variable of the action page.</td>
</tr>
<tr>
<td>completePath</td>
<td>Optional; No</td>
<td></td>
<td>• Yes: starts the Form.treename.path variable with the root of the tree path when cffree is submitted.</td>
</tr>
<tr>
<td></td>
<td>Applet, object</td>
<td></td>
<td>• No: omits the root level from the Form.treename.path variable; the value starts with the first child node in the tree.</td>
</tr>
</tbody>
</table>

For the preserveData attribute of cfform to work with the tree, you must set this attribute to Yes.
For tree items populated by a query, if you use the cftreeitem queryasroot attribute to specify a root name, that value is returned. If you do not specify a root name, ColdFusion returns the query name.
### Attribute | Req/Opt | Default | Description
--- | --- | --- | ---
appendKey | Optional; All | Yes | • Yes: if you use `cftreeitem.href` attributes, ColdFusion appends a `CFTREEITEMKEY` query string variable with the value of the selected tree item to the `cfform.action` URL.
• No: does not append the tree item value to the URL.
highlightHref | Optional; Applet, Object | Yes | • Yes: highlights as a link the displayed value for any `cftreeitem` tag that specifies a `href` attribute.
• No: disables highlighting.
onValidate | Optional; Applet | JavaScript function to validate user input. The JavaScript DOM form object, input object, and input object value are passed to the specified routine, which should return True if validation succeeds; False, otherwise.
message | Optional; Applet | Message to display if validation fails.
onError | Optional; Applet | JavaScript function to execute if validation fails.
lookAndFeel | Optional; windows, object | windows | • motif: renders the slider in Motif style.
• windows: renders the slider in Windows style.
• metal: renders the slider in Java Swing style.
If the platform does not support a style option, the tag defaults to platform default style.
font | Optional; Applet | Font name for text in the tree control.
fontSize | Optional; Applet, Flash | | Font size for text in the tree control, in points.
italic | Optional; Applet, Flash | No | • Yes: displays tree control text in italics.
• No
bold | Optional; Applet, Flash | No | • Yes: displays tree control text in bold.
• No
height | Optional; Applet, Flash | 320 (applet only) | Tree control height, in pixels. If you omit this attribute in Flash format, Flash automatically sizes the tree.
width | Optional; Applet, Flash | 200 (applet only) | Tree control width, in pixels. If you omit this attribute in Flash format, Flash automatically sizes the tree.
vSpace | Optional; Applet | | Vertical margin above and below tree control, in pixels.
hSpace | Optional; Applet | | Horizontal spacing to left and right of tree control, in pixels.
<table>
<thead>
<tr>
<th>Attribute</th>
<th>Req/Opt Format</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
</table>
| align       | Optional; Applet, object | • top  
• left  
• bottom  
• baseline  
• texttop  
• absbottom  
• middle  
• absmiddle  
• right |  |
| border      | Optional; Yes | Yes: displays a border around the tree control.  
• No |  |
| hScroll     | Optional; Yes | Yes: permits horizontal scrolling.  
• No |  |
| vScroll     | Optional; Yes | Yes: permits vertical scrolling.  
• No |  |
| style       | Optional; Flash | Must be a style specification in CSS format, with the same syntax and contents as used in Macromedia Flex for the corresponding Flash element. |  |
| enabled     | Optional; Yes | Flash format only; Boolean value specifying whether the control is enabled. A disabled control appears in light gray. |  |
| visible     | Optional; Yes | Flash format only; Boolean value specifying whether to show the control. Space that would be occupied by an invisible control is blank. |  |
| tooltip     | Optional; Flash | Flash format only; Text to display when the mouse pointer hovers over the control. |  |
| onChange    | Optional; Flash | ActionScript to run when the control changes due to user action.  
If you specify an onChange event handler, the Form scope of the ColdFusion action page does not automatically get information about selected items. The ActionScript onChange event handler must handle all changes and selections. |  |
| notSupported | Optional; Applet | Text to display if a page that contains a Java applet-based cfform control is opened by a browser that does not support Java or has Java support disabled. For example:  
"<b>Browser must support Java to view ColdFusion Java Applets</b>"  
Default message:  
"<b>Browser must support Java to view ColdFusion Java Applets!</b>" |  |
Note: All attributes are passed to the XML generated in XML format, but no ColdFusion MX skin interprets cftree XML.

Usage

This tag must be in a cfform tag block.

The applet format tree requires the client to download a Java applet. Also, if the client does not have an up-to-date Java plugin installed, the system might also have to download an updated Java plugin to display an applet format tree. The Flash format tree uses a Flash control, and can be embedded in an HTML format cfform tag. For this tag to work properly in either Flash or applet format, the browser must also be JavaScript-enabled.

Note: If you specify Flash format for this tag in an HTML format form, and you do not specify height and width attributes, Flash takes up more than the remaining visible area on the screen. If any other output follows the tree, including any form controls, users must scroll to see it. Therefore, if you follow a Flash tree in an HTML form with additional output, specify height and width values.

If the following conditions are true, a user’s selection from query data that populates this tag’s options continues to display after the user submits the form:

• The cfform preserveData attribute is set to "Yes"
• The cfform action attribute posts to the same page as the form itself (this is the default), or the action page has a form that contains controls with the same names as corresponding controls on the user entry form

For more information, see the cfform tag entry.

Form variables

When you select a tree item and submit the form that contains the tree, ColdFusion creates a structure with two variables in the action page Form scope. The structure name is the tree name. The fields are as follows:

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>path</td>
<td>The path through the tree to the selected node, in the form [root]node_1\node_2... In applet format, the path includes the root node only if the completePath attribute is True. In Flash format, the path always includes the root node.</td>
</tr>
<tr>
<td>node</td>
<td>The value of the selected tree node.</td>
</tr>
</tbody>
</table>

object format

If you specify object in the format attribute, ColdFusion returns the tree as a ColdFusion structure, and does not send the tree to the browser. You can, for example, loop over the structure to populate a menu, generate "breadcrumb" links for page navigation, or create a dhtml tree.

Note: If you specify an object format tree in an XML format form, ColdFusion does not generate the tree.
The structure variable name is specified by the `cftree name` attribute. The top level of the structure has two types of entries:

- Attribute settings
- A children array

### Attribute settings

The structure has top-level entries with the values of the following `cftree` attributes:

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>align</td>
<td>completePath highlightHref lookAndFeel</td>
</tr>
<tr>
<td>appendKey</td>
<td>delimiter hScroll name</td>
</tr>
<tr>
<td>bold</td>
<td>fontWeight italic vscroll</td>
</tr>
<tr>
<td>border</td>
<td></td>
</tr>
</tbody>
</table>

### Children array

The top-level children entry is an array of items entries. Each item has the following entries:

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>children</td>
<td>This item’s child items; an array of item structures.</td>
</tr>
<tr>
<td>display</td>
<td>Tree item label, as determined by the <code>cftreeitem display</code> attribute.</td>
</tr>
<tr>
<td>expand</td>
<td>Whether to expand the item to display any children; value of <code>cftreeitem expand</code> attribute.</td>
</tr>
<tr>
<td>href</td>
<td>The URL to link to when the user selects the item; value of the <code>cftreeitem href</code> attribute.</td>
</tr>
<tr>
<td>img</td>
<td>The tree image icon Image to display as an icon for the tree item; value of <code>cftreeitem img</code> attribute.</td>
</tr>
<tr>
<td>imgOpen</td>
<td>Image to display when the tree item is open (expanded); value of <code>cftreeitem imgopen</code> attribute.</td>
</tr>
<tr>
<td>parent</td>
<td>Value of this item’s parent item in the tree.</td>
</tr>
<tr>
<td>path</td>
<td>The node path from the tree root to the current element.</td>
</tr>
<tr>
<td>queryAsRoot</td>
<td>Whether the query is the root of the item; value of <code>cftreeitem queryAsRoot</code> attribute.</td>
</tr>
<tr>
<td>target</td>
<td>The link target, such as _blank; value of the item’s <code>cftreeitem target</code> attribute.</td>
</tr>
<tr>
<td>value</td>
<td>The item’s value, as determined by the <code>cftreeitem value</code> attribute.</td>
</tr>
</tbody>
</table>

### Example

The following example creates a tree that shows available courses from the CourseList table of the `cfdocexamples` database, and puts each department’s courses in a folder. This example is displayed in Flash and uses the Departments list to get department names.

```cfquery name="getCourses" datasource="cfdocexamples">
  select d.dept_name, c.course_id, c.CorName, c.CorLevel, c.corName + ' (' + c.corLevel + ')' as corLabel
```

```
from CourseList c, Departments d
where d.Dept_ID = c.Dept_ID
order by d.dept_Name, c.corName, c.corLevel
</cfquery>

<cfform name="studentForm" format="flash" width="400" height="450">
  <cftree name="courseTree" width="350" height="400">
    <cftreeitem
      query="getCourses"
      value="dept_name,Course_id"
      display="dept_name,CorLabel"
      queryasroot="NO" expand="yes,no">
  </cftree>
</cfform>

The following example creates a tree that shows the basic information about all employees in an
organization, organized by department. The departments are expanded to show all employees.
You can click the + signs to display additional information. If you click the employee name,
ColdFusion links back to the same page and displays the Path and node values for the selection.

<!--- Query the datasource to get employee information. --->
<!--- Group the output by Department. (All fields are required in Group By clause.) --->
<cfquery name = "GetEmployees" dataSource = "cfdocexamples">
  SELECT Emp_ID, FirstName, LastName, EMail, Phone, Department
  FROM Employees
  GROUP BY Department, Emp_ID, FirstName, LastName, EMail, Phone
</cfquery>
<html>
<body>
<h3>cftree Example</h3>
<!--- The following runs if the user clicked on a link in the tree. A complete application would use the ID for additional processing. --->
<cfif isdefined("Form.fieldnames")>
  <b>Selected item information</b><br>
  <cfoutput>
    <b>Path: </b>#form.Employees.Path#<br>
    <b>node: </b>#form.Employees.node#<br>
  </cfoutput>
</cfif>
<!--- Display the tree. The cftree tag must be in a cfform. --->
<cfform action="#cgi.script_name#" preservedata="Yes" format="Flash">
  <cftree name = "Employees" height = "400" width = "400"
    font = "Arial Narrow" italic="yes" highlightref="No" HScroll="no"
    VScroll="no"
    completepath="no" lookandfeel="windows" border="No" required="yes">
    <!--- cfoutput tag with a group attribute loops over the departments. --->
    <cfoutput group="Department" query = "GetEmployees">
      <cftreeitem value="#Department#" parent="Employees" expand="yes">
      <!--- This cfoutput tag loops over the records for the department. The cfoutput tag does not need any attributes. --->
    </cfoutput>
  </cftree>
</cfform>
<!--- Create an item for each employee in the department. Do not expand children. Each employee name links to this page, and sends the employee ID in the query string.--->
<cftreeitem value="#LastName#, #FirstName#"
    parent="#Department#" expand="false" img="cd"
    href="#cgi.script_name#/user_id=#emp_id#">
  <!--- Each Employee entry has Id. and contact info children. --->
  <cftreeitem value="#Emp_ID#" display="Employee ID: #Emp_ID#"
      parent="#LastName#, #FirstName#" img="remote">
    <!--- Each node must be unique value, so use Emp_ID om value. --->
    <cftreeitem value="#Emp_ID#_ContactInfo" img="computer"
        display="Contact Information"
        parent="#LastName#, #FirstName#" expand="false">
      <!--- ContactInfo has two children --->
      <cftreeitem value="#Phone#" parent="#Emp_ID#_ContactInfo">
      <cftreeitem value="#Email#" parent="#Emp_ID#_ContactInfo">
    </cftreeitem>
  </cftreeitem>
</cftree>
</cfoutput>
</cfoutput>
</cftree>
</cfinput type="Submit" name="submitit" value="Submit" width="100">
</cfform>
cftreeitem

Description
Populates a form tree control, created with the cftree tag, with one or more elements.

Category
Forms tags

Syntax
```
<cftreeitem
   value = "text"
   display = "text"
   parent = "parent_name"
   img = "filename"
   imgopen = "filename"
   href = "URL"
   target = "URL_target"
   query = "queryname"
   queryAsRoot = "yes" or "no"
   expand = "yes" or "no">
</cftreeitem>
```

See also
ciafaplet, cfform, cfformgroup, cfformitem, cfgrid, cfinput, cfselect, cfslider, cftextarea, cftree; “Building tree controls with the cftree tag” in Chapter 27, “Building Dynamic Forms with cfform Tags,” in ColdFusion MX Developer’s Guide

Attributes
Note: In XML format, ColdFusion MX passes all attributes to the XML. The supplied XSLT skins do not handle or display XML format trees, but do display applet and Flash format trees.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Req/Opt; Format</th>
<th>Default Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>Required; All</td>
<td>Value passed when cfform is submitted. When populating a tree with data from a cfquery, you can specify multiple columns to use in a delimited list; for example, value = &quot;dept_id,emp_id&quot;. In this case, each column generates an item that is a child of the column that precedes it in the list.</td>
</tr>
<tr>
<td>display</td>
<td>Optional; value All</td>
<td>Tree item label. When populating a tree with data from a query, specify names in a delimited list. Example: display = &quot;dept_name,emp_name&quot;</td>
</tr>
<tr>
<td>parent</td>
<td>Optional; All</td>
<td>Value of the tree item parent. Determines the item’s placement in the tree hierarchy. If omitted, the item is placed at the tree root level, or if the queryAsRoot attribute is True, directly under the query.</td>
</tr>
<tr>
<td>Attribute</td>
<td>Req/Opt; Format</td>
<td>Default</td>
</tr>
<tr>
<td>-----------</td>
<td>----------------</td>
<td>---------</td>
</tr>
<tr>
<td>img</td>
<td>Optional; Applet, object</td>
<td>folder</td>
</tr>
<tr>
<td>imgopen</td>
<td>Optional; Applet, object</td>
<td></td>
</tr>
<tr>
<td>href</td>
<td>Optional; All</td>
<td></td>
</tr>
<tr>
<td>target</td>
<td>Optional; All</td>
<td></td>
</tr>
<tr>
<td>query</td>
<td>Optional; Applet, Flash</td>
<td></td>
</tr>
</tbody>
</table>
This tag requires the client to download a Java applet. Downloading an applet takes time; therefore, using this tag might be slightly slower than using an HTML form element or the cfinput tag to get the same information.

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

If you do not use a query to populate this tag, it creates a single tree item. If you do use a query, it creates multiple items; each row of the query creates a hierarchically nested set of items with one item per column.

Example

The following example creates a simple tree using a single cftreeitem tag and a query:

```cfc```<cfform action = "#cgi.script_name#">
   <cftree name = "Employees" height = "400" width = "200">
      <cftreeitem value="LastName, FirstName, Emp_ID" query="getEmployees"
       queryAsRoot="False">
      </cftree>
   </cftree>
</cfform>```

The following example creates a tree that shows the basic information about all employees in an organization, organized by department. The departments are expanded to show all employees. You can click the + signs to display additional information. If you click the employee name, ColdFusion links back to the same page and displays the selected employee's ID.

```
<!--- Query the datasource to get employee information.--->
<!--- Group the output by Department.  
(All fields are required in Group By clause.) --->
<cfquery name = "GetEmployees" dataSource = "cfdocexamples">
   SELECT Emp_ID, FirstName, LastName, EMaill, Phone, Department
   FROM Employees
   GROUP BY Department, Emp_ID, FirstName, LastName, EMaill, Phone
</cfquery>
```
<html><body><h3>cftreeitem Example</h3>

<!--- The following runs if the user clicked on a link in the tree. --->
<cfif isdefined("URL.user_ID")>
<cfoutput>
<!--- URL.cftreeitemkey is the selected tree item's value attribute. --->
You Requested information on #URL.cftreeitemKey#; User ID #URL.user_ID#
</cfoutput>
<br><br></cfif>

<!--- Display the tree. The cftree tag must be in a cfform. --->
<cfform>
<cftree name = "Employees" height = "400" width = "200"
        font = "Arial Narrow" highlightref="No" hscroll="No">
<!--- cfoutput tag with a group attribute loops over the departments. --->
<cfoutput group="Department" query = "GetEmployees">
<!--- This cfoutput tag loops over the records for the department. --->
<!--- The cfoutput tag does not need any attributes. --->
<cfoutput><cftreeitem value="#Department#" parent="Employees" expand="yes">
<!--- Each Employee entry has ID and ContactInfo children. --->
<cfoutput value="#Emp_ID#" display="#LastName#, #FirstName#"
         parent="#LastName#, #FirstName#" expand="no" href="#cgi.script_name#?user_id=#emp_id#">
<!--- ContactInfo has two children. --->
<cfoutput value="#Phone#" display="Contact Information"
         parent="#Emp_ID#_ContactInfo" expand="No">
<cfoutput value="#Email#" display="#Email#" parent="#Emp_ID#_ContactInfo"
</cfoutput>
</cfoutput></cfoutput>
</cftreeitem>
</cfform>
cftry

Description
Used with one or more cfcatch tags. Together, they catch and process exceptions in ColdFusion pages. Exceptions are events that disrupt the normal flow of instructions in a ColdFusion page, such as failed database operations, missing include files, and developer-specified events.

Category
Exception handling tags

Syntax
<cftry>
  Code that might throw an exception
  One or more cfcatch blocks
</cftry>

See also
cfcatch, cferror, cfrethrow, cfthrow, onError; Chapter 14, “Handling Errors” in ColdFusion MX Developer’s Guide

History
ColdFusion MX: Changed cfscript to include try and catch statements that are equivalent to the cftry and cfcatch tags.

Usage
Within a cftry block, put the code that might throw an exception, followed by one or more cfcatch tags that catch and process exceptions. This tag requires an end tag.

Example
<!--- cftry example, using TagContext to display the tag stack. --->
<h3>cftry Example</h3>
<!--- Open a cftry block. --->
<cftry>
  <!--- Note misspelled tablename "employees" as "employeeas". --->
  <cfquery name = "TestQuery" dataSource = "cfdocexamples">
      SELECT *
      FROM EMPLOYEEAS
  </cfquery>

  <!--- <p>... other processing goes here --->
  <!--- specify the type of error for which we search --->
  <cfcatch type = "Database">
      <!--- the message to display --->
      <h3>You've Thrown a Database <b>Error</b></h3>
      <cfoutput>
          <!--- and the diagnostic message from the ColdFusion server --->
          <p>#cfcatch.message#</p>
          <p>Caught an exception, type = #CFCATCH.TYPE#</p>
      </cfoutput>
      <cfloop index = i from = 1
          to = #ArrayLen(CFCATCH.TAGCONTEXT)#>
          <!--- the contents of the tag stack are: --->
          <p>#CFCATCH.TAGCONTEXT#</p>
  </cfcatch>
</cftry>

<!--- <p>... other processing goes here --->
<!--- specify the type of error for which we search --->
<cfcatch type = "Database">
  <!--- the message to display --->
  <h3>You've Thrown a Database <b>Error</b></h3>
  <cfoutput>
      <!--- and the diagnostic message from the ColdFusion server --->
      <p>#cfcatch.message#</p>
      <p>Caught an exception, type = #CFCATCH.TYPE#</p>
  </cfoutput>
  <cfloop index = i from = 1
      to = #ArrayLen(CFCATCH.TAGCONTEXT)#>
      <!--- the contents of the tag stack are: --->
      <p>#CFCATCH.TAGCONTEXT#</p>
  </cfcatch>
</cftry>

440 Chapter 2: ColdFusion Tags
<cfset sCurrent = #CFCATCH.TAGCONTEXT[i]#>
<cftry>
<br>#i# #sCurrent["ID"]#
(#sCurrent["LINE"]#: #sCurrent["COLUMN"]#:)
#sCurrent["TEMPLATE"]#
</cfloop>
</cfoutput>
</cfcatch>
</cftry>
**cfupdate**

**Description**

Updates records in a data source from data in a ColdFusion form or form Scope.

**Category**

Database manipulation tags

**Syntax**

```html
<cfupdate
  dataSource = "ds_name"
  tableName = "table_name"
  tableOwner = "name"
  tableQualifier = "qualifier"
  username = "username"
  password = "password"
  formFields = "field_names">
```

**See also**

cfinsert, cfproccparam, cfprocresult, cfquery, cfqueryparam, cfstoredproc, cftransaction; “Creating an update action page with cfupdate” in Chapter 21, “Updating Your Database,” in ColdFusion MX Developer’s Guide

**History**

ColdFusion MX: Deprecated the connectString, dbName, dbServer, dbtype, provider, and providerDSN attributes. They do not work, and might cause an error, in releases later than ColdFusion 5.

**Attributes**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Req/Opt</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dataSource</td>
<td>Required</td>
<td></td>
<td>Name of the data source that contains the table.</td>
</tr>
<tr>
<td>tableName</td>
<td>Required</td>
<td></td>
<td>Name of table to update.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• For ORACLE drivers, must be uppercase.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• For Sybase driver, case-sensitive; must be in same case as used when the table was created.</td>
</tr>
<tr>
<td>tableOwner</td>
<td>Optional</td>
<td></td>
<td>For data sources that support table ownership (for example, SQL Server, Oracle, Sybase SQL Anywhere), the table owner.</td>
</tr>
<tr>
<td>tableQualifier</td>
<td>Optional</td>
<td></td>
<td>For data sources that support table qualifiers. The purpose of table qualifiers is as follows:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• SQL Server and Oracle: name of the database that contains the table</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Intersolv dBASE driver: directory of DBF files</td>
</tr>
<tr>
<td>username</td>
<td>Optional</td>
<td></td>
<td>Overrides username value specified in ODBC setup.</td>
</tr>
</tbody>
</table>
Example

<!---- This example allows you to update a person's telephone number in the employee table. --->
<cfif isDefined("form.phone")>
   <cfupdate datasource="cfdocexamples" tablename="Employees">
</cfif>

<cfquery name="empTable" datasource="cfdocexamples">
   select * from employees
</cfquery>

<!---- This code shows the contents of the employee table and allows you to choose a row for updating. --->
<table border="1">
   <cfoutput query="empTable">
      <tr>
         <td>#firstName#</td>
         <td>#lastName#</td>
         <td>#phone#</td>
         <td><a href="cfupdate.cfm?id=#emp_id#">Edit</a></td>
      </tr>
   </cfoutput>
</table>

<cfif isDefined("url.id")>
   <cfquery name="phoneQuery" datasource="cfdocexamples">
      select * from employees where emp_id=#url.id#
   </cfquery>

   <!---- This code displays the row to edit for update. --->
   <cfoutput query="phoneQuery">
      <form action="cfupdate.cfm" method="post">
         #phoneQuery.firstName# #phoneQuery.lastName#
         <input name="phone" type="text" value="#phone#" size="12">
         <input type="submit" value="Update">
         <input name="emp_id" type="hidden" value="#emp_id#">
      </form>
   </cfoutput>
</cfif>

### Attribute | Req/Opt | Default | Description
---|---|---|---
password | Optional | | Overrides the password value specified in ODBC setup.
formFields | Optional | (all on form, except keys) | Comma-delimited list of form fields to update. If a form field is not matched by a column name in the database, ColdFusion throws an error. The formFields list must include the database table primary key field, which must be present in the form. It can be hidden.
cfwddx

Description

Serializes and deserializes CFML data structures to the XML-based WDDX format. The WDDX is an XML vocabulary for describing complex data structures in a standard, generic way. Implementing it lets you use the HTTP protocol to such information among application server platforms, application servers, and browsers.

This tag generates JavaScript statements to instantiate JavaScript objects equivalent to the contents of a WDDX packet or CFML data structure. Interoperates with Unicode.

Category

Extensibility tags

Syntax

<cfwddx
  action = "action"
  input = "inputdata"
  output = "result variable name"
  topLevelVariable = "top-level variable name for JavaScript"
  useTimeZoneInfo = "yes" or "no"
  validate = "yes" or "no" >

See also

cfcollection, cfdump, cfexecute, cfindex, cfobject, cfreport, cfsearch, ToScript;

History

ColdFusion MX

• Changed column name case behavior: ColdFusion preserves the case of column names in JavaScript. (Earlier releases converted query column names to lowercase.)

• Changed encoding format support: this tag supports several encoding formats. The default encoding format is UTF-8. The tag interoperates with Unicode.

Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Req/Opt</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>action</td>
<td>Required</td>
<td></td>
<td>• cfml2wddx: serializes CFML to WDDX.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• wddx2cfml: deserializes WDDX to CFML.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• cfml2js: serializes CFML to JavaScript.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• wddx2js: deserializes WDDX to JavaScript.</td>
</tr>
<tr>
<td>input</td>
<td>Required</td>
<td>A value to process.</td>
<td></td>
</tr>
<tr>
<td>output</td>
<td>Required if</td>
<td></td>
<td>Name of variable for output. If action = &quot;WDDX2JS&quot; or &quot;CFML2JS&quot;, and this attribute is omitted, result is output in HTML stream.</td>
</tr>
</tbody>
</table>
ColdFusion preserves the case of column names cases in JavaScript.

The wddx2js and cfml2js actions create a WddxRecordset javascript object when they encounter a RecordSet java object. The serialized JavaScript code requires a wddx.js file.

This tag performs the following conversions:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Req/Opt</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>topLevelVariable</td>
<td>Required if <code>action = &quot;wddx2js&quot;</code> or &quot;cfml2js&quot;</td>
<td>Name of top-level JavaScript object created by deserialization. The object is an instance of the WddxRecordset object.</td>
<td></td>
</tr>
<tr>
<td>useTimeZoneInfo</td>
<td>Optional</td>
<td>Yes</td>
<td>Whether to output time-zone information when serializing CFML to WDDX.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Yes: the hour-minute offset, represented in ISO8601 format, is output.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• No: the local time is output.</td>
</tr>
<tr>
<td>validate</td>
<td>Optional</td>
<td>No</td>
<td>Applies if <code>action = &quot;wddx2cfml&quot;</code> or &quot;wddx2js&quot;.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• yes: validates WDDX input with an XML parser using WDDX DTD. If parser processes input without error, packet is deserialized. Otherwise, an error is thrown.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• no: does not perform input validation.</td>
</tr>
</tbody>
</table>

Usage

The wddx2js and cfml2js actions create a WddxRecordset javascript object when they encounter a RecordSet java object. The serialized JavaScript code requires a wddx.js file.

For more information, and a list of the ColdFusion array and structure functions that you can use to manage XML document objects and functions, see Chapter 35, “Using XML and WDDX” in ColdFusion MX Developer’s Guide.

Note: The cfwddx tag throws an exception if you attempt to serialize a CFC or user-defined function (UDF).

Example

<!--- This example shows basic use of the cfwddx tag. --->
<html>
<body>
<!--- Create a simple query. --->
<cfquery name = "q" dataSource = "cfdocexamples">
  select Message_Id, Thread_id, Username from messages
</cfquery>

The recordset data is:...
<p>
<cfoutput query = "q">
  "topLevelVariable Required if action = "wddx2js" or "cfml2js"
  "useTimeZoneInfo Optional Yes Whether to output time-zone information when serializing CFML to WDDX.
  "validate Optional No Applies if action = "wddx2cfml" or "wddx2js".
</cfoutput>
</p>
</body>
</html>
<!--- Serialize data to WDDX format. --->
Serializing CFML data...
<cfwddx action = "cfml2wddx" input = #q# output = "wddxText">

<!--- Display WDDX XML packet. --->
Resulting WDDX packet is:
<xmp><cfoutput>#wddxText#</cfoutput></xmp>

<!--- Deserialize to a variable named wddxResult. --->
Deserializing WDDX packet...
<cfwddx action = "wddx2cfml" input = #wddxText# output = "qnew">

The recordset data is:...
<cfoutput query = qnew>
  #Message_ID# #Thread_ID# #Username#<br>
</cfoutput></p>
**cfxml**

**Description**

Creates a ColdFusion XML document object that contains the markup in the tag body. This tag can include XML and CFML tags. ColdFusion processes the CFML code in the tag body, and then assigns the resulting text to an XML document object variable, which is always stored in Unicode.

**Category**

Extensibility tags

**Syntax**

```
<CFXML
  variable="xmlVarName"
  caseSensitive="yes" or "no">
```

**See also**

IsXmlDoc, IsXmlElem, IsXmlRoot, ToString, XmlChildPos, XmlNew, XmlParse, XmlSearch, XmlTransform; Chapter 35, “Using XML and WDDX” in ColdFusion MX Developer’s Guide

**History**

ColdFusion MX 7: Added support for using an XML declaration at the start of the text.
ColdFusion MX: Added this tag.

**Attributes**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Req/Opt</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>variable</td>
<td></td>
<td></td>
<td>Name of the document object.</td>
</tr>
<tr>
<td>caseSensitive</td>
<td>Optional</td>
<td>no</td>
<td>• yes: maintains the case of document elements and attributes.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• no</td>
</tr>
</tbody>
</table>

**Usage**

If your XML object is case-sensitive, you cannot use dot notation to reference an element or attribute name. Use the name in associative array (bracket) notation, or a reference that does not use the case-sensitive name (such as xmlChildren[1]) instead. In the following code, the first line will work with a case-sensitive XML object. The second and third lines cause errors:

```
MyDoc.xmlRoot.XmlAttributes["Version"] = "12b";
MyDoc.xmlRoot.XmlAttributes.Version = "12b";
MyDoc.MyRoot.XmlAttributes["Version"] = "12b";
```

Use the XmlFormat function to escape special characters such as &, > and <.

To convert an XML document object back into a string, use the ToString function, at which time ColdFusion automatically prepends the <?xml version="1.0" encoding="UTF-8" ?> XML declaration.
To change the declaration to specify another encoding, use the Replace function. To specify the encoding of the text that is returned to a browser or other application, use the cfcontent tag.

The following example illustrates this process:

```coldfusion
<cfprocessingdirective suppresswhitespace="Yes">
<cfcontent type="text/xml; charset=utf-16">
<cfxml variable="xmlObject">
<breakfast_menu>
<food>
  <name quantity="50">Belgian Waffles</name>
  <description>Our famous Belgian Waffles</description>
</food>
</breakfast_menu>
</cfxml>
</cfprocessingdirective>
```

The cfprocessingdirective tag prevents ColdFusion from putting white space characters in front of the XML declaration.

Example

This following example creates a document object whose root element is MyDoc. The object includes text that displays the value of the ColdFusion variable testVar. The code creates four nested child elements, which are generated by an indexed cfloop tag. The cfdump tag displays the XML document object.

```coldfusion
<cfset testVar = True>
<cfxml variable="MyDoc">
  <!DOCTYPE MyDoc [>]
  <MyDoc>
    <cfif testVar IS True>
      <cfoutput>The value of testVar is True.</cfoutput>
    <cfelse>
      <cfoutput>The value of testVar is False.</cfoutput>
    </cfif>
    <cfloop index = "LoopCount" from = "1" to = "4">
      <childNode>
        This is Child node <cfoutput>#{LoopCount}.</cfoutput>
      </childNode>
    </cfloop>
  </MyDoc>
</cfxml>
<cfdump var="#MyDoc#">
```
This chapter lists and categorizes ColdFusion Markup Language (CFML) functions.

Contents

Function list ................................................................. 449
Functions by category ...................................................... 452
Function changes since ColdFusion 5 .................................. 457
Tag descriptions ............................................................... 461

Function list

ColdFusion Markup Language (CFML) includes a set of functions that you use in Macromedia ColdFusion MX 7 pages to perform logical and arithmetic operations and manipulate data.

The following table lists CFML functions:

<table>
<thead>
<tr>
<th>Function</th>
<th>Function</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abs</td>
<td>GetFunctionList</td>
<td>Max</td>
</tr>
<tr>
<td>ACos</td>
<td>GetGatewayHelper</td>
<td>Mid</td>
</tr>
<tr>
<td>AddSOAPRequestHeader</td>
<td>GetHttpRequestData</td>
<td>Min</td>
</tr>
<tr>
<td>AddSOAPResponseHeader</td>
<td>GetHttpTimeString</td>
<td>Minute</td>
</tr>
<tr>
<td>ArrayAppend</td>
<td>GetLocale</td>
<td>Month</td>
</tr>
<tr>
<td>ArrayAvg</td>
<td>GetLocaleDisplayName</td>
<td>MonthAsString</td>
</tr>
<tr>
<td>ArrayClear</td>
<td>GetMetaData</td>
<td>Now</td>
</tr>
<tr>
<td>ArrayDeleteAt</td>
<td>GetMetricData</td>
<td>NumberFormat</td>
</tr>
<tr>
<td>ArrayInsertAt</td>
<td>GetPageContext</td>
<td>ParagraphFormat</td>
</tr>
<tr>
<td>ArrayIsEmpty</td>
<td>GetProfileSections</td>
<td>ParseDateTime</td>
</tr>
<tr>
<td>ArrayLen</td>
<td>GetProfileString</td>
<td>Pi</td>
</tr>
<tr>
<td>ArrayMax</td>
<td>GetSOAPRequest</td>
<td>PreserveSingleQuotes</td>
</tr>
<tr>
<td>ArrayMin</td>
<td>GetSOAPRequestHeader</td>
<td>Quarter</td>
</tr>
<tr>
<td>ArrayNew</td>
<td>GetSOAPResponse</td>
<td>QueryAddColumn</td>
</tr>
<tr>
<td>Function</td>
<td>Function</td>
<td>Function</td>
</tr>
<tr>
<td>---------------------------------</td>
<td>---------------------------------</td>
<td>-------------------------------</td>
</tr>
<tr>
<td>ArrayPrepend</td>
<td>GetSOAPResponseHeader</td>
<td>QueryAddRow</td>
</tr>
<tr>
<td>ArrayResize</td>
<td>GetTempDirectory</td>
<td>QueryNew</td>
</tr>
<tr>
<td>ArraySet</td>
<td>GetTempFile</td>
<td>QuerySetCell</td>
</tr>
<tr>
<td>ArraySort</td>
<td>GetTickCount</td>
<td>QuotedValueList</td>
</tr>
<tr>
<td>ArraySum</td>
<td>GetTimeZoneInfo</td>
<td>Rand</td>
</tr>
<tr>
<td>ArraySwap</td>
<td>GetToken</td>
<td>Randomize</td>
</tr>
<tr>
<td>ArrayToList</td>
<td>Hash</td>
<td>RandRange</td>
</tr>
<tr>
<td>Asc</td>
<td>Hour</td>
<td>REFind</td>
</tr>
<tr>
<td>ASin</td>
<td>HTMLCodeFormat</td>
<td>REFindNoCase</td>
</tr>
<tr>
<td>Atan</td>
<td>HTMLEditFormat</td>
<td>ReleaseComObject</td>
</tr>
<tr>
<td>BinaryDecode</td>
<td>IIf</td>
<td>RemoveChars</td>
</tr>
<tr>
<td>BinaryEncode</td>
<td>IncrementValue</td>
<td>RepeatString</td>
</tr>
<tr>
<td>BitAnd</td>
<td>InputBaseN</td>
<td>Replace</td>
</tr>
<tr>
<td>BitMaskClear</td>
<td>Insert</td>
<td>ReplaceList</td>
</tr>
<tr>
<td>BitMaskRead</td>
<td>Int</td>
<td>ReplaceNoCase</td>
</tr>
<tr>
<td>BitMaskSet</td>
<td>IsArray</td>
<td>RREplace</td>
</tr>
<tr>
<td>BitNot</td>
<td>IsBinary</td>
<td>RREPLACENoCase</td>
</tr>
<tr>
<td>BitOr</td>
<td>IsBoolean</td>
<td>Reverse</td>
</tr>
<tr>
<td>BitSHLN</td>
<td>IsCustomFunction</td>
<td>Right</td>
</tr>
<tr>
<td>BitSHRN</td>
<td>IsDate</td>
<td>RJustify</td>
</tr>
<tr>
<td>BitXor</td>
<td>IsDebugMode</td>
<td>Round</td>
</tr>
<tr>
<td>Ceiling</td>
<td>IsDefined</td>
<td>RTRim</td>
</tr>
<tr>
<td>CharsetDecode</td>
<td>IsLeapYear</td>
<td>SendGatewayMessage</td>
</tr>
<tr>
<td>CharsetEncode</td>
<td>IsNumeric</td>
<td>SetEncoding</td>
</tr>
<tr>
<td>Chr</td>
<td>IsNumericDate</td>
<td>SetLocale</td>
</tr>
<tr>
<td>CJustify</td>
<td>IsObject</td>
<td>SetProfileString</td>
</tr>
<tr>
<td>Compare</td>
<td>IsQuery</td>
<td>SetVariable</td>
</tr>
<tr>
<td>CompareNoCase</td>
<td>IsSimpleValue</td>
<td>Sgn</td>
</tr>
<tr>
<td>Cos</td>
<td>IsSOAPRequest</td>
<td>Sin</td>
</tr>
<tr>
<td>CreateDate</td>
<td>IsStruct</td>
<td>SpanExcluding</td>
</tr>
<tr>
<td>CreateDateTime</td>
<td>IsUserInRole</td>
<td>SpanIncluding</td>
</tr>
<tr>
<td>CreateObject</td>
<td>IsValid</td>
<td>Sqr</td>
</tr>
<tr>
<td>CreateODBCDate</td>
<td>IsWDDX</td>
<td>StripCR</td>
</tr>
<tr>
<td>CreateODBCDateTime</td>
<td>IsXML</td>
<td>StructAppend</td>
</tr>
<tr>
<td>CreateOBCTime</td>
<td>IsXmlAttribute</td>
<td>StructClear</td>
</tr>
<tr>
<td>CreateTime</td>
<td>IsXMLDoc</td>
<td></td>
</tr>
<tr>
<td>Function</td>
<td>IsXmlElem</td>
<td>StructCopy</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>----------------------</td>
<td>--------------------</td>
</tr>
<tr>
<td>CreateTimeSpan</td>
<td>IsXmlNode</td>
<td>StructCount</td>
</tr>
<tr>
<td>CreateUUID</td>
<td>IsXmlRoot</td>
<td>StructDelete</td>
</tr>
<tr>
<td>DateAdd</td>
<td>JavaCast</td>
<td>StructFind</td>
</tr>
<tr>
<td>DateCompare</td>
<td>JSStringFormat</td>
<td>StructFindKey</td>
</tr>
<tr>
<td>DateConvert</td>
<td>LCase</td>
<td>StructFindValue</td>
</tr>
<tr>
<td>DateDiff</td>
<td>Left</td>
<td>StructGet</td>
</tr>
<tr>
<td>DateFormat</td>
<td>Len</td>
<td>StructInsert</td>
</tr>
<tr>
<td>Day</td>
<td>ListAppend</td>
<td>StructIsEmpty</td>
</tr>
<tr>
<td>DayOfWeek</td>
<td>ListChangeDelims</td>
<td>StructKeyArray</td>
</tr>
<tr>
<td>DayOfWeekAsString</td>
<td>ListContains</td>
<td>StructKeyExists</td>
</tr>
<tr>
<td>DayOfYear</td>
<td>ListContainsNoCase</td>
<td>StructKeyList</td>
</tr>
<tr>
<td>DaysInMonth</td>
<td>ListDeleteAt</td>
<td>StructNew</td>
</tr>
<tr>
<td>DaysInYear</td>
<td>ListFind</td>
<td>StructSort</td>
</tr>
<tr>
<td>DE</td>
<td>ListFindNoCase</td>
<td>StructUpdate</td>
</tr>
<tr>
<td>DecimalFormat</td>
<td>ListFirst</td>
<td>Tan</td>
</tr>
<tr>
<td>DecrementValue</td>
<td>ListGetAt</td>
<td>TimeFormat</td>
</tr>
<tr>
<td>Decrypt</td>
<td>ListInsertAt</td>
<td>ToBase64</td>
</tr>
<tr>
<td>DeleteClientVariable</td>
<td>ListLast</td>
<td>ToBinary</td>
</tr>
<tr>
<td>DirectoryExists</td>
<td>ListLen</td>
<td>ToScript</td>
</tr>
<tr>
<td>DollarFormat</td>
<td>ListPrepend</td>
<td>ToString</td>
</tr>
<tr>
<td>Duplicate</td>
<td>ListQualify</td>
<td>Trim</td>
</tr>
<tr>
<td>Encrypt</td>
<td>ListRest</td>
<td>UCase</td>
</tr>
<tr>
<td>Evaluate</td>
<td>ListSetAt</td>
<td>URLDecode</td>
</tr>
<tr>
<td>Exp</td>
<td>ListSort</td>
<td>URLEncodedFormat</td>
</tr>
<tr>
<td>ExpandPath</td>
<td>ListToArray</td>
<td>URLSessionFormat</td>
</tr>
<tr>
<td>FileExists</td>
<td>ListValueCount</td>
<td>Val</td>
</tr>
<tr>
<td>Find</td>
<td>ListValueCountNoCase</td>
<td>ValueList</td>
</tr>
<tr>
<td>FindNoCase</td>
<td>LJustify</td>
<td>Week</td>
</tr>
<tr>
<td>FindOneOf</td>
<td>Log</td>
<td>Wrap</td>
</tr>
<tr>
<td>FirstDayOfMonth</td>
<td>Log10</td>
<td>WriteOutput</td>
</tr>
<tr>
<td>Fix</td>
<td>LSCurrencyFormat</td>
<td>XmlChildPos</td>
</tr>
<tr>
<td>FormatBaseN</td>
<td>LSDateFormat</td>
<td>XmlElemNew</td>
</tr>
<tr>
<td>GetTempDirectory</td>
<td>LSEuroCurrencyFormat</td>
<td>XmlFormat</td>
</tr>
<tr>
<td>GetAuthUser</td>
<td>LSIsCurrency</td>
<td>XmlNodeType</td>
</tr>
<tr>
<td>GetBaseTagData</td>
<td>LSIsDate</td>
<td>XmlNew</td>
</tr>
</tbody>
</table>
Functions by category

The following tables list functions by their category or purpose.

Array functions ........................................................................................................... 452
Conversion functions ................................................................................................. 453
Conversion functions ................................................................................................. 453
Date and time functions ............................................................................................. 453
Decision functions ....................................................................................................... 453
Display and formatting functions ................................................................................ 454
Dynamic evaluation functions .................................................................................... 454
Extensibility functions ................................................................................................ 454
Full-text search functions ........................................................................................... 454
International functions ................................................................................................. 454
List functions ................................................................................................................ 454
Mathematical functions ............................................................................................... 455
Other functions ............................................................................................................ 455
Query functions ............................................................................................................ 455
String functions ............................................................................................................ 455
Structure functions ....................................................................................................... 456
System functions .......................................................................................................... 456
XML functions .............................................................................................................. 457

Array functions

<table>
<thead>
<tr>
<th>GetBaseTagList</th>
<th>LSIsNumeric</th>
<th>XmlParse</th>
</tr>
</thead>
<tbody>
<tr>
<td>GetBaseTemplatePath</td>
<td>LSNumberFormat</td>
<td>Xm1Search</td>
</tr>
<tr>
<td>GetClientVariablesList</td>
<td>LSParseCurrency</td>
<td>Xm1Transform</td>
</tr>
<tr>
<td>GetCurrentTemplatePath</td>
<td>LSParseDateTime</td>
<td>Xm1Validate</td>
</tr>
<tr>
<td>GetDirectoryFromPath</td>
<td>LSParseEuroCurrency</td>
<td>Year</td>
</tr>
<tr>
<td>GetEncoding</td>
<td>LSParseNumber</td>
<td>YesNoFormat</td>
</tr>
<tr>
<td>GetException</td>
<td>LSTimeFormat</td>
<td></td>
</tr>
<tr>
<td>GetFileFromPath</td>
<td>LTrim</td>
<td></td>
</tr>
</tbody>
</table>

ArrayAppend ArrayIsEmpty ArrayPrepend ArraySwap
ArrayAvg ArrayLen ArrayResize ArrayToList
ArrayClear ArrayMax ArraySet IsArray
### Functions by category

#### Conversion functions

<table>
<thead>
<tr>
<th>ArrayDeleteAt</th>
<th>ArrayMin</th>
<th>ArraySort</th>
<th>ListToArray</th>
</tr>
</thead>
<tbody>
<tr>
<td>ArrayInsertAt</td>
<td>ArrayNew</td>
<td>ArraySum</td>
<td></td>
</tr>
</tbody>
</table>

| ArrayToList | Hash | ToScript | XmlFormat |
| BinaryDecode | LCase | ToString | XmlParse |
| BinaryEncode | ListToArray | URLEncode | XmlTransform |
| CharSetDecode | ToBase64 | URLEncodedFormat | |
| CharSetEncode | ToBinary | Val | |

#### Date and time functions

| CreateDate | DateFormat | GetTimeZoneInfo | MonthAsString |
| CreateDateTime | DatePart | Hour | Now |
| CreateODBCDate | Day | IsDate | ParseDateTime |
| CreateODBCDateTime | DayOfWeek | IsLeapYear | Quarter |
| CreateODBCTime | DayOfWeekAsString | IsNumericDate | Second |
| CreateTime | DayOfYear | LSDateFormat | TimeFormat |
| createTimeSpan | DaysInMonth | LSIsDate | Week |
| DateAdd | DaysInYear | LSIsParseDateTime | Year |
| DateCompare | FirstDayOfMonth | LSTimeFormat | |
| DateConvert | GetHttpTimeString | Minute | |
| DateDiff | GetTickCount | Month | |

#### Decision functions

| DirectoryExists | IsDefined | IsSimpleValue | IsXmlNode |
| FileExists | IsK2ServerABroker | IsStruct | IsXmlRoot |
| IF | IsK2ServerDocCountExceeded | IsUserInRole | LSIsCurrency |
| IsArray | IsK2ServerOnline | IsValid | LSIsDate |
| IsBinary | IsLeapYear | IsWDDX | LSIsNumeric |
| IsBoolean | IsNumeric | XML | StructIsEmpty |
| IsCustomFunction | IsNumericDate | IsXmlAttribute | StructKeyExists |
| IsDate | IsObject | IsXmlDoc | YesNoFormat |
| IsDebugMode | IsQuery | IsXmlElem | |

---

*Functions by category* 453
### Display and formatting functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Function</th>
<th>Function</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>CJustify</td>
<td>HTMLCodeFormat</td>
<td>LSIsDate</td>
<td>NumberFormat</td>
</tr>
<tr>
<td>Date Format</td>
<td>HTMLEditFormat</td>
<td>LSNumberFormat</td>
<td>ParagraphFormat</td>
</tr>
<tr>
<td>Decimal Format</td>
<td>LJJustify</td>
<td>LSParseCurrency</td>
<td>RJustify</td>
</tr>
<tr>
<td>Dollar Format</td>
<td>LSCurrencyFormat</td>
<td>LSParseDateTime</td>
<td>StripCR</td>
</tr>
<tr>
<td>FormatBaseN</td>
<td>LSDateFormat</td>
<td>LSParseEuroCurrency</td>
<td>TimeFormat</td>
</tr>
<tr>
<td>GetLocale</td>
<td>LSEuroCurrencyFormat</td>
<td>LSParseNumber</td>
<td>YesNoFormat</td>
</tr>
<tr>
<td>GetLocaleDisplayName</td>
<td>LSIsCurrency</td>
<td>LSTimeFormat</td>
<td></td>
</tr>
</tbody>
</table>

### Dynamic evaluation functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>DE</td>
<td>Evaluate</td>
</tr>
</tbody>
</table>

### Extensibility functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Function</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>CreateObject</td>
<td>ReleaseComObject</td>
<td>ToScript</td>
</tr>
<tr>
<td>GetGatewayHelper</td>
<td>SendGatewayMessage</td>
<td></td>
</tr>
</tbody>
</table>

### Full-text search functions

#### History

ColdFusion MX 6.1: These functions are deprecated. They might not work, and might cause errors, in a future release.

<table>
<thead>
<tr>
<th>Function</th>
<th>Function</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>GetK2ServerDocCount</td>
<td>IsK2ServerABroker</td>
<td>IsK2ServerOnline</td>
</tr>
<tr>
<td>GetK2ServerDocCountLimit</td>
<td>IsK2ServerDocCountExceeded</td>
<td></td>
</tr>
</tbody>
</table>

### International functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Function</th>
<th>Function</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Date Convert</td>
<td>GetTimeZoneInfo</td>
<td>LSIsDate</td>
<td>LSParseEuroCurrency</td>
</tr>
<tr>
<td>GetEncoding</td>
<td>LSIsCurrency</td>
<td>LSParseDateTime</td>
<td>LSParseNumber</td>
</tr>
<tr>
<td>GetHttpTimeString</td>
<td>LSIsCurrencyFormat</td>
<td>LSEuroCurrencyFormat</td>
<td>LSTimeFormat</td>
</tr>
<tr>
<td>GetLocale</td>
<td>LSDateFormat</td>
<td>LSNumericUpDown</td>
<td></td>
</tr>
<tr>
<td>GetLocaleDisplayName</td>
<td>LSDateFormat</td>
<td>LSNumericUpDown</td>
<td></td>
</tr>
<tr>
<td>GetLocaleDisplayName</td>
<td>LSEuroCurrencyFormat</td>
<td>LSNumericUpDown</td>
<td></td>
</tr>
</tbody>
</table>

### List functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Function</th>
<th>Function</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>ArraySort</td>
<td>FindNoCase</td>
<td>ListContainsNoCase</td>
<td>ListQualify</td>
</tr>
<tr>
<td>ArrayToList</td>
<td>FindOneOf</td>
<td>ListDeleteAt</td>
<td>ListRest</td>
</tr>
<tr>
<td>Asc</td>
<td>FormatBaseN</td>
<td>ListFind</td>
<td>ListSetAt</td>
</tr>
<tr>
<td>Chr</td>
<td>GetClientVariablesList</td>
<td>ListFindNoCase</td>
<td>ListSort</td>
</tr>
<tr>
<td>CJustify</td>
<td>LCase</td>
<td>ListFirst</td>
<td>ListToArray</td>
</tr>
</tbody>
</table>
**Mathematical functions**

<table>
<thead>
<tr>
<th>Abs</th>
<th>BitNot</th>
<th>FormatBaseN</th>
<th>Randomize</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACos</td>
<td>BitOr</td>
<td>IncrementValue</td>
<td>RandRange</td>
</tr>
<tr>
<td>ArrayAvg</td>
<td>BitSHLN</td>
<td>InputBaseN</td>
<td>Round</td>
</tr>
<tr>
<td>ArraySum</td>
<td>BitSHRN</td>
<td>Int</td>
<td>Sgn</td>
</tr>
<tr>
<td>ASin</td>
<td>BitXor</td>
<td>Log</td>
<td>Sin</td>
</tr>
<tr>
<td>Atan</td>
<td>Ceiling</td>
<td>Log10</td>
<td>Sqr</td>
</tr>
<tr>
<td>BitAnd</td>
<td>Cos</td>
<td>Max</td>
<td>Tan</td>
</tr>
<tr>
<td>BitMaskClear</td>
<td>DecrementValue</td>
<td>Min</td>
<td></td>
</tr>
<tr>
<td>BitMaskRead</td>
<td>Exp</td>
<td>Pi</td>
<td></td>
</tr>
<tr>
<td>BitMaskSet</td>
<td>Fix</td>
<td>Rand</td>
<td></td>
</tr>
</tbody>
</table>

**Other functions**

<table>
<thead>
<tr>
<th>CreateUUID</th>
<th>GetBaseTagList</th>
<th>PreserveSingleQuotes</th>
</tr>
</thead>
<tbody>
<tr>
<td>DeleteClientVariable</td>
<td>GetBaseTemplatePath</td>
<td>URLSessionFormat</td>
</tr>
<tr>
<td>GetBaseTagData</td>
<td>GetClientVariablesList</td>
<td>WriteOutput</td>
</tr>
</tbody>
</table>

**Query functions**

<table>
<thead>
<tr>
<th>IsQuery</th>
<th>QueryAddRow</th>
<th>QuerySetCell</th>
<th>ValueList</th>
</tr>
</thead>
<tbody>
<tr>
<td>QueryAddColumn</td>
<td>QueryNew</td>
<td>QuotedValueList</td>
<td></td>
</tr>
</tbody>
</table>

**Security functions**

<table>
<thead>
<tr>
<th>Decrypt</th>
<th>GetAuthUser</th>
<th>GetTempDirectory</th>
<th>IsUserInRole</th>
</tr>
</thead>
<tbody>
<tr>
<td>Encrypt</td>
<td>GenerateSecretKey</td>
<td>Hash</td>
<td></td>
</tr>
</tbody>
</table>

**String functions**

**History**

ColdFusion MX: ColdFusion now supports the Java UCS-2 representation of Unicode character values 0–65535. (Earlier releases supported ASCII values.)
String-processing functions process any of these characters (including ASCII 0 (NUL) characters), and continue counting subsequent characters of the string, if any. (In earlier releases, some string-processing functions processed the ASCII 0 (NUL) character, but did not process subsequent characters of the string.)

<table>
<thead>
<tr>
<th>Asc</th>
<th>GetToken</th>
<th>LSParseDateTime</th>
<th>Reverse</th>
</tr>
</thead>
<tbody>
<tr>
<td>BinaryDecode</td>
<td>Hash</td>
<td>LSParseEuroCurrency</td>
<td>Right</td>
</tr>
<tr>
<td>BinaryEncode</td>
<td>HTMLCodeFormat</td>
<td>LSParseNumber</td>
<td>RJustify</td>
</tr>
<tr>
<td>CharsetDecode</td>
<td>HTMLEditFormat</td>
<td>LTrim</td>
<td>RTrim</td>
</tr>
<tr>
<td>CharSetEncode</td>
<td>Insert</td>
<td>Mid</td>
<td>SpanExcluding</td>
</tr>
<tr>
<td>Chr</td>
<td>JavaCast</td>
<td>MonthAsString</td>
<td>SpanIncluding</td>
</tr>
<tr>
<td>CJustify</td>
<td>JSGetStringFormat</td>
<td>ParagraphFormat</td>
<td>StripCR</td>
</tr>
<tr>
<td>Compare</td>
<td>LCase</td>
<td>ParseDateTime</td>
<td>ToBase64</td>
</tr>
<tr>
<td>CompareNoCase</td>
<td>Left</td>
<td>RFind</td>
<td>ToBinary</td>
</tr>
<tr>
<td>DayOfWeekAsString</td>
<td>Len</td>
<td>RFindNoCase</td>
<td>ToString</td>
</tr>
<tr>
<td>Decrypt</td>
<td>LJustify</td>
<td>RemoveChars</td>
<td>Trim</td>
</tr>
<tr>
<td>Encrypt</td>
<td>ListValueCount</td>
<td>RepeatString</td>
<td>UCase</td>
</tr>
<tr>
<td>Decrypt</td>
<td>ListValueCountNoCase</td>
<td>Replace</td>
<td>URLDecode</td>
</tr>
<tr>
<td>Decrypt</td>
<td>LSIsDate</td>
<td>ReplaceNoCase</td>
<td>URLEncodedFormat</td>
</tr>
<tr>
<td>Decrypt</td>
<td>LSIsNumeric</td>
<td>REReplace</td>
<td>Val</td>
</tr>
<tr>
<td>FormatBaseN</td>
<td>LSParseCurrency</td>
<td>REReplaceNoCase</td>
<td>Wrap</td>
</tr>
<tr>
<td>GenerateSecretKey</td>
<td>LSIsCurrency</td>
<td>ReplaceList</td>
<td>XMLFormat</td>
</tr>
</tbody>
</table>

See also “Conversion functions” on page 453.

**Structure functions**

| Duplicate | StructCount | StructGet | StructKeyList |
| IsStruct | StructDelete | StructInsert | StructNew |
| StructAppend | StructFind | StructIsEmpty | StructSort |
| StructClear | StructFindKey | StructKeyArray | StructUpdate |
| StructCopy | StructFindValue | StructKeyExists |

**System functions**

| DirectoryExists | GetFileFromPath | GetTempDirectory |
| Duplicate | GetFunctionList | GetTempFile |
| ExpandPath | GetHttpRequestData | GetTemplatePath |
| FileExists | GetLocale | GetTickCount |
| GetBaseTemplatePath | GetLocaleDisplayName | SetEncoding |
Function changes since ColdFusion 5

The following tables list functions, parameters and values that have changed since ColdFusion 5.0 and indicate the specific release in which the change was made.

New functions, parameters, and values .................................................. 457
Deprecated functions, parameters, and values .............................. 459
Obsolete functions, parameters, and values ...................................... 460

New functions, parameters, and values

<table>
<thead>
<tr>
<th>Function</th>
<th>Parameter or value</th>
<th>Added in this ColdFusion release</th>
</tr>
</thead>
<tbody>
<tr>
<td>BinaryDecode</td>
<td>All</td>
<td>ColdFusion MX 7</td>
</tr>
<tr>
<td>BinaryEncode</td>
<td>All</td>
<td>ColdFusion MX 7</td>
</tr>
<tr>
<td>CharSetDecode</td>
<td>All</td>
<td>ColdFusion MX 7</td>
</tr>
<tr>
<td>CharSetEncode</td>
<td>All</td>
<td>ColdFusion MX 7</td>
</tr>
<tr>
<td>CreateObject</td>
<td>portName parameter</td>
<td>ColdFusion MX 7</td>
</tr>
<tr>
<td></td>
<td>All</td>
<td>ColdFusion MX</td>
</tr>
<tr>
<td>DateAdd</td>
<td>datepart parameter</td>
<td>ColdFusion MX 6.1</td>
</tr>
<tr>
<td>DatePart</td>
<td>datepart parameter</td>
<td>ColdFusion MX 6.1</td>
</tr>
<tr>
<td>Decrypt</td>
<td>algorithm and encoding parameters</td>
<td>ColdFusion MX 7</td>
</tr>
<tr>
<td>Encrypt</td>
<td>algorithm and encoding parameters</td>
<td>ColdFusion MX 7</td>
</tr>
<tr>
<td>GenerateSecretKey</td>
<td>All</td>
<td>ColdFusion MX 7</td>
</tr>
<tr>
<td>GetGatewayHelper</td>
<td>All</td>
<td>ColdFusion MX 7</td>
</tr>
</tbody>
</table>

XML functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Parameter or value</th>
<th>Added in this ColdFusion release</th>
</tr>
</thead>
<tbody>
<tr>
<td>AddSOAPRequestHeader</td>
<td>IsSOAPRequest</td>
<td>XmlGetNodeType</td>
</tr>
<tr>
<td>AddSOAPResponseHeader</td>
<td>IsXML</td>
<td>IsWDDX</td>
</tr>
<tr>
<td>GetSOAPRequest</td>
<td>IsXmlAttribute</td>
<td>ToString</td>
</tr>
<tr>
<td>GetSOAPRequestHeader</td>
<td>IsXmlDoc</td>
<td>XmlChildPos</td>
</tr>
<tr>
<td>GetSOAPResponse</td>
<td>IsXmlElement</td>
<td>XMLElemNew</td>
</tr>
<tr>
<td>GetSOAPResponseHeader</td>
<td>IsXmlNode</td>
<td>XmlFormat</td>
</tr>
<tr>
<td></td>
<td></td>
<td>XmlValidate</td>
</tr>
</tbody>
</table>

Function changes since ColdFusion 5
<table>
<thead>
<tr>
<th>Function</th>
<th>Parameter or value</th>
<th>Added in this ColdFusion release</th>
</tr>
</thead>
<tbody>
<tr>
<td>GetAuthUser</td>
<td>All</td>
<td>ColdFusion MX</td>
</tr>
<tr>
<td>GetContextRoot</td>
<td>All</td>
<td>ColdFusion MX 7</td>
</tr>
<tr>
<td>GetEncoding</td>
<td>All</td>
<td>ColdFusion MX</td>
</tr>
<tr>
<td>GetLocaleDisplayName</td>
<td></td>
<td>ColdFusion MX 7</td>
</tr>
<tr>
<td>GetMetaData</td>
<td>All</td>
<td>ColdFusion MX</td>
</tr>
<tr>
<td>GetPageContext</td>
<td>All</td>
<td>ColdFusion MX</td>
</tr>
<tr>
<td>GetProfileSections</td>
<td>All</td>
<td>ColdFusion MX</td>
</tr>
<tr>
<td>GetSOAPRequest</td>
<td>All</td>
<td>ColdFusion MX 7</td>
</tr>
<tr>
<td>GetSOAPRequestHeader</td>
<td>All</td>
<td>ColdFusion MX 7</td>
</tr>
<tr>
<td>GetSOAPResponse</td>
<td>All</td>
<td>ColdFusion MX 7</td>
</tr>
<tr>
<td>GetSOAPResponseHeader</td>
<td>All</td>
<td>ColdFusion MX 7</td>
</tr>
<tr>
<td>Hash</td>
<td>algorithm and encoding parameters</td>
<td>ColdFusion MX 7</td>
</tr>
<tr>
<td>IsObject</td>
<td>All</td>
<td>ColdFusion MX</td>
</tr>
<tr>
<td>IsSOAPRequest</td>
<td>All</td>
<td>ColdFusion MX 7</td>
</tr>
<tr>
<td>IsUserInRole</td>
<td>All</td>
<td>ColdFusion MX</td>
</tr>
<tr>
<td>IsValid</td>
<td>All</td>
<td>ColdFusion MX 7</td>
</tr>
<tr>
<td>IsXML</td>
<td>All</td>
<td>ColdFusion MX 7</td>
</tr>
<tr>
<td>IsXmlAttribute</td>
<td>All</td>
<td>ColdFusion MX 7</td>
</tr>
<tr>
<td>IsXmlNode</td>
<td>All</td>
<td>ColdFusion MX 7</td>
</tr>
<tr>
<td>IsXmlRoot</td>
<td>All</td>
<td>ColdFusion MX 7</td>
</tr>
<tr>
<td>LSTimeFormat</td>
<td>key of mask parameter</td>
<td>ColdFusion MX 6.1</td>
</tr>
<tr>
<td>QueryAddColumn</td>
<td>datatype parameter</td>
<td>ColdFusion MX 7</td>
</tr>
<tr>
<td>QueryNew</td>
<td>columntolist parameter</td>
<td>ColdFusion MX 7</td>
</tr>
<tr>
<td>Rand</td>
<td>algorithm parameter</td>
<td>ColdFusion MX 7</td>
</tr>
<tr>
<td>Randomize</td>
<td>algorithm parameter</td>
<td>ColdFusion MX 7</td>
</tr>
<tr>
<td>RandRange</td>
<td>algorithm parameter</td>
<td>ColdFusion MX 7</td>
</tr>
<tr>
<td>ReleaseComObject</td>
<td>All</td>
<td>ColdFusion MX 6.1</td>
</tr>
<tr>
<td>SendGatewayMessage</td>
<td>All</td>
<td>ColdFusion MX 7</td>
</tr>
<tr>
<td>SetEncoding</td>
<td>All</td>
<td>ColdFusion MX</td>
</tr>
<tr>
<td>TimeFormat</td>
<td>key of mask parameter</td>
<td>ColdFusion MX 6.1</td>
</tr>
</tbody>
</table>
### Deprecated functions, parameters, and values

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

<table>
<thead>
<tr>
<th>Function</th>
<th>Parameter or value</th>
<th>Deprecated as of this ColdFusion release</th>
</tr>
</thead>
<tbody>
<tr>
<td>ToScript</td>
<td>All</td>
<td>ColdFusion MX 7</td>
</tr>
<tr>
<td>URLDecode</td>
<td>charset parameter</td>
<td>ColdFusion MX</td>
</tr>
<tr>
<td>URLEncodedFormat</td>
<td>charset parameter</td>
<td>ColdFusion MX</td>
</tr>
<tr>
<td>URLSessionFormat</td>
<td>All</td>
<td>ColdFusion MX</td>
</tr>
<tr>
<td>Wrap</td>
<td>All</td>
<td>ColdFusion MX 6.1</td>
</tr>
<tr>
<td>XmlChildPos</td>
<td>All</td>
<td>ColdFusion MX</td>
</tr>
<tr>
<td>XmlElemNew</td>
<td>All</td>
<td>ColdFusion MX</td>
</tr>
<tr>
<td>XmlElemNew</td>
<td>namespace parameter</td>
<td>ColdFusion MX 7</td>
</tr>
<tr>
<td>XmlGetNodeType</td>
<td>All</td>
<td>ColdFusion MX 7</td>
</tr>
<tr>
<td>XmlNew</td>
<td>All</td>
<td>ColdFusion MX</td>
</tr>
<tr>
<td>XmlParse</td>
<td>All</td>
<td>ColdFusion MX</td>
</tr>
<tr>
<td>XmlParse</td>
<td>validator parameter</td>
<td>ColdFusion MX 7</td>
</tr>
<tr>
<td>XmlSearch</td>
<td>All</td>
<td>ColdFusion MX</td>
</tr>
<tr>
<td>XmlTransform</td>
<td>All</td>
<td>ColdFusion MX</td>
</tr>
<tr>
<td>XmlTransform</td>
<td>parameters parameter</td>
<td>ColdFusion MX 7</td>
</tr>
<tr>
<td>XmlValidate</td>
<td>All</td>
<td>ColdFusion MX 7</td>
</tr>
</tbody>
</table>

---

### Deprecated functions, parameters, and values

<table>
<thead>
<tr>
<th>Function</th>
<th>Parameter or value</th>
<th>Deprecated as of this ColdFusion release</th>
</tr>
</thead>
<tbody>
<tr>
<td>GetMetricData</td>
<td>cachepops parameter</td>
<td>ColdFusion MX</td>
</tr>
<tr>
<td>GetK2ServerDocCount</td>
<td>All</td>
<td>ColdFusion MX 6.1</td>
</tr>
<tr>
<td>GetK2ServerDocCount</td>
<td>All</td>
<td>ColdFusion MX 6.1</td>
</tr>
<tr>
<td>GetK2ServerDocCount</td>
<td>Limit</td>
<td>ColdFusion MX</td>
</tr>
<tr>
<td>GetTemplatePath</td>
<td>All</td>
<td>ColdFusion MX</td>
</tr>
<tr>
<td>IsK2ServerABroker</td>
<td>All</td>
<td>ColdFusion MX 6.1</td>
</tr>
<tr>
<td>IsK2ServerDocCount</td>
<td>Exceeded</td>
<td>ColdFusion MX 6.1</td>
</tr>
<tr>
<td>IsK2ServerOnLine</td>
<td>All</td>
<td>ColdFusion MX 6.1</td>
</tr>
<tr>
<td>ParameterExists</td>
<td>All</td>
<td>ColdFusion MX. Use the IsDefined function.</td>
</tr>
<tr>
<td>SetLocale</td>
<td>locale = &quot;Spanish (Mexican)&quot; value</td>
<td>ColdFusion MX. Use Spanish (Standard).</td>
</tr>
</tbody>
</table>
Obsolete functions, parameters, and values

The following functions, parameters, and values are obsolete. Do not use them in ColdFusion applications. They do not work in releases later than ColdFusion 5.

<table>
<thead>
<tr>
<th>Function</th>
<th>Parameter or value</th>
<th>Obsolete as of this ColdFusion release</th>
</tr>
</thead>
<tbody>
<tr>
<td>AuthenticatedContext</td>
<td>All</td>
<td>ColdFusion MX</td>
</tr>
<tr>
<td>AuthenticatedUser</td>
<td>All</td>
<td>ColdFusion MX</td>
</tr>
<tr>
<td>isAuthenticated</td>
<td>All</td>
<td>ColdFusion MX</td>
</tr>
<tr>
<td>isAuthorized</td>
<td>All</td>
<td>ColdFusion MX</td>
</tr>
<tr>
<td>isProtected</td>
<td>All</td>
<td>ColdFusion MX</td>
</tr>
</tbody>
</table>
Abs

Description
Absolute-value function. The absolute value of a number is the number without its sign.

Returns
The absolute value of a number.

Category
Mathematical functions

Function syntax
Abs(number)

See also
Sgn

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>number</td>
<td>A number</td>
</tr>
</tbody>
</table>

Example
<h3>Abs Example</h3>
<p>The absolute value of the following numbers:
1,3,-4,-3.2,6 is
</p><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.</p>
ACos

Description

Arccosine function. The arccosine is the angle whose cosine is number.

Returns

The arccosine, in radians, of a number.

Category

Mathematical functions

Function syntax

ACos(number)

See also

Cos, Sin, ASin, Tan, Atn, Pi

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>number</td>
<td>Cosine of an angle. The value must be between -1.0 and 1.0, inclusive.</td>
</tr>
</tbody>
</table>

Usage

The range of the result is 0 to π.

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

Example

```html
<h3>ACos Example</h3>
<!--- output its arccosine value --->
<cfif IsDefined("FORM.CosNum")>
  <cfif IsNumeric(FORM.CosNum)>
    <cfif Abs(FORM.CosNum) LESS THAN OR EQUAL TO 1>
      <cfoutput>ACos(#FORM.CosNum#) = #ACos(FORM.cosNum)# Radians</cfoutput>
      <br>or<br>
      <cfoutput>ACos(#FORM.CosNum#) = #ACos(FORM.cosNum) * 180/PI()#</cfoutput>
    </cfif>
  </cfif>
</cfif>

<!--- if it is empty, output an error message --->
<h4>Enter a number between -1 and 1</h4>
</cfif>

<form method="post" action = "acos.cfm">
  <p>Enter a number to get its arccosine in Radians and Degrees.</p>
  <br><input type = "Text" name = "cosNum" size = "25">
  <p><input type = "Submit" name = ""> <input type = "RESET"></p>
</form>
```
AddSOAPRequestHeader

Description
Adds a SOAP header to a web service request before making the request.

Returns
Nothing.

Category
XML functions

History
ColdFusion MX 7: Added this function.

Function syntax
AddSOAPRequestHeader(webservice, namespace, name, value [, mustunderstand])

See also

Parameters
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>webservice</td>
<td>A web service object as returned from the cfobject tag or the CreateObject function.</td>
</tr>
<tr>
<td>namespace</td>
<td>A string that is the namespace for the header.</td>
</tr>
<tr>
<td>name</td>
<td>A string that contains the name of the SOAP header in the request.</td>
</tr>
<tr>
<td>value</td>
<td>The value for the SOAP header; this can be a CFML XML value.</td>
</tr>
<tr>
<td>mustunderstand</td>
<td>Optional. True or False (default). Sets the SOAP mustunderstand value for this header.</td>
</tr>
</tbody>
</table>

Usage
Used within CFML code by a consumer of a web service before it calls the web service.

If you pass XML in the value parameter, ColdFusion ignores the namespace and name parameters. If you require a namespace, define it within the XML itself.

Example
There are two parts to this example. The first part is the web service CFC that this function (as well as the other ColdFusion SOAP functions) uses to demonstrate its interaction with a web service. To implement the web service for this function, see the example for AddSOAPResponseHeader.

Execute the following example as a client to see how the AddSOAPRequestHeader function operates.
<!--- Note that you might need to modify the URL in the CreateObject function to match your server and the location of the headerservice.cfc file if it is different than shown here. Likewise for the cfinvoke tag at the end. --->

<h3>AddSOAPRequestHeader Example</h3>
<cfscript>
// Create the web service object.
ws = CreateObject("webservice", "http://localhost/soapheaders/
headerservice.cfc?WSDL");

// Set the username header as a string.
addSOAPRequestHeader(ws, "http://mynamespace/", "username", "tom", false);

// Set the password header as a CFML XML object.
doc = XmlNew();
doc.password = XmlElemNew(doc, "http://mynamespace/", "password");
doc.password.XmlText = "My Voice is my Password";
doc.password.XmlAttributes["xsi:type"] = "xsd:string";
addSOAPRequestHeader(ws, "ignoredNameSpace", "ignoredName", doc);

// Invoke the web service operation.
ret = ws.echo_me("argument");

// Get the first header as an object (string) and as XML.
header = getSOAPResponseHeader(ws, "http://www.tomj.org/myns",
"returnheader");
XMLheader = getSOAPResponseHeader(ws, "http://www.tomj.org/myns",
"returnheader", true);

// Get the second header as an object (string) and as XML.
header2 =  getSOAPResponseHeader(ws, "http://www.tomj.org/myns",
"returnheader2");
XMLheader2 = getSOAPResponseHeader(ws, "http://www.tomj.org/myns",
"returnheader2", true);
</cfscript>

<cfoutput>
Soap Header value: #HTMLCodeFormat(header)#
Soap Header XML value: #HTMLCodeFormat(XMLheader)#
Soap Header 2 value: #HTMLCodeFormat(header2)#
Soap Header 2 XML value: #HTMLCodeFormat(XMLheader2)#
Return value: #HTMLCodeFormat(ret)#
</cfoutput>

<!--- Note that you might need to modify the URL in the CreateObject function to match your server and the location of the headerservice.cfc file if it is different than shown here. Likewise for the cfinvoke tag at the end. --->

<cfinvoke component="soapheaders.headerservice" method="echo_me"
returnvariable="ret" in_here="hi">
</cfinvoke>
<cfoutput>Cfinvoke returned: #ret#</cfoutput>
AddSOAPResponseHeader

Description

Adds a SOAP response header to a web service response. Call only from within a CFC web service function that is processing a request as a SOAP web service.

Returns

Nothing

Category

XML functions

History

ColdFusion MX 7: Added this function.

Function syntax

AddSOAPResponseHeader(namespace, name, value [, mustunderstand])

See also


Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>namespace</td>
<td>A string that is the namespace for the header.</td>
</tr>
<tr>
<td>name</td>
<td>A string that contains the name of the SOAP header in the request.</td>
</tr>
<tr>
<td>value</td>
<td>The value for the SOAP header; this can be a CFML XML value.</td>
</tr>
<tr>
<td>mustunderstand</td>
<td>Optional. True or False (default). Sets the SOAP mustunderstand value for this header.</td>
</tr>
</tbody>
</table>

Usage

Call this function only from within a CFC web service function. It throws an error if it is invoked in a context that is not a web service request.

If you pass XML in the value parameter, ColdFusion ignores the namespace and name parameters. If you require a namespace, define it within the XML itself.

Use the IsSOAPRequest function to determine if the CFC is running as a web service.

Example

This example creates a CFC web service that illustrates the operation of the AddSOAPResponseHeader function and also provides a web service that illustrates the operation of other ColdFusion SOAP functions.
Save the following code as headerservice.cfc in a folder called soapheaders under your web root. Test its operation, and specifically the operation of the AddSOAPResponseHeader function, by executing the examples that invoke this web service. For example, see the example for AddSOAPRequestHeader.

AddSOAPRequestHeader Example
<!--- The headerservice.cfc CFC Web Service.--->
<cfcomponent displayName="tester" hint="Test for SOAP headers">
<cffunction name="echo_me" access="remote" output="false" returntype="string" displayname="Echo Test" hint="Header test">
  <cfargument name="in_here" required="true" type="string">
  <cfif isSOAPRequest()>
    <!--- Get the first header as a string and as XML. --->
    <cfset username = getSOAPRequestHeader("http://mynamespace/", "username")>
    <cfset return = "The service saw username: " & username>
    <cfset xmlusername = getSOAPRequestHeader("http://mynamespace/", "username", "TRUE")>
    <cfset return = return & "<br> as XML: " & xmlusername>
    <!--- Get the second header as a string and as XML. --->
    <cfset password = getSOAPRequestHeader("http://mynamespace/", "password")>
    <cfset return = return & "The service saw password: " & password>
    <cfset xmlpassword = getSOAPRequestHeader("http://mynamespace/", "password", "TRUE")>
    <cfset return = return & "<br> as XML: " & xmlpassword>
    <!--- Add a header as a string. --->
    <cfset addSOAPResponseHeader("http://www.tomj.org/myns", "returnheader", "AUTHORIZED VALUE", false)>
    <!--- Add a second header using a CFML XML value. --->
    <cfset doc = XmlNew()>
    <cfset x = XmlElemNew(doc, "http://www.tomj.org/myns", "returnheader2")>
    <cfset x.XmlText = "hey man, here I am in XML">
    <cfset x.XmlAttributes[xsi:type] = "xsd:string">
    <cfset tmp = addSOAPResponseHeader("ignoredNameSpace", "ignoredName", x)>
  </cfif>
  <cfelse>
    <!--- Add a header as a string - Must generate error! --->
    <cfset addSOAPResponseHeader("http://www.tomj.org/myns", "returnheader", "AUTHORIZED VALUE", false)>
  </cfelse>
  <cfreturn return = "Not invoked as a web service">"
</cffunction>
</cfcomponent>

466  Chapter 3: ColdFusion Functions
ArrayAppend

Description
Appends an array element to an array.

Returns
True, on successful completion.

Category
Array functions

Function syntax
ArrayAppend(array, value)

See also

History
ColdFusion MX: Changed behavior: this function can be used on XML objects.

Parameters
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>array</td>
<td>Name of an array</td>
</tr>
<tr>
<td>value</td>
<td>Value to add at end of array</td>
</tr>
</tbody>
</table>

Example
<h3>ArrayAppend Example</h3>
<cfquery name = "GetEmployeeNames" datasource = "cfdocexamples">
SELECT FirstName, LastName FROM Employees</cfquery>
<!--- create an array --->
<cfset myArray = ArrayNew(1)>
<!--- set element one to show where we are --->
<cfset myArray[1] = "Test Value">
<!--- loop through the query; append these names successively to the last element --->
<cfloop query = "GetEmployeeNames">
<cfoutput>#ArrayAppend(myArray, "##FirstName## ##LastName##")#</cfoutput>, Array was appended<br>
</cfloop>
<!--- show the resulting array as a list --->
<cfset myList = ArrayToList(myArray, ",")>
<!--- output the array as a list --->
<cfoutput>
<p>The contents of the array are as follows:
<p>#myList#</cfoutput>
</cfoutput>
ArrayAvg

Description
Calculates the average of the values in an array.

Returns
Number. If the array parameter value is an empty array, returns zero.

Category
Array functions, Mathematical functions

Function syntax
ArrayAvg(array)

See also

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>array</td>
<td>Name of an array</td>
</tr>
</tbody>
</table>

Usage
The following example uses the ColdFusion built-in variable Form.fieldNames, which is available on the action page of a form. It contains a comma-delimited list of the names of the fields on the form.

Example
<!--- This example shows the use of ArrayAvg --->
<!--- The following lines of code keep track of the form fields that can be dynamically generated on the screen. It uses the Fieldnames variable with the ListLen function to determine the number of fields on the form. --->
<cfset FormElem = 2>
<cfif Isdefined("Form.Submit")>
  <cfif Form.Submit is "More">
    <cfset FormElem = ListLen(Form.Fieldnames)>
    <cfif>
    <cfif>
</cfif>
</cfif>

<html>
<head>
<title>ArrayAvg Example</title>
</head>
<body>
<h3>ArrayAvg Example</h3>
<p>This example uses ArrayAvg to find the average of the numbers that you enter into an array.<br>To enter more than two numbers press the <b>more</b> button.</p>
</body>
<form action = "arrayavg.cfm">
<!--- The following code initially creates two fields. It adds fields if the user presses MORE. FormElem is initialized to two at the beginning of this code to show that the form has two fields. ----->
<input type = "submit" name = "submit" value = "more">
<table cellspacing = "2" cellpadding = "2" border = "0">
<cfloop index = "LoopItem" from = "1" to = "#FormElem#">
<tr>
<cfoutput>
<th align = "left">Number #LoopItem#</th>
<td><input type = "text" name = "number#LoopItem#"></td>
</cfoutput>
</tr>
</cfloop>
</table>
<input type = "submit" name = "submit" value = "get the average">
</form>

<!--- create an array --->
<cfif IsDefined("FORM.submit")>
<cfset myNumberArray = ArrayNew(1)>
<cfset Count = 1>
<cfloop index = "ListItem" list = "#Form.Fieldnames#">
<cfif Left(ListItem,3) is "Num">
<cfset myNumberArray[Count] = Val("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></p>
<cfelse>
<cfoutput>Try again. You must enter at least two numeric values.</cfoutput>
</cfif>
</cfif>
</body>
</html>
ArrayClear

Description
Deletes the data in an array.

Returns
True, on successful completion.

Category
Array functions

Function syntax
ArrayClear(array)

See also

History
ColdFusion MX: Changed behavior: this function can be used on XML objects.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>array</td>
<td>Name of an array</td>
</tr>
</tbody>
</table>

Example
<h3>ArrayClear Example</h3>
<!--- create a new array --->
<cfset MyArray = ArrayNew(1)>
<!--- populate an element or two --->
<cfset MyArray[1] = "Test">
<cfset MyArray[2] = "Other Test">
<!--- output the contents of the array --->
<p>Your array contents are: <cfoutput>#ArrayToList(MyArray)#</cfoutput>
<!--- check if the array is empty --->
<p>Is the array empty?: <cfoutput>#ArrayIsEmpty(MyArray)#</cfoutput>
<p>Now, clear the array:
<!--- now clear the array --->
<cfset Temp = ArrayClear(MyArray)>
<!--- check if the array is empty --->
<p>Is the array empty?: <cfoutput>#ArrayIsEmpty(MyArray)#</cfoutput>
ArrayDeleteAt

Description

Deletes an element from an array.

When an element is deleted, ColdFusion recalculates index positions. For example, in an array that contains the months of the year, deleting the element at position 5 removes the entry for May. After this, to delete the entry for June, you would delete the element at position 5 (not 6).

Returns

True, on successful completion.

Category

Array functions

Function syntax

ArrayDeleteAt(array, position)

See also

ArrayInsertAt; “Functions for XML object management” in Chapter 35, “Using XML and WDDX,” in ColdFusion MX Developer’s Guide

History

ColdFusion MX:

• Changed behavior: this function can be used on XML objects.
• Changed thrown exceptions: this function can throw the InvalidArrayIndexException error.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>array</td>
<td>Name of an array</td>
</tr>
<tr>
<td>position</td>
<td>Array position</td>
</tr>
</tbody>
</table>

Throws

If this function attempts to delete an element at position 0, or specifies a value for position that is greater than the size of array, this function throws an InvalidArrayIndexException error.

Example

We delete this element of the array:
#ArrayDeleteAt(DaysArray,2)#

<!--- the formerly third element, "Wednesday" is second element --->

This is what the array looks like after delete:
ArrayInsertAt

Description
Inserts a value into an array. Array elements whose indexes are equal to or greater than the new position are incremented by one. The array length increases by one.

Returns
True, on successful completion.

Category
Array functions

Function syntax
ArrayInsertAt(array, position, value)

See also

History
ColdFusion MX:
• Changed behavior: this function can be used on XML objects.
• Changed thrown exceptions: this function can throw the InvalidArrayIndexException error.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>array</td>
<td>Name of an array</td>
</tr>
<tr>
<td>position</td>
<td>Index position at which to insert value</td>
</tr>
<tr>
<td>value</td>
<td>Value to insert</td>
</tr>
</tbody>
</table>

Usage
To apply the ArrayInsert() function to a multidimensional array, you must specify all but the last index in the array parameter. For example, the following code inserts an element at myarray[2][4]:
<cfset ArrayInsertAt(myarray[2], 4, “test”)>

Throws
If this function attempts to insert an element at position 0, or specifies a value for position that is greater than the size of array, this function throws an InvalidArrayIndexException error.

Example
<h3>ArrayInsertAt Example</h3><p>
<!--- create a new array --->
<cfset DaysArray = ArrayNew(1)>
<!--- populate an element or two --->

<cfset DaysArray[1] = "Monday">
<cfset DaysArray[2] = "Tuesday">
<cfset DaysArray[3] = "Thursday">
<!---- add an element before position 3 --->
<p>Add an element before position 3:</p>
<cfoutput>%ArrayInsertAt(DaysArray,3,"Wednesday")%</cfoutput>
<p>Now output the array as a list:</p>
<cfoutput>%ArrayToList(DaysArray)%</cfoutput>
<!---- The array now has four elements. Element 3, "Thursday", has become element four --->

---
ArrayIsEmpty

Description
Determines whether an array is empty of data elements.

Returns
True, if the array is empty; False, otherwise.

Category
Array functions

Function syntax
ArrayIsEmpty(array)

See also

History
ColdFusion MX: Changed behavior: this function can be used on XML objects.

Parameters
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>array</td>
<td>Name of an array</td>
</tr>
</tbody>
</table>

Example

```<h3>ArrayIsEmpty Example</h3>
<!--- create a new array --->
<cfset MyArray = ArrayNew(1)>
<!--- populate an element or two --->
<cfset MyArray[1] = "Test">
<cfset MyArray[2] = "Other Test">
<!--- output the contents of the array --->
<p>Your array contents are:
<cfoutput>#ArrayToList(MyArray)#</cfoutput>
<!--- check if the array is empty --->
<p>Is the array empty?:
<cfoutput>#ArrayIsEmpty(MyArray)#</cfoutput>
<p>Now, clear the array:
<!--- now clear the array --->
<cfset Temp = ArrayClear(MyArray)>
<!--- check if the array is empty --->
<p>Is the array empty?:
<cfoutput>#ArrayIsEmpty(MyArray)#</cfoutput>```
ArrayLen

Description
Determines the number of elements in an array.

Returns
The number of elements in an array.

Category
Array functions

Function syntax
ArrayLen(array)

See also

History
ColdFusion MX: Changed behavior: this function can be used on child XML objects.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>array</td>
<td>Name of an array</td>
</tr>
</tbody>
</table>

Example
<h3>ArrayLen Example</h3>
<cfquery name = "GetEmployeeNames" datasource = "cfdocexamples">
SELECT FirstName, LastName FROM Employees
</cfquery>
<!--- create an array --->
<cfset myArray = ArrayNew(1)>
<!--- set element one to show where we are --->
<cfset myArray[1] = "Test Value">
<!--- loop through the query and append these names successively to the last element --->
<cfloop query = "GetEmployeeNames">
<cfset temp = ArrayAppend(myArray, 
"#FirstName# #LastName#")>
</cfloop>
<!--- show the resulting array as a list --->
<cfset myList = ArrayToList(myArray, ",")>
<!--- output the array as a list --->
<cfoutput>
<p>The contents of the array are as follows:
<p>#myList#
<p>This array has #ArrayLen(MyArray)# elements.
</cfoutput>
**ArrayMax**

**Description**

Array maximum function.

**Returns**

The largest numeric value in an array. If the array parameter value is an empty array, returns zero.

**Category**

Array functions

**Function syntax**

ArrayMax(array)

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>array</td>
<td>Name of an array</td>
</tr>
</tbody>
</table>

**Example**

<h3>ArrayMax Example</h3>

This example uses ArrayMax to find the largest number in an array.

```cfml
<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</p>
    <cfoutput>#ArrayMax(myNumberArray)#</cfoutput>
  </cfif>
</cfif>
```

The following form provides two numeric fields that are compared when the form is submitted.

```cfml
<form action = "arraymax.cfm">
  <input type = "hidden" name = "number1_Float">
  <input type = "hidden" name = "number2_Float">
  <input type = "text" name = "number1">
  <br>
  <input type = "text" name = "number2">
  <br>
  <input type = "submit" name = "submit" value = "Maximum Value">
</form>
```
ArrayMin

Description
Array minimum function.

Returns
The smallest numeric value in an array. If the array parameter value is an empty array, returns zero.

Category
Array functions

Function syntax
ArrayMin(array)

Parameters
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>array</td>
<td>Name of an array</td>
</tr>
</tbody>
</table>

Example

```
<h3>ArrayMin Example</h3>
<p>This example uses ArrayMin to find the smallest number in an array.</p>
<cfif IsDefined("FORM.submit")>
  <cfset myNumberArray = ArrayNew(1)>
  <cfset myNumberArray[1] = FORM.number1>
  <cfset myNumberArray[2] = FORM.number2>

  <cfif Form.Submit is "Minimum Value">
    <cfoutput>The smallest number that you entered is #ArrayMin(myNumberArray)#.</cfoutput>
  </cfif>
</cfif>

<!---- The following form provides two numeric fields that are compared when the form is submitted. ---->
<form action="arraymin.cfm">
  <input type="hidden" name="number1_Float">
  <input type="hidden" name="number2_Float">
  <input type="text" name="number1"><br>
  <input type="text" name="number2"><br>
  <input type="submit" name="submit" value="Minimum Value">
</form>
```
ArrayNew

Description

Creates an array of 1–3 dimensions. Index array elements with square brackets: [ ].
ColdFusion arrays expand dynamically as data is added.

Returns

An array

Category

Array functions

Function syntax

ArrayNew(dimension)

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dimension</td>
<td>Number of dimensions in new array: 1, 2, or 3</td>
</tr>
</tbody>
</table>

Example

```<h3>ArrayNew Example</h3> <!--- Make an array --->
<cfset MyNewArray = ArrayNew(1)>
<!--- ArrayToList does not function properly if the Array is not initialized with ArraySet --->
<cfset temp = ArraySet(MyNewArray, 1, 6, "")>
<!--- set some elements --->
<cfset MyNewArray[1] = "Sample Value">
<cfset MyNewArray[3] = "43">
<cfset MyNewArray[6] = "Another Value">
<!--- is it an array? --->
<cfoutput>
<p>Is this an array? #IsArray(MyNewArray)#
<p>It has #ArrayLen(MyNewArray)# elements.
<p>Contents: #ArrayToList(MyNewArray)#
<!--- the array has expanded dynamically to six elements with the use of ArraySet, even though we only set three values --->
</cfoutput>```
ArrayPrepend

Description
Inserts an array element at the beginning of an array.

Returns
True, on successful completion.

Category
Array functions

Function syntax
ArrayPrepend(array, value)

See also
ArrayAppend; “Adding elements to an array” in Chapter 5, “Using Arrays and Structures,” in ColdFusion MX Developer’s Guide

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>array</td>
<td>Name of an array</td>
</tr>
<tr>
<td>value</td>
<td>Value to insert at beginning of array</td>
</tr>
</tbody>
</table>

Example

```cf<br>
<h3>ArrayPrepend Example</h3>
<cfquery name = "GetEmployeeNames" datasource = "cfdocexamples">
    SELECT FirstName, LastName FROM Employees
</cfquery>
 <!--- create an array --->
<cfset myArray = ArrayNew(1)>
 <!--- set element one to show where we are --->
<cfset myArray[1] = "Test Value">
 <!--- loop through query. Append names successively before last element (list reverses itself from the standard queried output, as it keeps prepending the array entry) --->
<cfloop query = "GetEmployeeNames">
    <cfoutput>#ArrayPrepend(myArray, "#FirstName# #LastName#")#
</cfoutput>. Array was prepended<br>
</cfloop>
 <!--- show the resulting array as a list --->
<cfset myList = ArrayToList(myArray, ",")>
 <!--- output the array as a list --->
<cfoutput>
<p>The contents of the array are as follows:<p>
 #myList#
</cfoutput>
```
ArrayResize

Description

Resets an array to a specified minimum number of elements. This can improve performance, if used to size an array to its expected maximum. For more than 500 elements, use `ArrayResize` immediately after using the `ArrayNew` tag.

ColdFusion arrays expand dynamically as data is added.

Returns

True, on successful completion.

Category

Array functions

Function syntax

```
ArrayResize(array, minimum_size)
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>array</td>
<td>Name of an array</td>
</tr>
<tr>
<td>minimum_size</td>
<td>Minimum array size</td>
</tr>
</tbody>
</table>

Example

```<h3>ArrayResize Example</h3> <!--- perform a query to get the list --->
<cfquery name = "GetCourses" datasource = "cfdocexamples"> SELECT * FROM Courses </cfquery> <!--- make a new array --->
<cfset MyArray = ArrayNew(1)> <!--- resize that array to the number of records in the query --->
<cfset temp = ArrayResize(MyArray, GetCourses.RecordCount)> <cfoutput> The array is now #ArrayLen(MyArray)# elements, to match the query of #GetCourses.RecordCount# records. </cfoutput>```
ArraySet

Description

In a one-dimensional array, sets the elements in a specified index range to a value. Useful for initializing an array after a call to ArrayNew.

Returns

True, on successful completion.

Category

Array functions

Function syntax

ArraySet(array, start_pos, end_pos, value)

See also


History

ColdFusion MX: Changed behavior: this function can be used on XML objects.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>array</td>
<td>Name of an array.</td>
</tr>
<tr>
<td>start_pos</td>
<td>Starting index position of range to set.</td>
</tr>
<tr>
<td>end_pos</td>
<td>Ending index position of range to set. If this value is greater than array length, ColdFusion adds elements to array.</td>
</tr>
<tr>
<td>value</td>
<td>Value to which to set each element in the range.</td>
</tr>
</tbody>
</table>

Example

```
<h3>ArraySet Example</h3>

<!--- Make an array --->
<cfset MyNewArray = ArrayNew(1)>

<!--- ArrayToList does not function properly if the Array has not been initialized with ArraySet --->
<cfset temp = ArraySet(MyNewArray, 1,6, "Initial Value")>

<!--- set some elements --->
<cfset MyNewArray[1] = "Sample Value">
<cfset MyNewArray[3] = "43">
<cfset MyNewArray[6] = "Another Value">

...
ArraySort

Description
Sorts array elements numerically or alphanumerically.

Returns
True, if sort is successful; False, otherwise.

Category
Array functions, List functions

Function syntax
ArraySort(array, sort_type [, sort_order])

History
ColdFusion MX:
- Changed thrown exceptions: this function can throw the ArraySortSimpleValueException error and ValueNotNumeric error.
- Changed the order in which sorted elements are returned: in a textnocase, descending sort, this function might return elements in a different sort order than in earlier releases. If sort_type = "textnocase" and sort_order = "desc", ColdFusion MX processes elements that differ only in case differently from earlier releases, as follows:
  • ColdFusion MX reverses the elements’ original order.
  • Earlier releases of ColdFusion do not change the elements’ original order.
For example, in a textnocase, desc sort of d,a,a,b,A, the following occurs:
  • ColdFusion MX returns d,b,A,a,a
  • Earlier ColdFusion releases return d,b,a,a,A

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>array</td>
<td>Name of an array</td>
</tr>
</tbody>
</table>
Throws

If an array element is other than a simple element, this function throws an ArraySortSimpleValueException error. If sort_type is numeric and an array element is not numeric, this function throws a ValueNotNumeric error.

Example

<!--- This example shows ArraySort --->
<cfquery name = "GetEmployeeNames" datasource = "cfdocexamples">
  SELECT FirstName, LastName FROM Employees
</cfquery>
<!--- create an array --->
<cfset myArray = ArrayNew(1)>
<!--- loop through the query and append these names successively to the last element --->
<cfloop query = "GetEmployeeNames">
  <cfset temp = ArrayAppend(myArray, "#FirstName# #LastName#")>
</cfloop>
<!--- show the resulting array as a list --->
<cfset myList = ArrayToList(myArray, ",")>
<!--- sort that array descending alphabetically --->
<cfset isSuccessful = ArraySort(myArray, "textnocase", "desc")>
...
ArraySum

Description
Array sum function.

Returns
The sum of values in an array. If the array parameter value is an empty array, returns zero.

Category
Array functions, Mathematical functions

Function syntax
ArraySum(array)

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>array</td>
<td>Name of an array</td>
</tr>
</tbody>
</table>

Example
<h3>ArraySum Example</h3>
<p>This example uses ArraySum to add two numbers together.</p>
<!--- After checking whether the form has been submitted, the code creates an array and assigns the form fields to the first two elements in the array. --->
<cfif IsDefined("FORM.submit")>
<cfset myNumberArray = ArrayNew(1)>
<cfset myNumberArray[1] = number1>
<cfset myNumberArray[2] = number2>

<cfif Form.Submit is "Add">
<!--- use ArraySum to add the number in the array --->
<p>The sum of the numbers is</p>
<cfoutput>#ArraySum(myNumberArray)#.</cfoutput>
</cfif>
</cfif>
<!--- This form provides two numeric fields that are added when the form is submitted. --->
<form action = "arraysum.cfm" method="post">
<input type = "hidden" name = "number1_Float">
<input type = "hidden" name = "number2_Float">
<input type = "text" name = "number1">  
<br>
<input type = "text" name = "number2">  
<br>
<input type = "submit" name = "submit" value = "Add">  
</form>
**ArraySwap**

**Description**
Swaps array values of an array at specified positions. This function is more efficient than multiple `cfset` tags.

**Returns**
True, on successful completion.

**Category**
Array functions

**Function syntax**
```
ArraySwap(array, position1, position2)
```

**See also**

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>array</td>
<td>Name of an array</td>
</tr>
<tr>
<td>position1</td>
<td>Position of first element to swap</td>
</tr>
<tr>
<td>position2</td>
<td>Position of second element to swap</td>
</tr>
</tbody>
</table>

**Example**

```coldfusion
<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>
```

486 Chapter 3: ColdFusion Functions
ArrayToList

Description
Converts a one-dimensional array to a list.

Returns
Delimited list, as a string.

Category
Array functions, Conversion functions, List functions

Function syntax
ArrayToList(array [, delimiter ])

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>array</td>
<td>Name of array</td>
</tr>
<tr>
<td>delimiter</td>
<td>Character or multicharacter string to separate list elements. The default value is comma.</td>
</tr>
</tbody>
</table>

Example
<cfquery name = "GetEmployeeNames" datasource = "cfdocexamples">
SELECT FirstName, LastName FROM Employees
</cfquery>

<!--- create an array --->
<cfset myArray = ArrayNew(1)>

<!--- loop through query, append names successively to last element --->
<cfloop query = "GetEmployeeNames">
    <cfset temp = ArrayAppend(myArray, "#FirstName# #LastName#")>
</cfloop>

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

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

<!--- show the resulting alphabetized array as a list --->
<cfset myAlphaList = ArrayToList(myArray, ",")>

<!--- output the array as a list --->
<cfoutput>
<p>The contents of the array are as follows:
<p>#myList#
<p>This array, alphabetized by first name (descending):
<p>#myAlphaList#
<p>This array has #ArrayLen(MyArray)# elements.
</cfoutput>
Asc

Description
Determined the value of a character.

Returns
The value of the first character of a string; if string is empty, returns zero.

Category
String functions

Function syntax
Asc(string)

See also
Chr

History
ColdFusion MX: Changed Unicode support: ColdFusion supports the Java UCS-2 representation of Unicode characters, up to a value of 65536. (Earlier releases supported 1-255.)

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>string</td>
<td>A string</td>
</tr>
</tbody>
</table>

Example

```html
<h3>Asc Example</h3>
<!--- if the character string is not empty, output its ASCII value --->
<cfif IsDefined("FORM.charVals")>
    <cfif FORM.charVals is not ">
        <cfoutput>Left(FORM.charVals,1) = #Asc(FORM.charVals)#</cfoutput>
    </cfif>
<cfelse>
    <!--- if it is empty, output an error message --->
    <h4>Enter a character</h4>
</cfif>
</cfif>

<form action = "asc.cfm" method=post>
Enter a character to see its ASCII value
<br><input type = "Text" name = "CharVals" size = "1" maxlength = "1">
<br><input type = "Submit" name = ""> <input type = "RESET">
</form>
```
ASin

Description
Determines the arcsine of a number. The arcsine is the angle whose sine is number.

Returns
The arcsine, in radians, of a number.

Category
Mathematical functions

Function syntax
ASin(number)

See also
Sin, Cos, ACos, Tan, Atn, Pi

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>number</td>
<td>Sine of an angle. The value must be between -1 and 1, inclusive.</td>
</tr>
</tbody>
</table>

Usage
The range of the result is $-\pi/2$ to $\pi/2$ radians. To convert degrees to radians, multiply degrees by $\pi/180$. To convert radians to degrees, multiply radians by $180/\pi$.

Example

```
<h3>ASin Example</h3>
<!--- output its arcsine value --->
<cfif IsDefined("FORM.SinNum")>
  <cfif IsNumeric(FORM.SinNum)>
    <cfif FORM.SinNum LESS THAN OR EQUAL TO 1>
      <cfif FORM.SinNum GREATER THAN OR EQUAL TO -1>
        ASin(<cfoutput>#FORM.SinNum#</cfoutput>) = <cfoutput>#ASin(FORM.sinNum)# Radians</cfoutput>
        or <cfoutput>#ASin(FORM.sinNum# * 180/Pi())# Degrees</cfoutput>
      </cfif>
    </cfif>
  </cfif>
</cfif>
<!--- if it is less than negative one, output an error message --->
<h4>Enter the sine of the angle to calculate, in degrees and radians. The value must be between 1 and -1, inclusive.</h4>
</cfif>
<!--- if it is greater than one, output an error message --->
<h4>Enter the sine of the angle to calculate, in degrees and radians. The value must be between 1 and -1, inclusive.</h4>
```
<cfelse>
<!--- if it is empty, output an error message --->
    <h4>Enter the sine of the angle to calculate, in degrees and radians. The value must be between 1 and -1, inclusive.</h4>
</cfif>
</cfif>
<form action="./evaltest.cfm" method="post">
<p>Enter a number to get its arcsine in Radians and Degrees.
<br><input type = "Text" name = "SinNum" size = "25">
<br><input type = "Submit" name = ""> <input type = "RESET">
</form>
Atn

Description

Arctangent function. The arctangent is the angle whose tangent is number.

Returns

The arctangent, in radians, of a number.

Category

Mathematical functions

Function syntax

Atn(number)

See also

Atn, Sin, ASin, Cos, ACos, Pi

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>number</td>
<td>Tangent of an angle</td>
</tr>
</tbody>
</table>

Usage

The range of the result is \(-\pi/2\) to \(\pi/2\) radians. To convert degrees to radians, multiply degrees by \(\pi/180\). To convert radians to degrees, multiply radians by \(180/\pi\).

Example

```html
<h3>Atn Example</h3>
<!--- output its Atn value --->
<cfif IsDefined("FORM.AtnNum")>
  <cfif IsNumeric(FORM.AtnNum)>
    Atn(<cfoutput>#FORM.AtnNum#</cfoutput>) is <cfoutput>#Atn(FORM.AtnNum)# radians is #Atn(FORM.AtnNum * 180/PI())# Degrees</cfoutput>
  </cfifelse>
<!--- if it is empty, output an error message --->
  <h4>Enter a number</h4>
</cfif>
```

<cfif>
<form action = "evaltest.cfm" method="post">
  <p>Enter a number to get its arctangent in Radians and Degrees</p>
  <br>
  <input type = "Text" name = "atnNum" size = "25">
  <br>
  <input type = "Submit" name = ""> <input type = "RESET">
</form>
```
AuthenticatedContext

Description

This function is obsolete. Use the newer security tools; see “Conversion functions” on page 453 and Chapter 16, “Securing Applications” in ColdFusion MX Developer's Guide.

History

ColdFusion MX: This function is obsolete. It does not work in ColdFusion MX and later ColdFusion releases.
AuthenticatedUser

Description

This function is obsolete. Use the newer security tools; see “Conversion functions” on page 453 and Chapter 16, “Securing Applications” in ColdFusion MX Developer's Guide.

History

ColdFusion MX: This function is obsolete. It does not work in ColdFusion MX and later ColdFusion releases.
BinaryDecode

Description
Converts a string to a binary object. Used to convert binary data that has been encoded into string format back into binary data.

Returns
A binary object.

Category
Conversion functions, String functions

Function syntax
BinaryDecode(string, binaryencoding)

See also
BinaryEncode, CharsetEncode, CharsetDecode

History
ColdFusion MX 7: Added this function.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>string</td>
<td>A string containing encoded binary data.</td>
</tr>
<tr>
<td>binaryencoding</td>
<td>A string specifying the algorithm used to encode the original binary data into a string; must be one of the following:</td>
</tr>
<tr>
<td></td>
<td>• Hex: the characters 0-9 and A-F represent the hexadecimal value of each byte; for example, 3A.</td>
</tr>
<tr>
<td></td>
<td>• UU: data is encoded using the UNIX UUencode algorithm.</td>
</tr>
<tr>
<td></td>
<td>• Base64: data is encoded using the Base64 algorithm, as specified by IETF RFC 2045, at <a href="http://www.ietf.org/rfc/rfc2045.txt">www.ietf.org/rfc/rfc2045.txt</a>.</td>
</tr>
</tbody>
</table>

Usage
Use this function to convert a binary-encoded string representation of binary data back to a binary object for use in your application. Binary data is often encoded as a string for transmission over many Internet protocols, such as HTTP and SMTP, or for storage in a database.

Macromedia recommends that you use the BinaryDecode function, not the ToBinary(base64data) function, to convert Base64-encoded data to binary data in all new applications.

See the following pages for additional information on handling binary data:
• cffile for loading and reading binary data in files
• cfwddx for serializing and deserializing binary data
• IsBinary for checking variables for binary format
• Len for determining the length of a binary object
Example

The following example reads a GIF file as binary data, converts it to a binary-encoded string, converts the binary-encoded data back to binary data and writes the result to a file. It displays the encoded string and the image in the output file.

<h3>Binary Encoding Conversion Example</h3>

<!--- Do the following if the form has been submitted. --->
<cfif IsDefined("Form.binEncoding")>

<!--- Read in a binary data file. --->
<cffile action="readbinary"
file="C:\CFusionMX7\wwwroot\CFIDE\administrator\images\help.gif"
variable="binimage">

<!--- Convert the read data to binary encoding and back to binary data. --->
<cfscript>
binencode=BinaryEncode(binimage, Form.binEncoding);
bindecode=BinaryDecode(binencode, Form.binEncoding);
</cfscript>

<!--- Write the converted results to a file. --->
<cffile action="write" file="C:\temp\help.gif" output="#bindecode#"
addnewline="No">

<!--- Display the results. --->
<cfoutput>
<p><b>The binary encoding:</b> #Form.binEncoding#</p>
<p><b>The image converted into a binary-encoded string by BinaryEncode</b>
<br>#binencode#</p>
<p><b>The image as written back to a file after converting back to binary
using BinaryDecode</b><br><img src="C:\temp\help.gif"></p>
</cfoutput>
</cfif>

<!--- The input form. --->
<form action="#CGI.SCRIPT_NAME#" method="post">
<b>Select binary encoding</b><br>
<select size="1" name="binEncoding">
<option selected>UU</option>
<option>Base64</option>
<option>Hex</option>
</select><br>
<br>
<input type = "Submit" value = "convert my data">
</form>
BinaryEncode

Description
Converts binary data to a string.

Returns
An encoded string representing the binary data.

Category
Conversion functions, String functions

Function syntax
BinaryEncode(binarydata, encoding)

See also
BinaryDecode, CharsetEncode, CharsetDecode

History
ColdFusion MX 7: Added this function.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>binarydata</td>
<td>A variable containing the binary data to encode.</td>
</tr>
<tr>
<td>encoding</td>
<td>A string specifying the encoding method to use to represent the data; one of the following:</td>
</tr>
</tbody>
</table>

- Hex: use the characters 0-9 and A-F to represent the hexadecimal value of each byte; for example, 3A.
- UU: use the UNIX UUencode algorithm to convert the data.
- Base64: use the Base64 algorithm to convert the data, as specified by IETF RFC 2045, at www.ietf.org/rfc/rfc2045.txt.

Usage
Binary objects and, in some cases, 8-bit characters, cannot be transported over many Internet protocols, such as HTTP and SMTP, and might not be supported by some database systems. By Binary encoding the data, you convert the data into a format that you can transfer over any Internet protocol or store in a database as character data. To convert the data back to a binary format, use the BinaryDecode function.

Macromedia recommends that you use the BinaryEncode function, and not the ToBase64(binarydata) function, to convert binary data to Base64 data in all new applications.

This function provides a superset of the functionality of the ToBase64(binarydata) function.

See the following pages for additional information on handling binary data:
-cffile for loading and reading binary data
- cfwddx for serializing and deserializing binary data
• **IsBinary** for checking variables for binary format
• **Len** for determining the length of a binary object

**Example**

The following example reads a GIF file as binary data, converts it to a binary-encoded string, converts the binary-encoded data back to binary data, and writes the result to a file. It displays the encoded string and the image in the output file.

```html
<h3>Binary Encoding Conversion Example</h3>

<!--- Do the following if the form has been submitted. --->
<cfif IsDefined("Form.binEncoding")>

<!--- Read in a binary data file. --->
<cfif IsDefined("Form.binEncoding")>
<cffile action="readbinary"
file="C:\CFusionMX7\wwwroot\CFIDE\administrator\images\help.gif"
variable="binimage">

<!--- Convert the read data to binary encoding and back to binary data. --->
<cfscript>
binencode=BinaryEncode(binimage, Form.binEncoding);
bindecode=BinaryDecode(binencode, Form.binEncoding);
</cfscript>

<!--- Write the converted results to a file. --->
<cffile action="write" file="C:\temp\help.gif" output="#bindecode#"
addnewline="No" >

<!--- Display the results. --->
<cfoutput>
<p><b>The binary encoding:</b> #Form.binEncoding#</p>
<p><b>The image converted into a binary-encoded string by BinaryEncode</b>
<br>#binencode#</p>
<p><b>The image as written back to a file after converting back to binary</b>
<br>using BinaryDecode</p><img src="C:\temp\help.gif">
</cfoutput>
</cfif>

<!--- The input form. --->
<form action="#CGI.SCRIPT_NAME#" method="post">
<b>Select binary encoding</b><br>
<select size="1" name="binEncoding">
<option selected>UU</option>
<option>Base64</option>
<option>Hex</option>
</select><br>
<br>
<input type = "Submit" value = "convert my data">
</form>
**BitAnd**

**Description**
Performs a bitwise logical AND operation.

**Returns**
The bitwise AND of two long integers.

**Category**
Mathematical functions

**Function syntax**
`BitAnd(number1, number2)`

**See also**
BitNot, BitOr, BitXor

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>number1</td>
<td>32-bit signed integer</td>
</tr>
<tr>
<td>number2</td>
<td>32-bit signed integer</td>
</tr>
</tbody>
</table>

**Usage**
Bit functions operate on 32-bit signed integers, in the range -2147483648 – 2147483647.

**Example**

```coldfusion
<h3>BitAnd Example</h3>
<p>Returns the bitwise AND of two long integers.</p>
<p>BitAnd(5,255): <cfoutput>#BitAnd(5,255)#</cfoutput></p>
<p>BitAnd(5,0): <cfoutput>#BitAnd(5,0)#</cfoutput></p>
<p>BitAnd(128,128): <cfoutput>#BitAnd(128,128)#</cfoutput></p>
```
BitMaskClear

Description
Performs a bitwise mask clear operation.

Returns
A number, bitwise cleared, with length bits beginning at start.

Category
Mathematical functions

Function syntax
BitMaskClear(number, start, length)

See also
BitMaskRead, BitMaskSet

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>number</td>
<td>32-bit signed integer</td>
</tr>
<tr>
<td>start</td>
<td>Integer, in the range 0-31, inclusive; start bit for mask</td>
</tr>
<tr>
<td>length</td>
<td>Integer, in the range 0-31, inclusive; length of mask</td>
</tr>
</tbody>
</table>

Usage
Bit functions operate on 32-bit signed integers, in the range -2147483648 – 2147483647.

Example
<h3>BitMaskClear Example</h3>
<p>Returns number bitwise cleared with length bits beginning from start.</p>

```cftoutput
BitMaskClear(255, 4, 4): #BitMaskClear(255, 4, 4)#
</cftoutput>

```cftoutput
BitMaskClear(255, 0, 4): #BitMaskClear(255, 0, 4)#
</cftoutput>

```cftoutput
BitMaskClear(128, 0, 7): #BitMaskClear(128, 0, 7)#
</cftoutput>
**BitMaskRead**

**Description**

Performs a bitwise mask read operation.

**Returns**

An integer, created from *length* bits of *number*, beginning at *start*.

**Category**

Mathematical functions

**Function syntax**

`BitMaskRead(number, start, length)`

**See also**

`BitMaskClear, BitMaskSet`

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>number</td>
<td>32-bit signed integer to mask</td>
</tr>
<tr>
<td>start</td>
<td>Integer, in the range 0-31, inclusive; start bit for read</td>
</tr>
<tr>
<td>length</td>
<td>Integer, in the range 0-31, inclusive; length of mask</td>
</tr>
</tbody>
</table>

**Usage**

Bit functions operate on 32-bit signed integers, in the range -2147483648 – 2147483647.

**Example**

```coldfusion
<p>BitMaskRead Example</p>
<p>Returns integer created from *length* bits of *number*, beginning with *start*.</p>
<p>BitMaskRead(255, 4, 4): <cfoutput>#BitMaskRead(255, 4, 4)#</cfoutput></p>
<p>BitMaskRead(255, 0, 4): <cfoutput>#BitMaskRead(255, 0, 4)#</cfoutput></p>
<p>BitMaskRead(128, 0, 7): <cfoutput>#BitMaskRead(128, 0, 7)#</cfoutput></p>
```
**BitMaskSet**

**Description**

Performs a bitwise mask set operation.

**Returns**

A number, bitwise masked with *length* bits of *mask* beginning at *start*.

**Category**

Mathematical functions

**Function syntax**

```
BitMaskSet(number, mask, start, length)
```

**See also**

BitMaskClear, BitMaskRead

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>number</td>
<td>32-bit signed integer</td>
</tr>
<tr>
<td>mask</td>
<td>32-bit signed integer; mask</td>
</tr>
<tr>
<td>start</td>
<td>Integer, in the range 0-31, inclusive; start bit for mask</td>
</tr>
<tr>
<td>length</td>
<td>Integer, in the range 0-31, inclusive; length of mask</td>
</tr>
</tbody>
</table>

**Usage**

Bit functions operate on 32-bit signed integers, in the range -2147483648 – 2147483647.

**Example**

```
<h3>BitMaskSet Example</h3>
<p>Returns number bitwise masked with length bits of mask beginning at start.

<p>BitMaskSet(255, 255, 4, 4):
<cfoutput>@BitMaskSet(255, 255, 4, 4)@</cfoutput>
<p>BitMaskSet(255, 0, 4, 4):
<cfoutput>@BitMaskSet(255, 0, 4, 4)@</cfoutput>
<p>BitMaskSet(0, 15, 4, 4):
<cfoutput>@BitMaskSet(0, 15, 4, 4)@</cfoutput>
```
BitNot

Description
Performs a bitwise logical NOT operation.

Returns
A number; the bitwise NOT of a long integer.

Category
Mathematical functions

Function syntax
BitNot(number)

See also
BitAnd, BitOr, BitXor

Parameters
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>number</td>
<td>32-bit signed integer</td>
</tr>
</tbody>
</table>

Usage
Bit functions operate on 32-bit signed integers, in the range -2147483648 – 2147483647.

Example
<h3>BitNot Example</h3>
<p>Returns the bitwise NOT of a long integer.</p>

<p>BitNot(0): <cfoutput>#BitNot(0)#</cfoutput></p>
<p>BitNot(255): <cfoutput>#BitNot(255)#</cfoutput></p>
**BitOr**

**Description**

Performs a bitwise logical OR operation.

**Returns**

A number; the bitwise OR of two long integers.

**Category**

Mathematical functions

**Function syntax**

\[ \text{BitOr}(\text{number1}, \text{number2}) \]

**See also**

BitAnd, BitNot, BitXor

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>number1</td>
<td>32-bit signed integer</td>
</tr>
<tr>
<td>number2</td>
<td>32-bit signed integer</td>
</tr>
</tbody>
</table>

**Usage**

Bit functions operate on 32-bit signed integers, in the range -2,147,483,648 – 2,147,483,647.

**Example**

```html
<h3>BitOr Example</h3>
<p>Returns the bitwise OR of two long integers.</p>

<p>BitOr(5,255): <cfoutput>#$BitOr(5,255)#</cfoutput></p>
<p>BitOr(5,0): <cfoutput>#$BitOr(5,0)#</cfoutput></p>
<p>BitOr(7,8): <cfoutput>#$BitOr(7,8)#</cfoutput></p>
```
**BitSHLN**

**Description**
Performs a bitwise shift-left, no-rotation operation.

**Returns**
A number, bitwise shifted without rotation to the left by *count* bits.

**Category**
Mathematical functions

**Function syntax**
```
BitSHLN(number, count)
```

**See also**
BitSHRN

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>number</td>
<td>32-bit signed integer</td>
</tr>
<tr>
<td>count</td>
<td>Integer, in the range 0-31, inclusive; number of bits to shift the number</td>
</tr>
</tbody>
</table>

**Usage**
Bit functions operate on 32-bit signed integers, in the range -2147483648 – 2147483647.

**Example**

```html
<h3>BitSHLN Example</h3>
<p>Returns the number, bitwise shifted, without rotation, to the left by count bits.</p>
<p>BitSHLN(1,1): <cfoutput>&#1058;BitSHLN(1,1);&#1048;</cfoutput></p>
<p>BitSHLN(1,30): <cfoutput>&#1058;BitSHLN(1,30);&#1048;</cfoutput></p>
<p>BitSHLN(1,31): <cfoutput>&#1058;BitSHLN(1,31);&#1048;</cfoutput></p>
<p>BitSHLN(2,31): <cfoutput>&#1058;BitSHLN(2,31);&#1048;</cfoutput>
```
**BitSHRN**

**Description**

Performs a bitwise shift-right, no-rotation operation.

**Returns**

A number, bitwise shifted, without rotation, to the right by `count` bits.

**Category**

Mathematical functions

**Function syntax**

`BitSHRN(number, count)`

**See also**

`BitSHLN`

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>number</code></td>
<td>32-bit signed integer</td>
</tr>
<tr>
<td><code>count</code></td>
<td>Integer, in the range 0-31, inclusive. Number of bits to shift the number</td>
</tr>
</tbody>
</table>

**Usage**

Bit functions operate on 32-bit signed integers, in the range -2147483648 – 2147483647.

**Example**

```xml
<p>BitSHRN Example</p>
<p>Returns a number, bitwise shifted, without rotation, to the right, by count bits.</p>

<p>BitSHRN(1, 1): <output>BitSHRN(1, 1)</output></p>
<p>BitSHRN(255, 7): <output>BitSHRN(255, 7)</output></p>
<p>BitSHRN(-2147483548, 1): <output>BitSHRN(-2147483548, 1)</output></p>
```
BitXor

Description
Performs a bitwise logical XOR operation.

Returns
Bitwise XOR of two long integers.

Category
Mathematical functions

Function syntax
BitXor(number1, number2)

See also
BitAnd, BitNot, BitOr

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>number1</td>
<td>32-bit signed integer</td>
</tr>
<tr>
<td>number2</td>
<td>32-bit signed integer</td>
</tr>
</tbody>
</table>

Usage
Bit functions operate on 32-bit signed integers, in the range -2147483648 – 2147483647.

Example

```coldfusion
<cfset number1 = 5>
<cfset number2 = 255>
<cfoutput>
BitXor(number1, number2): #BitXor(number1, number2)#
</cfoutput>

<cfset number1 = 5>
<cfset number2 = 0>
<cfoutput>
BitXor(number1, number2): #BitXor(number1, number2)#
</cfoutput>

<cfset number1 = 128>
<cfset number2 = 128>
<cfoutput>
BitXor(number1, number2): #BitXor(number1, number2)#
</cfoutput>
```
Ceiling

Description
Determines the closest integer that is greater than a specified number.

Returns
The closest integer that is greater than a given number.

Category
Mathematical functions

Function syntax
Ceiling(number)

See also
Int, Fix, Round

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>number</td>
<td>A real number</td>
</tr>
</tbody>
</table>

Example

<h3>Ceiling Example</h3>

<cfoutput>
<p>The ceiling of 3.4 is #ceiling(3.4)#
<p>The ceiling of 3 is #ceiling(3)#
<p>The ceiling of 3.8 is #ceiling(3.8)#
<p>The ceiling of -4.2 is #ceiling(-4.2)#
</cfoutput>
**CharsetDecode**

**Description**
Converts a string to a binary representation.

**Returns**
A binary object that represents the string.

**Category**
Conversion functions, String functions

**Function syntax**
```
CharsetDecode(string, encoding)
```

**See also**

**History**
ColdFusion MX 7: Added this function.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>string</td>
<td>A string containing data to encode in binary format.</td>
</tr>
</tbody>
</table>
| encoding  | A string specifying encoding of the input data. Must be a character encoding name recognized by the Java runtime. The following list includes commonly used values:  
  - utf-8  
  - iso-8859-1  
  - windows-1252  
  - us-ascii  
  - shift_jis  
  - iso-2022-jp  
  - euc-jp  
  - euc-kr  
  - big5  
  - euc-cn  
  - utf-16  

For a complete list of character encoding names supported by Sun Java runtimes, see [http://java.sun.com/j2se/1.3/docs/guide/intl/encoding.doc.html](http://java.sun.com/j2se/1.3/docs/guide/intl/encoding.doc.html) and [http://java.sun.com/j2se/1.4/docs/guide/intl/encoding.doc.html](http://java.sun.com/j2se/1.4/docs/guide/intl/encoding.doc.html).

**Usage**
This function converts a string directly to a binary object. In releases of ColdFusion through ColdFusion MX 6.1, you had to use the ToBase64 function to convert the string to Base64 and then use the ToBinary function to convert strings to binary data.
Example

The following example uses the `CharsetDecode` function to convert a string from a form to a binary object, and uses the `CharsetEncode` function to convert it back to the original value. You can change the character encoding that ColdFusion uses for the conversion. Notice that if you select the Asian language encodings, characters that are not in the specified character set do get successfully converted.

```cfs
Example Character Encoding Conversion Example</h3>
<!--- Do the following if the form has been submitted. --->
<cfif IsDefined("Form.myString")>

<!--- Do the conversions. --->
<cfscript>
chardecode=CharsetDecode(Form.myString, Form.charEncoding);
charencode=CharsetEncode(chardecode, Form.charEncoding);
</cfscript>

<!--- Display the input values and results. --->
<cfoutput>
<h3>Parameter Settings</h3>
<p><b>The string:</b> #Form.myString#</p>
<p><b>The character encoding:</b> #Form.charEncoding#</p>
<h3>Results of the operations:</h3>
<p><b>Dump of the string converted to a binary object by CharsetDecode:</b>
<cfdump var="#chardecode#"></p>
<p><b>The binary object converted back to a string by CharsetEncode:</b>
<b>#charencode#</b></p>
</cfoutput>
</cfif>
</form>
```

<!--- The input form. --->
<form action="#CGI.SCRIPT_NAME#" method="post">
  <b>Select the character encoding</b>
  <br><b>This is a subset, additional encodings are available. ---></b>
  <select size="1" name="charEncoding">
    <option selected>UTF-8</option>
    <option>ASCII</option>
    <option>ISO8859_1</option>
    <option>CP1252</option>
    <option>SJIS</option>
    <option>MS932</option>
    <option>EUC_CN</option>
    <option>Big5</option>
  </select>
  <br>
  <b>Enter a string</b>
</form>
The following four characters are not in all character encodings: ½àç+
**CharsetEncode**

**Description**

Uses the specified encoding to convert binary data to a string.

**Returns**

A string representation of the binary object.

**Category**

Conversion functions, String functions

**Function syntax**

`CharsetEncode(binaryobject, encoding)`

**See also**

`BinaryDecode, BinaryEncode, CharsetDecode`; “Determining the page encoding of server output” in Chapter 17, “Developing Globalized Applications,” in *ColdFusion MX Developer’s Guide*

**History**

ColdFusion MX 7: Added this function.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>binaryobject</code></td>
<td>A variable containing binary data to decode into text.</td>
</tr>
</tbody>
</table>
| `encoding` | The character encoding that was used to encode the string into binary format. It must be a character encoding name recognized by the Java runtime. The following list includes commonly used values:  
  • utf-8  
  • iso-8859-1  
  • windows-1252  
  • us-ascii  
  • shift_jis  
  • iso-2022-jp  
  • euc-jp  
  • euc-kr  
  • big5  
  • euc-cn  
  • utf-16  
For a complete list of character encoding names supported by Sun Java runtimes, see [http://java.sun.com/j2se/1.3/docs/guide/intl/encoding.doc.html](http://java.sun.com/j2se/1.3/docs/guide/intl/encoding.doc.html) and [http://java.sun.com/j2se/1.4/docs/guide/intl/encoding.doc.html](http://java.sun.com/j2se/1.4/docs/guide/intl/encoding.doc.html). |

**Usage**

Macromedia recommends that you use this function, and not the `ToString` function, to convert binary data to strings in all new applications.
Example

The following example uses the CharsetDecode function to convert a string from a form to a binary object, and uses the CharsetEncode function to convert it back to the original value. You can change the character encoding that ColdFusion uses for the conversion. Notice that if you select the Asian language encodings, characters that are not in the specified character set do get successfully converted.

<h3>Character Encoding Conversion Example</h3>

<!--- Do the following if the form has been submitted. --->
<cfif IsDefined("Form.myString")>

<!--- Do the conversions. --->
<cfscript>
  chardecode=CharsetDecode(Form.myString, Form.charEncoding);
  charencode=CharsetEncode(chardecode, Form.charEncoding);
</cfscript>

<!--- Display the input values and results. --->
<cfoutput>
  <h3>Parameter Settings</h3>
  <p><b>The string:</b> #Form.myString#</p>
  <p><b>The character encoding:</b> #Form.charEncoding#</p>
  <h3>Results of the operations:</h3>
  <p><b>Dump of the string converted to a binary object by CharsetDecode:</b>
  <cfdump var="#chardecode#"></p>
  <p><b>The binary object converted back to a string by CharsetEncode:</b>
  #charencode#</p>
</cfoutput>
</cfif>

<!--- The input form. --->
<form action="#CGI.SCRIPT_NAME#" method="post">
  <b>Select the character encoding</b><br>
  <!--- This is a subset, additional encodings are available. --->
  <select size="1" name="charEncoding">
    <option selected>UTF-8</option>
    <option>ASCII</option>
    <option>ISO8859_1</option>
    <option>CP1252</option>
    <option>SJIS</option>
    <option>MS932</option>
    <option>EUC_CN</option>
    <option>Big5</option>
  </select><br>
  <b>Enter a string</b><br>
</form>
The following four characters are not in all character encodings: ½àç÷

<input type="Submit" value="convert my data">
Chr

Converts a numeric value to a UCS-2 character.

Returns

A character with the specified UCS-2 character code.

Category

String functions

Function syntax

`Chr(number)`

See also

Asc

History

ColdFusion MX: Changed Unicode support: ColdFusion supports the Java UCS-2 representation of Unicode characters, up to a value of 65535. (Earlier releases supported 1-255.)

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>number</td>
<td>A value (a number in the range 0 to 65535, inclusive)</td>
</tr>
</tbody>
</table>

Usage

The values 0 – 31 are standard, nonprintable codes. For example:

- `Chr(10)` returns a linefeed character
- `Chr(13)` returns a carriage return character
- The two-character string `Chr(13) & Chr(10)` returns a Windows newline

Note: For a complete list of the Unicode characters and their codes, see [www.unicode.org/charts/](http://www.unicode.org/charts/).

Example

```html
<!--- If the character string is not empty, then output its Chr value. --->
<cfif IsDefined("form.charVals")>
  <cfoutput>#form.charVals# = #Chr(form.charVals)#</cfoutput>
</cfif>

<cfform action="#CGI.script_name#" method="POST">
  <p>Type an integer character code from 1 to 65535 <br>
  to see its corresponding character. <br>
  <cfinput type="Text" name="CharVals" range="1,65535" message="Please enter an integer from 1 to 65535"
  validate="integer"
  required="Yes"
  size="5"/>
</cfform>
```
CJustify

Description
Centers a string in a field length.

Returns
String, center-justified by adding spaces before or after the input parameter. If length is less than the length of the input parameter string, the string is returned unchanged.

Category
Display and formatting functions, String functions

Function syntax
CJustify(string, length)

See also
LJustify, RJustify

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>string</td>
<td>A string or a variable that contains one. May be empty. If it is a variable that is defined as a number, the function processes it as a string.</td>
</tr>
</tbody>
</table>
| length    | A positive integer or a variable that contains one. Length of field. Can be coded as:  
- A number; for example, 6  
- A string representation of a number; for example, “6”  
Any other value causes ColdFusion to throw an error. |

Example
<!--- This example shows how to use CJustify --->
<CFPARAM name = "jstring" DEFAULT = ">">
<cfif IsDefined("FORM.submit")>
<cfdump var="#FORM#">
<cfset jstring = CJustify("#FORM.justifyString#", 35)>
</cfif>
<html>
<head>
<title>CJustify Example</title>
</head>
<body>
<h3>CJustify</h3>
<p>Enter a string; it will be center-justified within the sample field.</p>
<form action = "cjustify.cfm" method="post">  
<p><input type = "Text" value = "<cfoutput>#jString#</cfoutput>" size = 35 name = "justifyString"></p>
<p><input type = "Submit" name = "submit">  
<Input type = "RESET"></p>
</form>
</body>
</html>
Compare

**Description**
Performs a case-sensitive comparison of two strings.

**Returns**
- -1, if *string1* is less than *string2*
- 0, if *string1* is equal to *string2*
- 1, if *string1* is greater than *string2*

**Category**
String functions

**Function syntax**
`Compare(string1, string2)`

**See also**
`CompareNoCase`, `Find`

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>string1</td>
<td>A string or a variable that contains one</td>
</tr>
<tr>
<td>string2</td>
<td>A string or a variable that contains one</td>
</tr>
</tbody>
</table>

**Usage**
Compares the values of corresponding characters in *string1* and *string2*.

**Example**

```cfml
<h3>Compare Example</h3>
<p>The compare function performs a <strong>case-sensitive</strong> 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>
</cfswitch>
```

Compare 517
<h3>This is the default case</h3>
</cfdefaultcase>
</cfswitch>
</cfif>
<form action = "compare.cfm" method="post">
<p>String 1
<input type = "Text" name = "string1">
</p>
<p>String 2
<input type = "Text" name = "string2">
</p>
<p><input type = "Submit" value = "Compare these Strings" name = ">
<input type = "RESET">  
</p>
</form>
**CompareNoCase**

**Description**
Performs a case-insensitive comparison of two strings.

**Returns**
An indicator of the difference:

- A negative number, if `string1` is less than `string2`
- 0, if `string1` is equal to `string2`
- A positive number, if `string1` is greater than `string2`

**Category**
String functions

**Function syntax**
`CompareNoCase(string1, string2)`

**See also**
`Compare`, `FindNoCase`;
“Ambiguous type expressions and strings” in Chapter 3, “Using ColdFusion Variables,” in *ColdFusion MX Developer’s Guide*

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>string1</td>
<td>A string or a variable that contains one</td>
</tr>
<tr>
<td>string2</td>
<td>A string or a variable that contains one</td>
</tr>
</tbody>
</table>

**Example**

```cfdx
<H3>CompareNoCase Example</H3>
This function performs a case-insensitive 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">String 1 is less than String 2</CFCASE>
    <CFCASE value="0">String 1 is equal to String 2</CFCASE>
    <CFCASE value="1">String 1 is greater than String 2</CFCASE>
<CFDEFAULTCASE>This is the default case</CFDEFAULTCASE>
</CFSWITCH>
</CFIF>
```

---

`CompareNoCase` 519
<FORM ACTION="comparenocase.cfm" METHOD="POST">
  <P>String 1</P>
  <BR><INPUT TYPE="Text" NAME="string1">
  <P>String 2</P>
  <BR><INPUT TYPE="Text" NAME="string2">
  <P><INPUT TYPE="Submit" VALUE="Compare these Strings" NAME=""></P>
  <INPUT TYPE="RESET">
</FORM>
Cos

Description
Calculates the cosine of an angle that is entered in radians.

Returns
A number; the cosine of the angle.

Category
Mathematical functions

Function syntax
Cos(number)

See also
ACos, Sin, ASin, Tan, Atn, Pi

Parameters
- number: Angle, in radians, for which to calculate the cosine

Usage
The range of the result is -1 to 1.

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

Note: Because the function uses floating point arithmetic, it returns a very small number (such as 6.12323399574E-017) for angles that should produce 0. To test for a 0 value, check whether the value is less than 0.0000000000001.

Example
<h3>Cos Example</h3>
<!--- Calculate cosine if form has been submitted --->
<cfif IsDefined("FORM.cosNum")>
<!--- Make sure input is a number --->
<cfif IsNumeric(#FORM.cosNum#)>
<!--- Convert degrees to radians, call the Cos function. --->
<cfset cosValue=#Cos((Form.cosNum * PI()) / 180)#>
<!--- 0.0000000000001 is the function's precision limit. If absolute value of returned cosine value is less, set result to 0 ---->
<cfif Abs(cosValue) LT 0.0000000000001>
<cfset cosValue=0>
</cfif>
<cfoutput>
Cos(#FORM.cosNum#) = #cosValue#<br>
</cfoutput>
</cfif>
</cfoutput>
<!--- If input is not a number, show an error message --->
<h4>You must enter a numeric angle in degrees.</h4>
<form action="#CGI.script_name#" method="post">
  Enter an angle in degrees to get its cosine:
  <br>
  <input type="text" name="cosNum" size="15">
  <br>
  <input type="submit" name=""&nbsp;&nbsp;
  <input type="reset" name=""&nbsp;&nbsp;
</form>
CreateDate

Description
Creates a date/time object.

Returns
A date/time value.

Category
Date and time functions

Function syntax
CreateDate(year, month, day)

See also
CreateDateTime, CreateODBCDate; “Date-time functions and queries when ODBC is not supported” in Chapter 3, “Using ColdFusion Variables” in ColdFusion MX Developer’s Guide

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>year</td>
<td>Integer in the range 0-9999. Integers in the range 0-29 are converted to 2000-2029. Integers in the range 30-99 are converted to 1930-1999. You cannot specify dates before AD 100.</td>
</tr>
<tr>
<td>month</td>
<td>Integer in the range 1 (January)-12 (December)</td>
</tr>
<tr>
<td>day</td>
<td>Integer in the range 1-31</td>
</tr>
</tbody>
</table>

Usage
CreateDate is a subset of CreateDateTime.

The time in the returned object is set to 00:00:00.

Example
<h3>CreateDate Example</h3>
<CFIF IsDefined("form.year")>
<p>Your date value, generated with CreateDate:
<CFSET yourDate = CreateDate(form.year, form.month, form.day)>
<cfoutput>
<ul>
<li>Formatted with CreateDate: #CreateDate(form.year, form.month, form.day)#
<li>Formatted with CreateDateTime: #CreateDateTime(form.year, form.month, form.day, 12,13,0)#
<li>Formatted with CreateODBCDate: #CreateODBCDate(yourDate)#
<li>Formatted with CreateODBCDateTime: #CreateODBCDateTime(yourDate)#
</ul>
<p>The same value can be formatted with DateFormat:
<ul>
<li>Formatted with CreateDate and DateFormat:
  #DateFormat(CreateDate(form.year, form.month, form.day), "mmm-dd-yyyy")#</li>
<li>Formatted with CreateDateTime and DateFormat:
  #DateFormat(CreateDateTime(form.year, form.month, form.day, 12, 13, 0))#</li>
<li>Formatted with CreateODBCDate and DateFormat:
  #DateFormat(CreateODBCDate(yourDate), "mmmm d, yyyy")#</li>
<li>Formatted with CreateODBCDateTime and DateFormat:
  #DateFormat(CreateODBCDateTime(yourDate), "d/m/yy")#</li>
</ul>
</cfoutput>
</CFIF>
</CFoutput>
</CFIFORM ACTION="createdate.cfm" METHOD="POST">
<p>Enter the year, month and day, as integers:
</p>
<pre>
Year<CFINPUT TYPE="Text" NAME="year" VALUE="1998" VALIDATE="integer" REQUIRED="Yes">
Month<CFINPUT TYPE="Text" NAME="month" VALUE="6" VALIDATE="integer" REQUIRED="Yes">
Day<CFINPUT TYPE="Text" NAME="day" VALUE="8" VALIDATE="integer" REQUIRED="Yes"></p>
</pre>
<p><INPUT TYPE="Submit" NAME=""> <INPUT TYPE="RESET"></p>
</cfiform>
CreateDateDateTime

**Description**

Creates a date-time object.

**Returns**

A date/time value.

**Category**

Date and time functions

**Function syntax**

`CreateDateDateTime(year, month, day, hour, minute, second)`

**See also**

`CreateDate, CreateTime, CreateODBCDateTime, Now`; “Date-time functions and queries when ODBC is not supported” in Chapter 3, “Using ColdFusion Variables,” in *ColdFusion MX Developer’s Guide*

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>year</td>
<td>Integer in the range 0-9999. Integers in the range 0-29 are converted to 2000-2029. Integers in the range 30-99 are converted to 1930-1999. You cannot specify dates before AD 100.</td>
</tr>
<tr>
<td>month</td>
<td>Integer in the range 1 (January)–12 (December)</td>
</tr>
<tr>
<td>day</td>
<td>Integer in the range 1–31</td>
</tr>
<tr>
<td>hour</td>
<td>Integer in the range 0–23</td>
</tr>
<tr>
<td>minute</td>
<td>Integer in the range 0–59</td>
</tr>
<tr>
<td>second</td>
<td>Integer in the range 0–59</td>
</tr>
</tbody>
</table>

**Example**

```html
<h3>CreateDateTime Example</h3>

<CFIF IsDefined("form.year")>
Your date value, generated with CreateDateTime:
<CFSET yourDate = CreateDateTime(form.year, form.month, form.day, form.hour, form.minute, form.second)>

<cfoutput>
<ul>
<li>Formatted with CreateDate: #CreateDate(form.year, form.month, form.day)#
<li>Formatted with CreateDateTime: #CreateDateTime(form.year, form.month, form.day, form.hour, form.minute, form.second)#
<li>Formatted with CreateODBCDate: #CreateODBCDate(yourDate)#
<li>Formatted with CreateODBCDateTime: #CreateODBCDateTime(yourDate)#
</ul>
</cfoutput>
```
The same value can be formatted with DateFormat:

- Formatted with CreateDate and DateFormat:
  ```coldf```
  #DateFormat(CreateDate(form.year, form.month, form.day), "mmm-dd-yyyy")#
  ```coldf```
- Formatted with CreateDateTime and DateFormat:
  ```coldf```
  #DateFormat(CreateDateTime(form.year, form.month, form.day, form.hour, form.minute, form.second))#
  ```coldf```
- Formatted with CreateODBCDate and DateFormat:
  ```coldf```
  #DateFormat(CreateODBCDate(yourDate), "mmmm d, yyyy")#
  ```coldf```
- Formatted with CreateODBCDateTime and DateFormat:
  ```coldf```
  #DateFormat(CreateODBCDateTime(yourDate), "d/m/yy")#
  ```coldf```

</cfoutput>
</CFIF>
</CFFORM>

<CFFORM ACTION="createdatetime.cfm" METHOD="POST">
<p>Please enter the year, month, and day, in integer format, for a date to view:
</p>
<pre>
Year<CFINPUT TYPE="Text" NAME="year" VALUE="1998" VALIDATE="integer"
REQUIRED="Yes">
Month<CFINPUT TYPE="Text" NAME="month" VALUE="6" RANGE="1,12"
MESSAGE="Please enter a month (1-12)" VALIDATE="integer"
REQUIRED="Yes">
Day<CFINPUT TYPE="Text" NAME="day" VALUE="8" RANGE="1,31"
MESSAGE="Please enter a day of the month (1-31)" VALIDATE="integer"
REQUIRED="Yes">
Hour<CFINPUT TYPE="Text" NAME="hour" VALUE="16" RANGE="0,23"
MESSAGE="You must enter an hour (0-23)" VALIDATE="integer"
REQUIRED="Yes">
Minute<CFINPUT TYPE="Text" NAME="minute" VALUE="12" RANGE="0,59"
MESSAGE="You must enter a minute value (0-59)" VALIDATE="integer"
REQUIRED="Yes">
Second<CFINPUT TYPE="Text" NAME="second" VALUE="0" RANGE="0,59"
MESSAGE="You must enter a value for seconds (0-59)" VALIDATE="integer"
REQUIRED="Yes">
</pre>

</cfoutput>
</CFFORM>
CreateObject

Description

Creates a ColdFusion object, of a specified type.

Returns

An object, of the specified type.

Note: You can enable and disable this function in the ColdFusion Administrator, ColdFusion Basic Security, Tag Restrictions page.

Category

Extensibility functions

History

ColdFusion MX 7: For web service object: added the portName parameter, which specifies a port named in the service element of the WSDL.

ColdFusion MX:

• Changed instantiation behavior: this function, and the cfobject tag, can instantiate ColdFusion components and web services. Executing operations on a CFC object executes CFML code that implements the CFC's method in the CFC file.
  For more information, see ColdFusion MX Developer's Guide.
• For CORBA object: changed the Naming Service separator format for addresses from a dot to a forward slash. For example, if "context=NameService", for a class, use either of the following formats for the class parameter:
  ■ "Macromedia/Eng/CF"
  ■ "Macromedia.current/Eng.current/CF"
  (In earlier releases, the format was "Macromedia.Eng.CF").
• For CORBA object: changed the locale parameter; it specifies the Java config that contains the properties file.

CreateObject object types

For information about using this function, see these sections:

• “CreateObject: COM object” on page 528
• “CreateObject: component object” on page 530
• “CreateObject: CORBA object” on page 531
• “CreateObject: Java or EJB object” on page 533
• “CreateObject: web service object” on page 534

Note: On UNIX, this function does not support COM objects.
CreateObject: COM object

Description
The CreateObject function can create a Component Object Model (COM) object.

To create a COM object, you must provide this information:

• The object’s program ID or filename
• The methods and properties available to the object through the IDispatch interface
• The arguments and return types of the object’s methods

For most objects, you can get this information from the OLEView utility.

Note: On UNIX, this function does not support COM objects.

Returns
A COM object.

Function syntax
CreateObject(type, class, context, serverName)

See also
ReleaseComObject, cfobject; Chapter 38, “Integrating COM and CORBA Objects in CFML Applications” in ColdFusion MX Developer’s Guide.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>Type of object to create.</td>
</tr>
<tr>
<td></td>
<td>• com</td>
</tr>
<tr>
<td></td>
<td>• corba</td>
</tr>
<tr>
<td></td>
<td>• java</td>
</tr>
<tr>
<td></td>
<td>• component</td>
</tr>
<tr>
<td></td>
<td>• webservice</td>
</tr>
<tr>
<td>class</td>
<td>Component ProgID for the object to invoke</td>
</tr>
<tr>
<td>context</td>
<td>• InProc</td>
</tr>
<tr>
<td></td>
<td>• Local</td>
</tr>
<tr>
<td></td>
<td>• Remote</td>
</tr>
<tr>
<td>serverName</td>
<td>Server name, using UNC or DNS convention, in one of these forms:</td>
</tr>
<tr>
<td></td>
<td>• \lanserver</td>
</tr>
<tr>
<td></td>
<td>• lanserver</td>
</tr>
<tr>
<td></td>
<td>• <a href="http://wwwservername.com">http://wwwservername.com</a></td>
</tr>
<tr>
<td></td>
<td>• wwwservername.com</td>
</tr>
<tr>
<td></td>
<td>• 127.0.0.1</td>
</tr>
<tr>
<td></td>
<td>If context = “remote”, this parameter is required.</td>
</tr>
</tbody>
</table>

528 Chapter 3: ColdFusion Functions
Usage

The following example creates the Windows Collaborative Data Objects (CDO) for NTS NewMail object to send mail. You would use this code within a cfscript tag.

Mailer = CreateObject("COM", "CDONTS.NewMail");
**CreateObject: component object**

**Description**

The `CreateObject` function can create an instance of a ColdFusion component (CFC) object.

**Returns**

A component object.

**Function syntax**

```
CreateObject(type, component-name)
```

**See also**

Chapter 10, “Building and Using ColdFusion Components” in *ColdFusion MX Developer’s Guide*

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>Type of object to create.</td>
</tr>
<tr>
<td>component-name</td>
<td>The CFC name; corresponds to the name of the file that defines the component; for example, use engineComp to specify the component defined in the engineComp.cfc file</td>
</tr>
</tbody>
</table>

**Usage**

On UNIX systems, ColdFusion searches first for a file with a name that matches the specified component name, but is all lowercase. If it does not find the file, it looks for a file name that matches the component name exactly, with the identical character casing.

In the following example, the CFScript statements assign the `tellTimeCFC` variable to the `tellTime` component using the `CreateObject` function. The `CreateObject` function references the component in another directory. To invoke component methods, you use function syntax.

```cfc
<b>Server's Local Time:</b>
<cfscript>
tellTimeCFC=CreateObject("component","appResources.components.tellTime");
tellTimeCFC.getLocalTime();
</cfscript>
<br>
<b>Calculated UTC Time:</b>
<cfscript>
tellTimeCFC.getUTCTime();
</cfscript>
```
CreateObject: CORBA object

Description
The CreateObject function can call a method on a CORBA object. The object must be defined and registered for use.

Returns
A handle to a CORBA interface.

Function syntax
CreateObject(type, context, class, locale)

See also
Chapter 38, "Integrating COM and CORBA Objects in CFML Applications" in ColdFusion MX Developer’s Guide

History
See the History section of the main CreateObject function page.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>Type of object to create.</td>
</tr>
<tr>
<td></td>
<td>• com</td>
</tr>
<tr>
<td></td>
<td>• corba</td>
</tr>
<tr>
<td></td>
<td>• java</td>
</tr>
<tr>
<td></td>
<td>• component</td>
</tr>
<tr>
<td></td>
<td>• webservice</td>
</tr>
<tr>
<td>context</td>
<td>• IOR: ColdFusion uses IOR to access CORBA server</td>
</tr>
<tr>
<td></td>
<td>• NameService: ColdFusion uses naming service to access server. Valid only with the InitialContext of a VisiBroker ORB.</td>
</tr>
<tr>
<td>class</td>
<td>• If context = 'ior': absolute path of file that contains string version of the Interoperable Object Reference (IOR). ColdFusion must be able to read file; it should be local to ColdFusion server or accessible on network</td>
</tr>
<tr>
<td></td>
<td>• If context = 'nameservice': forward slash-delimited naming context for naming service. For example: Allaire//Doc/empobject</td>
</tr>
<tr>
<td>locale</td>
<td>The name of the Java config that holds the properties file. For more information, see Configuring and Administering ColdFusion MX.</td>
</tr>
</tbody>
</table>

Usage
In the class parameter, if "context=NameService", use a dot separator for the first part of the string. Use either of the following formats:
• "Macromedia/Eng/CF"
• "Macromedia.current/Eng.current/CF"
ColdFusion Enterprise supports CORBA through the Dynamic Invocation Interface (DII). To use this function with CORBA objects, you must provide the name of the file that contains a string version of the IOR, or the object’s naming context in the naming service. You must provide the object’s attributes, method names and method signatures.

This function supports user-defined types (structures, arrays, and sequences).

Example

```pascal
myobj = CreateObject("corba", "d:\temp\tester.ior", "ior", "visibroker") // uses IOR

myobj = CreateObject("corba", "Macromedia/Eng/CF", "nameservice", "visibroker") // uses nameservice

myobj = CreateObject("corba", "d:\temp\tester.ior", "nameservice") // uses nameservice and default configuration
```
CreateObject: Java or EJB object

Description

The `CreateObject` function can create a Java object, and, by extension, an EJB object.

Returns

A Java object.

Function syntax

`CreateObject(type, class)`

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>Type of object to create.</td>
</tr>
<tr>
<td>com</td>
<td>A Java class name</td>
</tr>
<tr>
<td>corba</td>
<td></td>
</tr>
<tr>
<td>java</td>
<td></td>
</tr>
<tr>
<td>component</td>
<td></td>
</tr>
<tr>
<td>webservice</td>
<td></td>
</tr>
<tr>
<td>class</td>
<td></td>
</tr>
</tbody>
</table>

Usage

Any Java class available in the class path that is specified in the ColdFusion Administrator can be loaded and used from ColdFusion with the `CreateObject` function.

To access Java methods and fields:

1. Call the `CreateObject` function or the `cfobject` tag to load the class.
2. Use the `init` method, with appropriate arguments, to call an instance of the class. For example:
   `<cfset ret = myObj.init(arg1, arg2)>`

Calling a public method on the object without first calling the "init" method invokes a static method. Arguments and return values can be any Java type (simple, array, object). If strings are passed as arguments, ColdFusion does the conversions; if strings are received as return values, ColdFusion does no conversion.

Overloaded methods are supported if the number of arguments is different. Future enhancements will let you use cast functions that allow method signatures to be built more accurately.
CreateObject: web service object

Description
This function can create a web service object.

Returns
A web service object.

Function syntax
CreateObject(type, urltowsdl [, portname ])

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>Type of object to create.</td>
</tr>
<tr>
<td></td>
<td>• com</td>
</tr>
<tr>
<td></td>
<td>• corba</td>
</tr>
<tr>
<td></td>
<td>• java</td>
</tr>
<tr>
<td></td>
<td>• component</td>
</tr>
<tr>
<td></td>
<td>• webservice</td>
</tr>
<tr>
<td>urltowsdl</td>
<td>WSDL file URL; location of web service</td>
</tr>
<tr>
<td>portname</td>
<td>The port name for the web service. This value is case-sensitive and corresponds to the port element's name attribute under the service element. Specify this parameter if the web service contains multiple ports. If no port name is specified, ColdFusion uses the first port found in the WSDL.</td>
</tr>
</tbody>
</table>

Usage
You can use the CreateObject function to create a web service.

Example
<cfscript>
    ws = CreateObject("webservice", 
        "http://www.xmethods.net/sd/2001/TemperatureService.wsdl");
    xlatstring = ws.getTemp(zipcode = "55907");
    writeoutput("The temperature at 55907 is " & xlatstring);
</cfscript>
**CreateODBCDate**

**Description**
 creates an ODBC date object.

**Returns**
 a date object, in normalized ODBC date format.

**Category**
 Date and time functions

**Function syntax**
 CreateODBCDate(date)

**See also**
CreateDate, CreateODBCDateTime

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>date</td>
<td>Date or date/time object in the range 100 AD–9999 AD.</td>
</tr>
</tbody>
</table>

**Usage**

This function does not parse or validate values. To ensure that dates are entered and processed correctly (for example, to ensure that a day/month/year entry is not confused with a month/day/year entry, and so on), Macromedia recommends that you parse entered dates with the DateFormat function, using the mm-dd-yyyy mask, into three elements. Ensure that values are within appropriate ranges; for example, to validate a month value, use the attributes validate = "integer" and range = "1,12".

**Example**

```
<h3>CreateODBCDate Example</h3>
<CFIF IsDefined("form.year")>
<p>Your date value, generated with CreateDateTime:
<cfset yourDate = CreateDateTime(form.year, form.month, form.day, form.hour, form.minute, form.second)>
<cfoutput>
<ul>
<li>Formatted with CreateDate: #CreateDate(form.year, form.month, form.day)#
<li>Formatted with CreateDateTime: #CreateDateTime(form.year, form.month, form.day, form.hour, form.minute, form.second)#
<li>Formatted with CreateODBCDate: #CreateODBCDate(yourDate)#
<li>Formatted with CreateODBCDateTime: #CreateODBCDateTime(yourDate)#
</ul>
<p>The same value can be formatted with DateFormat:
<cfoutput>
<ul>
<li>Formatted with CreateDate and DateFormat: #DateFormat(CreateDate(form.year,form.month, form.day), "mmm-dd-yyyy")#
<li>Formatted with CreateDateTime and DateFormat: #DateFormat(CreateDateTime(form.year,form.month, form.day), "mmm-dd-yyyy")#
</cfoutput>
```
#DateFormat(CreateDateTime(form.year, form.month, form.day, form.hour, form.minute, form.second))#

<li>Formatted with CreateODBCDate and DateFormat:
#DateFormat(CreateODBCDate(yourDate), "mmmm d, yyyy")#
</li>

<li>Formatted with CreateODBCDateTime and DateFormat:
#DateFormat(CreateODBCDateTime(yourDate), "d/m/yy")#
</li>

</cfoutput>
</cfif>
.cfiform action="createodbcdate.cfm" method="POST">
<p>Enter the year, month and day, as integers:
<pre>
Year <cfinput type="text" name="year" value="1998" validate="integer"
required="yes">
Month<cfinput type="text" name="month" value="6" range="1,12"
message="please enter a month (1-12)" validate="integer"
REQUIRED="Yes">
Day <cfinput type="text" name="day" value="8" range="1,31"
MESSAGE="Please enter a day of the month (1-31)" VALIDATE="integer"
REQUIRED="Yes">
Hour <cfinput type="text" name="hour" value="16" range="0,23"
MESSAGE="You must enter an hour (0-23)" VALIDATE="integer"
REQUIRED="Yes">
Minute<cfinput type="text" name="minute" value="12" range="0,59"
MESSAGE="You must enter a minute value (0-59)" VALIDATE="integer"
REQUIRED="Yes">
Second<cfinput type="text" name="second" value="0" range="0,59"
MESSAGE="You must enter a value for seconds (0-59)" VALIDATE="integer"
REQUIRED="Yes">
</pre>
<p><INPUT TYPE="Submit" NAME=""> <INPUT TYPE="RESET">
</cfiform>
CreateODBCDateTime

Description

Creates an ODBC date-time object.

Returns

A date-time object, in ODBC timestamp format.

Category

Date and time functions

Function syntax

CreateODBCDateTime (date)

See also

CreateDate, CreateODBCDate, CreateODBCTime, Now; “Evaluation and type conversion issues” in Chapter 3, “Using ColdFusion Variables,” in ColdFusion MX Developer’s Guide

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>date</td>
<td>Date-time object in the range 100 AD–9999 AD.</td>
</tr>
</tbody>
</table>

Usage

When passing a date-time value as a string, you must enclose it in quotation marks. Otherwise, it is interpreted as a number representation of a date-time object.

Example

<!--- This example shows how to use CreateDate, CreateDateTime, CreateODBCDate, and CreateODBCDateTime --->

<h3>CreateODBCDateTime Example</h3>

<cfif IsDefined("form.year")>

Your date value, generated using CreateDateTime:
<cfset yourDate = CreateDateTime (form.year, form.month, form.day, form.hour, form.minute, form.second)>
<cfoutput>
<ul>
<li>Formatted with CreateDate: #CreateDate(form.year, form.month, form.day)#
<li>Formatted with CreateDateTime: #CreateDateTime(form.year, form.month, form.day, form.hour, form.minute, form.second)#
<li>Formatted with CreateODBCDate: #CreateODBCDate(yourDate)#
<li>Formatted with CreateODBCDateTime: #CreateODBCDateTime(yourDate)#
</ul>
<p>The same value can be formatted with DateFormat:
<ul>
<li>Formatted with CreateDate and DateFormat: #DateFormat(CreateDate(form.year, form.month, form.day), "mmm-dd-yyyy")#
<li>Formatted with CreateDateTime and DateFormat: #DateFormat(CreateDateTime(form.year, form.month, form.day, form.hour, form.minute, form.second))#
</ul>

</cfif>
<li>Formatted with CreateODBCDate and DateFormat:
   #DateFormat(CreateODBCDate(yourDate), "mmmm d, yyyy")#
</li>
<li>Formatted with CreateODBCDateTime and DateFormat:
   #DateFormat(CreateODBCDateTime(yourDate), "d/m/yy")#
</li>
</ul>
</cfoutput>
</cfif>

<cfoutput>
</cfif>

<cfform action="createodbcdatetime.cfm" method="post">

<p>Enter a year, month and day, as integers:
<pre>
Year <cfinput type="text" name="year" value="1998" valida/*" integer" required="yes"/>
Month <cfinput type="text" name="month" value="6" range="1,12" message="Enter a month (1-12)" required="yes"/>
Day <cfinput type="text" name="day" value="8" range="1,31" message="Enter a day of the month (1-31)" required="yes"/>
Hour <cfinput type="text" name="hour" value="16" range="0,23" message="You must enter an hour (0-23)" required="yes"/>
Minute <cfinput type="text" name="minute" value="12" range="0,59" message="You must enter a minute value (0-59)" required="yes"/>
Second <cfinput type="text" name="second" value="0" range="0,59" message="You must enter a seconds value (0-59)" required="yes"/>
</pre>
<p><input type="submit" name=""/> <input type="reset" name=""></p>
</cfform>
**CreateODBCTime**

**Description**

Creates an ODBC time object.

**Returns**

A time object, in ODBC timestamp format.

**Category**

Date and time functions

**Function syntax**

```latex
CreateODBCTime(date)
```

**See also**

CreateODBCDateTime,CreateTime, “Evaluation and type conversion issues” in Chapter 3, “Using ColdFusion Variables,” in *ColdFusion MX Developer's Guide*

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>date</td>
<td>Date/time object in the range 100 AD–9999 AD.</td>
</tr>
</tbody>
</table>

**Usage**

When passing a date-time value as a string, you must enclose it in quotation marks. Otherwise, it is interpreted as a number representation of a date-time object.

**Example**

```html
<h3>CreateODBCTime Example</h3>
<cfif IsDefined("form.hour")>
    Your time value, created with CreateTime...
    <cfset yourTime = CreateTime(form.hour, form.minute, form.second)>
    <cfoutput>
        <ul>
            <li>Formatted with CreateODBCTime: #CreateODBCTime(yourTime)#</li>
            <li>Formatted with TimeFormat: #TimeFormat(yourTime)#</li>
        </ul>
    </cfoutput>
</cfif>

<cfform action="createodbctime.cfm" method="post">
    <pre>
    Hour <cfinput type="Text" name="hour" value="16" range="0,23" message="You must enter an hour (0-23)" validate="integer" required="Yes">
    Minute<cfinput type="Text" name="minute" value="12" range="0,59" message="You must enter a minute value (0-59)" validate="integer" required="yes">
    second<cfinput type="text" name="second" value="0" range="0,59" message="You must enter a value for seconds (0-59)" validate="integer" required="yes">
    </pre>
</cfform>
```

CreateODBCTime 539
CreateTime

Description
Creates a time variable.

Returns
A time variable.

Category
Date and time functions

Function syntax
CreateTime(hour, minute, second)

See also
CreateODBCTime, CreateDateTime; “Evaluation and type conversion issues” in Chapter 3, “Using ColdFusion Variables,” in ColdFusion MX Developer’s Guide

Parameters
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hour</td>
<td>Number in the range 0–23</td>
</tr>
<tr>
<td>minute</td>
<td>Number in the range 0–59</td>
</tr>
<tr>
<td>second</td>
<td>Number in the range 0–59</td>
</tr>
</tbody>
</table>

Usage
CreateTime is a subset of CreateDateTime.

A time variable is a special case of a date-time variable. The date part of a time variable is set to December 30, 1899.

Example
<h3>CreateTime Example</h3>
<cfif IsDefined("FORM.hour")>
<cfset yourTime = CreateTime(FORM.hour, FORM.minute, FORM.second)>
<cfoutput><ul>
<li>Formatted with timeFormat: #TimeFormat(yourTime)#
<li>Formatted with timeFormat and hh:mm:ss: #TimeFormat(yourTime, 'hh:mm:ss')#
</ul></cfoutput>
</cfif>

<!-- CFFORM action="createTime.cfm" METHOD="post" -->
<!-- PRE Hour <CFINPUT TYPE="Text" NAME="hour" VALUE="16" RANGE="0,23" MESSAGE="You must enter an hour (0-23)" VALIDATE="integer" REQUIRED="Yes"> Minute <cfinput type="text" name="minute" value="12" range="0,59" MESSAGE="You must enter a minute value (0-59)" VALIDATE="integer" REQUIRED="Yes"> Second <cfinput type="text" name="second" value="0" range="0,59" -->
MESSAGE="You must enter a value for seconds (0-59)" VALIDATE="integer"
REQUIRED="Yes">
</PRE>
<p><input type="submit" name="" /> <input type="reset">
</cform>
CreateTimeSpan

Description

Creates a date/time object that defines a time period. You can add or subtract it from other date-time objects and use it with the cachedWithin attribute of cfquery.

Returns

A date-time object.

Category

Date and time functions

Function syntax

CreateTimeSpan(days, hours, minutes, seconds)

See also

CreateDateTime, DateAdd, DateConvert; “Defining application-level settings and variables” in Chapter 13, “Designing and Optimizing a ColdFusion Application,” in ColdFusion MX Developer’s Guide

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>days</td>
<td>Integer in the range 0–32768; number of days in time period</td>
</tr>
<tr>
<td>hours</td>
<td>Number of hours in time period</td>
</tr>
<tr>
<td>minutes</td>
<td>Number of minutes in time period</td>
</tr>
<tr>
<td>seconds</td>
<td>Number of seconds in time period</td>
</tr>
</tbody>
</table>

Usage

Creates a special date-time object that should be used only to add and subtract from other date-time objects or with the cfquery cachedWithin attribute.

If you use the cachedWithin attribute of cfquery, and the original query date falls within the time span you define, cached query data is used. In this case, the CreateTimeSpan function is used to define a period of time from the present backwards. The cachedWithin attribute takes effect only if you enable query caching in the ColdFusion Administrator. For more information, see cfquery.

Example

<!--- This example shows the use of CreateTimeSpan with cfquery --->
<h3>CreateTimeSpan Example</h3>
<!--- define startrow and maxrows to facilitate 'next N' style browsing --->
<cfparam name = "MaxRows" default = "10">
<cfparam name = "StartRow" default = "1">
<!--- Query database for information, if cached database information has not been updated in the last six hours. -------->
<cfoutput>
<cfquery name = "GetParks" datasource = "cfdocexamples"
```xml
cachedWithin = "#CreateTimeSpan(0, 6, 0, 0)#">
SELECT PARKNAME, REGION, STATE
FROM Parks
ORDER BY ParkName, State
</cfquery>
</cfoutput>
<cfoutput query="GetParks" StartRow="#StartRow#" maxrows="#maxrows#">
<table>
<thead>
<tr>
<th>Park Name</th>
<th>Region</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>#GetParks.CurrentRow#</td>
<td>#Region#</td>
<td>#State#</td>
</tr>
</tbody>
</table>
</cfoutput>
<cfif (StartRow + MaxRows) LTE GetParks.RecordCount>
<a href="cfquery.cfm?startrow=#StartRow + MaxRows#">See next #MaxRows# rows</a>
</cfif>
</table>
```

```xml
createTimeSpan 543
```
CreateUUID

Description

Creates a Universally Unique Identifier (UUID). A UUID is a 35-character string representation of a unique 128-bit integer.

Returns

A ColdFusion format UUID, in the format xxxxxxxx-xxxx-xxxx-xxxxxxxxxxxxxxxx, where x is a hexadecimal digit (0-9 or A-F). (The character groups are 8-4-4-16.)

Category

Other functions

Function syntax

CreateUUID()

Usage

The ColdFusion UUID generation algorithm uses the unique time-of-day value, the IEEE 802 Host ID, and a cryptographically strong random number generator to generate UUIDs that conform to the principles laid out in the draft IEEE RFC "UUIDs and GUIDs."

The ColdFusion UUID format is as follows:

xxxxxxxx-xxxx-xxxx-xxxxxxxxxxxxxxxx (8-4-4-16).

This does not conform to the Microsoft/DCE standard, which is as follows:

xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx (8-4-4-4-12)

There are UUID test tools and a user-defined function called CreateGUID, which converts CFML UUIDs to UUID/Microsoft GUID format, available on the web at www.cflib.org.

Use this function to generate a persistent identifier in a distributed environment. To a very high degree of certainty, this function returns a unique value; no other invocation on the same or any other system returns the same value.

UUIDs are used by distributed computing frameworks, such as DCE/RPC, COM+, and CORBA. In ColdFusion, you can use UUIDs as primary table keys for applications in which data is stored in shared databases. In such cases, using numeric keys can cause primary-key constraint violations during table merges. Using UUIDs, you can eliminate these violations.

Example

<h3>CreateUUID Example</h3>

This example uses CreateUUID to generate a UUID when you submit the form.

You can submit the form more than once. 

<!--- Checks whether the form was submitted; if so, creates UUID. --->
<cfif IsDefined("Form.CreateUUID") Is True>
<hr>
<p>Your new UUID is: <cfoutput>#CreateUUID()#</cfoutput></p>
</cfif>
<form action = "createuuid.cfm">
<p><input type = "Submit" name = "CreateUUID" /></p>
</form>
DateAdd

Description

Adds units of time to a date.

Returns

A date/time object.

Category

Date and time functions

Function syntax

DateAdd("datepart", number, "date")

See also

DateConvert, DatePart, CreateTimeSpan

History

ColdFusion MX 6.1: Added the datepart character L or l to represent milliseconds.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>datepart</td>
<td>String:</td>
</tr>
<tr>
<td></td>
<td>• yyyy: Year</td>
</tr>
<tr>
<td></td>
<td>• q: Quarter</td>
</tr>
<tr>
<td></td>
<td>• m: Month</td>
</tr>
<tr>
<td></td>
<td>• y: Day of year</td>
</tr>
<tr>
<td></td>
<td>• d: Day</td>
</tr>
<tr>
<td></td>
<td>• w: Weekday</td>
</tr>
<tr>
<td></td>
<td>• ww: Week</td>
</tr>
<tr>
<td></td>
<td>• h: Hour</td>
</tr>
<tr>
<td></td>
<td>• n: Minute</td>
</tr>
<tr>
<td></td>
<td>• s: Second</td>
</tr>
<tr>
<td></td>
<td>• l: Millisecond</td>
</tr>
<tr>
<td>number</td>
<td>Number of units of datepart to add to date (positive, to get dates in the future; negative, to get dates in the past). Number must be an integer.</td>
</tr>
<tr>
<td>date</td>
<td>Date/time object, in the range 100 AD–9999 AD.</td>
</tr>
</tbody>
</table>

Usage

The datepart specifiers y, d, and w add a number of days to a date.

When passing a date/time object as a string, you must enclose it in quotation marks. Otherwise, it is interpreted as a numeric representation of a date/time object.

Example

<!--- This example shows the use of DateAdd --->
<cfparam name="value" default="70">
<cfparam name="type" default="m">

DateAdd 545
<!--- if numbers passed, then use those --->
<cfif IsDefined("form.value")>
   <cfset value = form.value>
</cfif>
<cfif IsDefined("form.type")>
   <cfset type = form.type>
</cfif>

<cfquery name="GetMessages" datasource="cfdocexamples">
SELECT   UserName, Subject, Posted
FROM     Messages
</cfquery>

<p>This example uses DateAdd to determine when a message in
the database will expire. Currently, messages older
than <cfoutput>#value#</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="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>
are expired.

<table>
<tr>
   <td>UserName</td>
   <td>Subject</td>
   <td>Posted</td>
</tr>
<cfoutput query="GetMessages">
<tr>
   <td>#UserName#</td>
   <td>#Subject#</td>
   <td>#Posted# <cfif DateAdd(type, value, posted) LT Now()><font color="red">EXPIRED</font></cfif></td>
</tr>
</cfoutput>
</table>

<cfform action="#CGI.Script_Name#" method="post">
Select an expiration value:
<cfinput type="Text" name="value" value="#value#" message="Please enter whole numbers only" validate="integer" required="Yes">
<select name="type"
<option value="yyyy">years</option>
<option value="m" selected>months</option>
<option value="d">days</option>
<option value="ww">weeks</option>
<option value="h">hours</option>
<option value="n">minutes</option>
<option value="s">seconds</option>
</select>

<input type="Submit" value="Submit"/>
</cform>
DateCompare

Description
Performs a full date/time comparison of two dates.

Returns
• -1, if date1 is earlier than date2
• 0, if date1 is equal to date2
• 1, if date1 is later than date2

Category
Date and time functions

Function syntax
DateCompare("date1", "date2" [, "datePart"])

See also
CreateDateDateTime, DatePart

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>date1</td>
<td>Date/time object, in the range 100 AD–9999 AD.</td>
</tr>
<tr>
<td>date2</td>
<td>Date/time object, in the range 100 AD–9999 AD.</td>
</tr>
<tr>
<td>datePart</td>
<td>Optional. String. Precision of the comparison.</td>
</tr>
<tr>
<td></td>
<td>• s Precise to the second (default)</td>
</tr>
<tr>
<td></td>
<td>• n Precise to the minute</td>
</tr>
<tr>
<td></td>
<td>• h Precise to the hour</td>
</tr>
<tr>
<td></td>
<td>• d Precise to the day</td>
</tr>
<tr>
<td></td>
<td>• m Precise to the month</td>
</tr>
<tr>
<td></td>
<td>• yyyy Precise to the year</td>
</tr>
</tbody>
</table>

Usage
When passing a date/time object as a string, you must enclose it in quotation marks. Otherwise, it is interpreted as a numeric representation of a date/time object.

Example
<h3>DateCompare Example</h3>
<p>The DateCompare function compares two date/time values.
<cfif IsDefined("FORM.date1")>
    <cfif IsDate(FORM.date1) and IsDate(FORM.date2)>
        <cfset comparison = DateCompare(FORM.date1, FORM.date2, FORM.precision)>
    </cfif>
</cfif>

<!--- switch on the variable to give various responses --->
<cfswitch expression = #comparison#>
    <cfcase value = "-1"> <h3>#DateFormat(FORM.date1)# #TimeFormat(FORM.date1)#</h3> (Date 1) is earlier than #DateFormat(FORM.date2)#</cfcase>
</cfswitch>
<cfswitch>
<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>
</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>
</form>
</option>
</select>
<p><input type="Submit" value="Compare these dates" name="">
<input type="reset">
</form>
DateConvert

Description
Converts local time to Coordinated Universal Time (UTC), or UTC to local time. The function
uses the daylight savings settings in the executing computer to compute daylight savings time, if
required.

Returns
UTC- or local-formatted time object.

Category
Date and time functions

Function syntax
DateConvert("conversion-type", "date")

See also
GetTimeZoneInfo, CreateDateTime, DatePart

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| conversion-type | • local2Utc: Converts local time to UTC time.  
• utc2Local: Converts UTC time to local time. |
| date | Date and time string or a variable that contains one. To create, use CreateDateTime. |

Usage
When passing a date/time object as a string, you must enclose it in quotation marks. Otherwise, it
is interpreted as a numeric representation of a date/time object.

Note: You can pass the CreateDate or Now function as the date parameter of this function; for
example: #DateConvert(CreateDate(2001, 3, 3))#

Example
<h3>DateConvert Example</h3>
<!--- This shows conversion of current date - time to UTC and back. --->
<cfset curDate = Now()>
<p>The current date and time: #curDate#. </p>
<cfset utcDate = DateConvert("local2utc", curDate)>
<p>The current date and time converted to UTC time: #utcDate#.</p>
<!--- This code checks whether form was submitted. If so, the code generates
the CFML date with the CreateDateTime function. --->
<cfif IsDefined("FORM.submit")>
<cfset yourDate = CreateDateTime(FORM.year, FORM.month, FORM.day,  
FORM.hour, FORM.minute, FORM.second)>
<p>Your date value, presented as a ColdFusion date/time string:#yourdate#.</p>
</cfoutput>
<cfset yourUTC = DateConvert("local2utc", yourDate)>
<p><cfoutput>Your date and time value, converted to Coordinated Universal Time (UTC): #yourUTC#.</cfoutput></p>
<p><cfoutput>Your UTC date and time, converted back to local date and time: #DateConvert("utc2local", yourUTC)#.</cfoutput></p>
<cfelse>
Type the date and time, and press Enter to see the conversion.
</cfif>
<Hr size = "2" color = "#0000A0">
<form action = "dateconvert.cfm">
<p>Enter year, month and day in integer format for date value to view:
<table cellspacing = "2" cellpadding = "2" border = "0">
<tr><td>Year</td><td><input type = "Text" name = "year" value = "1998" validate = "integer" required = "Yes"></td></tr>
<tr><td>Month</td><td><input type = "Text" name = "month" value = "6" range = "1,12" message = "Enter a month (1-12)" validate = "integer" required = "Yes"></td></tr>
<tr><td>Day</td><td><input type = "Text" name = "day" value = "8" range = "1,31" message = "Enter a day of the month (1-31)" validate = "integer" required = "Yes"></td></tr>
<tr><td>Hour</td><td><input type = "Text" name = "hour" value = "16" range = "0,23" message = "You must enter an hour (0-23)" validate = "integer" required = "Yes"></td></tr>
<tr><td>Minute</td><td><input type = "Text" name = "minute" value = "12" range = "0,59" message = "You must enter a minute value (0-59)" validate = "integer" required = "Yes"></td></tr>
<tr><td>Second</td><td><input type = "Text" name = "second" value = "0" range = "0,59" message = "You must enter a value for seconds (0-59)" validate = "integer" required = "Yes"></td></tr>
<tr><td><input type = "Submit" name = "submit" value = "Submit"></td><td><input type = "RESET"></td></tr>
</table>
</form>
DateDiff

Description
Determines the integer number of units by which \textit{date1} is less than \textit{date2}.

Returns
A number of units, of type \textit{datepart}.

Category
\textit{Date and time functions}

Function syntax
\texttt{DateDiff("datepart", "date1", "date2")}

See also
\texttt{DateAdd,.DatePart,CreateTimeSpan}

History
ColdFusion MX:
- Changed how negative date differences are calculated: this function calculates negative date differences correctly; its output may be different from that in earlier releases.
- Changed the \texttt{w} and \texttt{ww} masks; they determine the number of full weeks between the two dates.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>datepart</td>
<td>String specifying the units in which to count; for example yyyy requests a date difference in whole years.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• yyy: Years</td>
</tr>
<tr>
<td></td>
<td>• q: Quarters</td>
</tr>
<tr>
<td></td>
<td>• m: Months</td>
</tr>
<tr>
<td></td>
<td>• y: Days of year (same as d)</td>
</tr>
<tr>
<td></td>
<td>• d: Days</td>
</tr>
<tr>
<td></td>
<td>• w: Weekdays (same as ww)</td>
</tr>
<tr>
<td></td>
<td>• ww: Weeks</td>
</tr>
<tr>
<td></td>
<td>• h: Hours</td>
</tr>
<tr>
<td></td>
<td>• n: Minutes</td>
</tr>
<tr>
<td></td>
<td>• s: Seconds</td>
</tr>
<tr>
<td>date1</td>
<td>Date/time object, in the range 100 AD–9999 AD.</td>
</tr>
<tr>
<td>date2</td>
<td>Date/time object, in the range 100 AD–9999 AD.</td>
</tr>
</tbody>
</table>

Usage
The \texttt{DateDiff} function determines the number of complete \textit{datepart} units between the two dates; for example, if the \textit{datepart} parameter is "m" and the dates differ by 55 days, the function returns 1.
Enclose string constant dates in quotation marks. If the text contains only numbers (such as 1932), and is not surrounded by quotation marks, ColdFusion interprets it as a date/time object, resulting in an incorrect value.

Example

```coldfusion
<cfif IsDefined("form.value")>
  <cfset value = form.value>
</cfif>
<cfif IsDefined("form.type")>
  <cfset type = form.type>
</cfif>

<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 dateparts between date1 and date2.
    <cfswitch expression = "#form.type#">
      <cfcase value="yyyy">years</cfcase>
      <cfcase value="q">quarters</cfcase>
      <cfcase value="m">months</cfcase>
      <cfcase value="y">days</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>
    <cfif DateCompare("#form.date1#","#form.date2#") is not 0>
      <p>The difference is <cfoutput>#Abs(DateDiff(type, form.date2, form.date1))#</cfoutput> <cfswitch expression = "#form.type#">
        <cfcase value="yyyy">years</cfcase>
        <cfcase value="q">quarters</cfcase>
        <cfcase value="m">months</cfcase>
        <cfcase value="y">days</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>.
    </cfif>
    <cfelse>
      <p>The two dates are equal! Try changing one of the values ...
    </cfif>
</cfif>
```
<p>Please enter two valid date/time values, formatted like this:
<cfoutput>#DateFormat(Now())#</cfoutput>
</cfif>
</cfif>
<form action="index.cfm" method="post">
<pre>
Date 1
<input type="Text" name="date1" value="<CFOUTPUT>#DateFormat(Now())#</CFOUTPUT>">
Date 2
<input type="Text" name="date2" value="<CFOUTPUT>#DateFormat(Now())#</CFOUTPUT>">
What kind of unit to show difference?
<select name="type">
<option value="yyyy" selected>years
<option value="q">quarters
<option value="m">months
<option value="y">days of year
<option value="d">days
<option value="w">weekdays
<option value="ww">weeks
<option value="h">hours
<option value="n">minutes
<option value="s">seconds
</select>
</pre>
<input type="Submit" name=""/><input type="RESET">
</form>
**DateFormat**

**Description**

Formats a date value using U.S. date formats. For international date support, use `LDateFormat`.

**Returns**

A text string representing the date formatted according to the mask. If no mask is specified, returns the value in `dd-mmm-yy` format.

**Category**

Date and time functions

**Function syntax**

```
DateFormat("date", "mask")
```

**See also**

`Now`, `CreateDate`, `LDateFormat`, `LSParseDateTime`, `LSTimeFormat`, `TimeFormat`, `ParseDateTime`

**History**

ColdFusion MX: Added support for the following `mask` parameter options: `short`, `medium`, `long`, and `full`.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>date</td>
<td>Date/time object, in the range 100 AD–9999 AD.</td>
</tr>
</tbody>
</table>
| mask      | Characters that show how ColdFusion displays a date:  
|           | • d: Day of the month as digits; no leading zero for single-digit days.  
|           | • dd: Day of the month as digits; leading zero for single-digit days.  
|           | • dddd: Day of the week as a three-letter abbreviation.  
|           | • ddddd: Day of the week as its full name.  
|           | • m: Month as digits; no leading zero for single-digit months.  
|           | • mm: Month as digits; leading zero for single-digit months.  
|           | • mmm: Month as a three-letter abbreviation.  
|           | • mmmm: Month as its full name.  
|           | • yy: Year as last two digits; leading zero for years less than 10.  
|           | • yyyy: Year represented by four digits.  
|           | • gg: Period/era string. Ignored. Reserved.  
|           | The following masks tell how to format the full date and cannot be combined with other masks:  
|           | • short: equivalent to m/d/y  
|           | • medium: equivalent to mmm d, yyyy  
|           | • long: equivalent to mmmm d, yyyy  
|           | • full: equivalent to dddd, mmmm d, yyyy |
Usage

When passing a date/time object as a string, you must enclose it in quotation marks. Otherwise, it is interpreted as a numeric representation of a date/time object.

**Note:** You can pass the `CreateDate` function or the `Now` function as the `date` parameter of this function; for example: `#DateFormat(CreateDate(2001, 3, 3))#`

Date and time values in database query results can vary in sequence and formatting unless you use functions to format them. To ensure that application users correctly understand displayed dates and times, Macromedia recommends that you use this function and the `LSDateFormat`, `TimeFormat`, and `LSTimeFormat` functions to format resultset values. For more information and examples, see TechNote 22183, "ColdFusion Server (5 and 4.5.x) with Oracle: Formatting Date and Time Query Results," on the Macromedia website at [www.coldfusion.com/Support/KnowledgeBase/SearchForm.cfm](http://www.coldfusion.com/Support/KnowledgeBase/SearchForm.cfm).

**Note:** The `DateFormat` function is best used for formatting output, not for formatting input. For formatting input, use one of the date/time creation functions (for example, `CreateDate`) instead.

Example

```cf
<cfset todayDate = Now()>
<body>
<h3>DateFormat Example</h3>
<p>Today's date is <cfoutput>#todayDate#</cfoutput>.</p><p>Using DateFormat, we can display that date in different ways:</p><cfoutput>
<ul>
  <li>#DateFormat(todayDate)##
  <li>#DateFormat(todayDate, "mmm-dd-yyyy")#
  <li>#DateFormat(todayDate, "mmm d, yyyy")#
  <li>#DateFormat(todayDate, "mm/dd/yyyy")#
  <li>#DateFormat(todayDate, "d-mmm-yyyy")#
  <li>#DateFormat(todayDate, "ddd, mmmm dd, yyyy")#
  <li>#DateFormat(todayDate, "short")#
  <li>#DateFormat(todayDate, "medium")#
  <li>#DateFormat(todayDate, "long")#
  <li>#DateFormat(todayDate, "full")#
</ul>
</cfoutput>
</body>
```
DatePart

Description
Extracts a part from a date value.

Returns
Part of a date, as an integer.

Category
Date and time functions

Function syntax
DatePart("datepart", "date")

See also
DateAdd, DateConvert

History
ColdFusion MX 6.1: Added the datepart character L or l to represent milliseconds.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>datepart</td>
<td>String:</td>
</tr>
<tr>
<td></td>
<td>• yyyy: Year</td>
</tr>
<tr>
<td></td>
<td>• q: Quarter</td>
</tr>
<tr>
<td></td>
<td>• m: Month</td>
</tr>
<tr>
<td></td>
<td>• y: Day of year</td>
</tr>
<tr>
<td></td>
<td>• d: Day</td>
</tr>
<tr>
<td></td>
<td>• w: Weekday</td>
</tr>
<tr>
<td></td>
<td>• ww: Week</td>
</tr>
<tr>
<td></td>
<td>• h: Hour</td>
</tr>
<tr>
<td></td>
<td>• n: Minute</td>
</tr>
<tr>
<td></td>
<td>• s: Second</td>
</tr>
<tr>
<td></td>
<td>• l: Millisecond</td>
</tr>
<tr>
<td>date</td>
<td>Date/time object, in the range 100 AD–9999 AD.</td>
</tr>
</tbody>
</table>

Usage
When passing a date/time object as a string, you must enclose it in quotation marks. Otherwise, it is interpreted as a numeric representation of a date/time object.

Example
<!--- This example shows information available from DatePart --->
<cfset todayDate = Now()>
<h3>DatePart Example</h3>
<p>Today's date is <cfoutput>#todayDate#</cfoutput>,</p>
<p>Using datepart, we extract an integer representing the dateparts from that value</p>
<cfoutput>
<ul>
<li>year: #DatePart("yyyy", todayDate)#</li>
<li>quarter: #DatePart("q", todayDate)#</li>
<li>month: #DatePart("m", todayDate)#</li>
<li>day of year: #DatePart("y", todayDate)#</li>
<li>day: #DatePart("d", todayDate)#</li>
<li>weekday: #DatePart("w", todayDate)#</li>
<li>week: #DatePart("ww", todayDate)#</li>
<li>hour: #DatePart("h", todayDate)#</li>
<li>minute: #DatePart("n", todayDate)#</li>
<li>second: #DatePart("s", todayDate)#</li>
</ul>
Day

Description
Determines the day of the month, in a date.

Returns
The ordinal for the day of the month, ranging from 1 to 31.

Category
Date and time functions

Function syntax
Day("date")

See also
DayOfWeek, DayOfWeekAsString, DayOfYear, DaysInMonth, DaysInYear, FirstDayOfMonth

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>date</td>
<td>Date/time object, in the range 100 AD–9999 AD.</td>
</tr>
</tbody>
</table>

Usage
When passing a date/time object as a string, you must enclose it in quotation marks. Otherwise, it is interpreted as a numeric representation of a date/time object.

Note: You can pass the CreativeDate function or Now function as the date parameter of this function; for example: #Day(CreateDate(2001, 3, 3))#

Example

```coldfusion
<h3>Day Example</h3>
<cfif IsDefined("FORM.year")>
  <p>More information about your date:</p>
  <cfset yourDate = CreateDate(FORM.year, FORM.month, FORM.day)>
  <cfoutput>
    <p>Date: #DateFormat(yourDate)#. <br>
    It is #DayOfWeekAsString(DayOfWeek(yourDate))#, day #DayOfWeek(yourDate)# in the week. <br>
    This is day #Day(yourDate)# in the month of #MonthAsString(Month(yourDate))#, which has #DaysInMonth(yourDate)# days. <br>
    We are in week #Week(yourDate)# of #Year(YourDate)# <br>
    (day #DayOfYear(yourDate)# of #DaysInYear(yourDate)#). <br>
    <cfif IsLeapYear(Year(yourDate))>This is a leap year</cfif>
    <cfelse>This is not a leap year</cfif>
  </cfoutput>
</cfif>
```

560 Chapter 3: ColdFusion Functions
DayOfWeek

Description
Determine the day of the week, in a date.

Returns
The ordinal for the day of the week, as an integer in the range 1 (Sunday) to 7 (Saturday).

Category
Date and time functions

Function syntax
DayOfWeek("date")

See also
Day, DayOfWeekAsString, DayOfYear, DaysInMonth, DaysInYear, FirstDayOfMonth

Parameters
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>date</td>
<td>Date/time object, in the range 100 AD–9999 AD.</td>
</tr>
</tbody>
</table>

Usage
When passing a date/time object as a string, you must enclose it in quotation marks. Otherwise, it is interpreted as a numeric representation of a date/time object.

Note: You can pass the CreateDate function or Now function as the date parameter of this function; for example, #DayOfWeek(CreateDate(2001, 3, 3))#

Example
<h3>DayOfWeek Example</h3>
<cfif IsDefined("FORM.year")>
  More information about your date:
  <cfset yourDate = CreateDate(FORM.year, FORM.month, FORM.day)>
  <cfoutput>
  Your date, #DateFormat(yourDate)#.
  It is #DayOfWeekAsString(DayOfWeek(yourDate))#, day #DayOfWeek(yourDate)# in the week.
  This is day #Day(YourDate)# in the month of #MonthAsString(Month(yourDate))#, which has #DaysInMonth(yourDate)# days.
  We are in week #Week(yourDate)# of #Year(YourDate)# (day #DayofYear(yourDate)# of #DaysInYear(yourDate)#).
  <cfif IsLeapYear(Year(yourDate))>This is a leap year
  <cfelse>This is not a leap year</cfif>
  </cfoutput>
</cfif>

</cfoutput>
DayOfWeekAsString

Description
Determines the day of the week, in a date, as a string function.

Returns
The day of the week, as a string in the current locale, that corresponds to day_of_week.

Category
Date and time functions, String functions

Function syntax
DayOfWeekAsString(day_of_week)

See also
Day, DayOfWeek, DayOfYear, DaysInMonth, DaysInYear, FirstDayOfMonth

History
ColdFusion MX 7: Changed behavior. The returned string is now in the language of the current locale.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>day_of_week</td>
<td>Integer, in the range 1 (Sunday) - 7 (Saturday).</td>
</tr>
</tbody>
</table>

Example
The following example shows the use of the DayOfWeekAsString function. It is the action page for a form that submits year, month, and day fields.

```cfoutput```
<h3>DayOfWeekAsString Example</h3>
<cfif IsDefined("FORM.year")>
More information about your date:
<cfset yourDate = CreateDate(FORM.year, FORM.month, FORM.day)>
<br>Your date, #DateFormat(yourDate)#. 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<br><cfelse>This is not a leap year</cfif></cfoutput>
```
DayOfYear

Description
Determines the day of the year, in a date.

Returns
The ordinal value of day of the year, as an integer.

Category
Date and time functions

Function syntax
DayOfYear("date")

See also
Day, DayOfWeek, DayOfWeekAsString, DaysInMonth, DaysInYear, FirstDayOfMonth

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>date</td>
<td>Date/time object, in the range 100 AD–9999 AD.</td>
</tr>
</tbody>
</table>

Usage
This function accounts for leap years.

When passing a date/time object as a string, you must enclose it in quotation marks. Otherwise, it is interpreted as a numeric representation of a date/time object.

**Note:** You can pass the `CreateDate` function or `Now` function as the `date` parameter of this function; for example, `#DayOfYear(CreateDate(2001, 3, 3))#`

Example

```html
<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)#.</p>
<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)#
<br,#DayOfYear(yourDate)# of #DaysInYear(yourDate)#).
<cfif IsLeapYear(Year(yourDate))>This is a leap year</cfif>
<cfelse>This is not a leap year</cfif>
</cfoutput>
</cfif>
```
DaysInMonth

Description
Determines the number of days in a month.

Returns
The number of days in the month in Date.

Category
Date and time functions

Function syntax
DaysInMonth("date")

See also
Day, DayOfWeek, DayOfWeekAsString, DayOfYear, DaysInYear, FirstDayOfMonth

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>date</td>
<td>Date/time object, in the range 100 AD–9999 AD.</td>
</tr>
</tbody>
</table>

Usage
When passing a date/time object as a string, you must enclose it in quotation marks. Otherwise, it is interpreted as a numeric representation of a date/time object.

Note: You can pass the Now function or the CreateDate function as the date parameter of this function; for example: #DaysInMonth(CreateDate(2001, 3, 3))#

Example
<h3>DaysInMonth Example</h3>
<cfif IsDefined("FORM.year")>
  More information about your date:
  <cfset yourDate = CreateDate(FORM.year, FORM.month, FORM.day)>
  <cfoutput>
  Your date. #DateFormat(yourDate)#.
  It is #DayOfWeekAsString(DayOfWeek(yourDate))#, day #DayOfWeek(yourDate)# in the week.
  This is day #Day(yourDate)# in the month of #MonthAsString(Month(yourDate))#, which has #DaysInMonth(yourDate)# days.
  We are in week #Week(yourDate)# of #Year(yourDate)# (day #DayOfWeek(yourDate)# of #DaysInYear(yourDate)#).
  <cfif IsLeapYear(Year(yourDate))>This is a leap year  
  <cfelse>This is not a leap year</cfif>
  </cfoutput>
</cfif>
DaysInYear

Description
Determines the number of days in a year.

Returns
The number of days in a year.

Category
Date and time functions

Function syntax
DaysInYear("date")

See also
Day, DayOfWeek, DayOfWeekAsString, DayOfYear, DaysInMonth, DaysInYear, FirstDayOfMonth, IsLeapYear

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>date</td>
<td>• Date/time object, in the range 100 AD–9999 AD.</td>
</tr>
</tbody>
</table>

Usage
DaysInYear accounts for leap years.

When passing a date/time object as a string, you must enclose it in quotation marks. Otherwise, it is interpreted as a numeric representation of a date/time object.

**Note:** You can pass the CreateDate function or the Now function as the date parameter of this function; for example: #DaysInYear(CreateDate(2001, 3, 3))#

Example

```html
<h3>DaysInYear Example</h3>
<cfif IsDefined("FORM.year")>
  More information about your date:
  <cfset yourDate = CreateDate(FORM.year, FORM.month, FORM.day)>
  <cfoutput>
    Your date, #DateFormat(yourDate)#.
    It is #DayOfWeekAsString(DayOfWeek(yourDate))#, day #DayOfWeek(yourDate)# in the week.
    This is day #Day(YourDate)# in the month of #MonthAsString(Month(yourDate))#, which has #DaysInMonth(yourDate)# days.
    We are in week #Week(yourDate)# of #Year(yourDate)# (day #DayOfYear(yourDate)# of #DaysInYear(yourDate)#).
  </cfoutput>
</cfif>
```
DE

Description

Escapes any double-quotation marks in the parameter and wraps the result in double-quotation marks.

Returns

Parameter, surrounded by double-quotation marks, with any inner double-quotation marks escaped.

Category

Dynamic evaluation functions

Function syntax

DE(string)

See also

Evaluate, IIf, Chapter 4, “Using Expressions and Number Signs” in ColdFusion MX Developer's Guide

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>string</td>
<td>String to evaluate, after delay</td>
</tr>
</tbody>
</table>

Usage

The DE function postpones evaluation of a string that is passed as a parameter to the IIf or Evaluate functions.

This function is especially useful with the IIf function, which automatically evaluates its second and third parameters as expressions. You can use the DE function to prevent the function from evaluating a string parameter that is to be output as a variable, and should not be treated as an expression. The following example show this use; it uses IIF to alternate table-row background colors, white and gray, and uses the DE function to prevent ColdFusion from evaluating the color strings.

```plaintext
<cfoutput>
<table border="1" cellpadding="3">
<cfloop index="i" from="1" to="10">
  <tr bgcolor="#IIF( i mod 2 eq 0, DE("white"), DE("gray") )#">
    <td>
      hello #i#
    </td>
  </tr>
</cfloop>
</table>
</cfoutput>
```

The DE function does not delay evaluation of variable names that are surrounded by number signs (#). ColdFusion function evaluates the variable regardless of whether the DE function is present.
The following example shows how you can use the DE function and number signs together, and shows how the function works with an IIF function:

```coldfusion
<cfoutput>
<cfset var1=Blue>
<cfset var2=Green>
<cfset myresult=IIf( 1 eq 2, DE(#Var1#), DE(#Var2#))>
The expression is #myresult#
</cfoutput>

ColdFusion processes this code as follows:
1. ColdFusion sets the variables var1 and var2 to be the strings Blue and Green.
2. In the fourth line, ColdFusion evaluates the variables surrounded by number signs first, replacing them with the strings Blue and Green, the values of the variables.
3. The IIF function evaluates the test expression, determines that it is False, and then evaluates the third parameter.
4. The third parameter is a DE function, which takes the string Green and surrounds it in quotation marks.
5. The IIF function returns the string "Green", including the quotation marks.
6. The cfset tag gets the expression result="Green", and sets the value of the myresult variable to the string Green.
7. ColdFusion evaluates #myresult# in the output text, replaces the variable with its value, the string Green, and displays the result.
```

**Example**

<!--- This example shows the use of DE and Evaluate --->
```coldfusion
<h3>DE Example</h3>
<cfif IsDefined("FORM.myExpression")>
<cftry>
<!--- Show the expression and the results of evaluating it --->
<cfoutput>
<h3>Evaluate the Expression #FORM.MyExpression#</h3>
The code:<br>
#Evaluate(FORM.myExpression)#<br><br>
The result:<br>
#Evaluate(FORM.myExpression)#
</cfoutput>

Use DE to prevent the Evaluate function from evaluating</h3>
The code:<br>
#Evaluate(DE(FORM.MyExpression))#<br><br>
The result:<br>
#Evaluate(DE(FORM.MyExpression))#
</cfoutput>
</cftry>
<!--- Error handling code for bad expressions and any other error.--->
</cfcatch type = "Any">
```
<!--- the message to display --->
<h3>Sorry, there's been an <B>Error</B>.</h3>
Try a simple expression, such as "2+2".</h3>
<cfoutput>
<!--- Display the diagnostic message from ColdFusion. --->
<p>#cfcatch.message#</p>
</cfoutput>
</cfcatch>
</cftry>
</cfif>

<h3>Enter any valid ColdFusion expression</h3>
<cfform>
<cfinput name="myExpression" type="Text" size="40">
<cfinput type="submit" name="submitit">
</cfform>
**DecimalFormat**

**Description**

Converts a number to a decimal-formatted string.

**Returns**

A *number* as a string formatted with two decimal places and a thousands separator.

**Category**

Display and formatting functions

**Function syntax**

`DecimalFormat(number)`

**See also**

`DollarFormat, NumberFormat`

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>number</td>
<td>Number to format</td>
</tr>
</tbody>
</table>

**Example**

```html
<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 * sqrt(2))#
  </cfoutput>
  <br>
</cfloop>
```
DecrementValue

Description
Decr...
Decrypt

Description
Decrypts a string that is encrypted using a standard encryption technique, including strings encrypted by the Encrypt function.

Returns
An unencrypted string.

Category
Security functions, String functions

Function syntax
`Decrypt(encrypted_string, key[, algorithm[, encoding]])`

See also
Duplicate, Encrypt

ColdFusion MX 7: Added the algorithm and encoding parameters.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>encrypted_string</td>
<td>String to decrypt.</td>
</tr>
<tr>
<td>key</td>
<td>String. For the CFMX_COMPAT algorithm, the seed that was used to encrypt the string; for all other algorithms, the key that was used to encrypt the string.</td>
</tr>
<tr>
<td>algorithm</td>
<td>(Optional) The algorithm to use to decrypt the string. Must be the same as the algorithm used to encrypt the string. ColdFusion MX installs a cryptography library with the following algorithms:</td>
</tr>
<tr>
<td></td>
<td>• CFMX_COMPAT: the algorithm used in ColdFusion MX and prior releases. This algorithm is the least secure option (default).</td>
</tr>
<tr>
<td></td>
<td>• AES: the Advanced Encryption Standard specified by the National Institute of Standards and Technology (NIST) FIPS-197.</td>
</tr>
<tr>
<td></td>
<td>• BLOWFISH: the Blowfish algorithm defined by Bruce Schneier.</td>
</tr>
<tr>
<td></td>
<td>• DES: the Data Encryption Standard algorithm defined by NIST FIPS-46-3.</td>
</tr>
<tr>
<td></td>
<td>• DESEDE: the “Triple DES” algorithm defined by NIST FIPS-46-3.</td>
</tr>
<tr>
<td></td>
<td>If you install a security provider with additional cryptography algorithms, you can also specify any of its string encryption and decryption algorithms.</td>
</tr>
<tr>
<td>encoding</td>
<td>(Optional; if you specify this parameter, you must also specify the algorithm parameter.) The binary encoding used to represent the data as a string. Must be the same as the algorithm used to encrypt the string.</td>
</tr>
<tr>
<td></td>
<td>• Base64: the Base64 algorithm, as specified by IETF RFC 2045,</td>
</tr>
<tr>
<td></td>
<td>• Hex: the characters A-F and 0-9 represent the hexadecimal byte values.</td>
</tr>
<tr>
<td></td>
<td>• UU: the UNIX standard UUEncode algorithm (default).</td>
</tr>
</tbody>
</table>
Usage

This function uses a symmetric key-based algorithm, in which the same key is used to encrypt and decrypt a string. The parameter values must match the values used to encode string. The security of the encrypted string depends on maintaining the secrecy of the key.

ColdFusion MX 7 uses the Java Cryptography Extension (JCE) and installs a Sun Java 1.4.2 runtime that includes the Sun JCE default security provider. This provider includes the algorithms listed in the Parameters section. The JCE framework includes facilities for using other provider implementations; however, Macromedia cannot provide technical support for third-party security providers.

Example

<!--- Do the following if the form has been submitted. --->
<cfif IsDefined("Form.myString")>
<cfscript>
/* GenerateSecretKey does not generate key for the CFMX_COMPAT algorithm, so use the key from the form. */
if (Form.myAlgorithm EQ "CFMX_COMPAT")
   theKey=Form.MyKey;
// For all other encryption techniques, generate a secret key.
else
   theKey=generateSecretKey(Form.myAlgorithm);
//Encrypt the string
encrypted=encrypt(Form.myString, theKey, Form.myAlgorithm, Form.myEncoding);
//Decrypt it
decrypted=decrypt(encrypted, theKey, Form.myAlgorithm, Form.myEncoding);
</cfscript>
<!--- Display the values used for encryption and decryption, and the results. --->
<cfoutput>
<b>The algorithm:</b> #Form.myAlgorithm#<br>
<b>The key:</b> #theKey#<br>
<b>The string:</b> #Form.myString# <br>
<b>Encrypted:</b> #encrypted#<br>
<b>Decrypted:</b> #decrypted#<br>
</cfoutput>
</cfif>

<!--- The input form. --->
<form action="#CGI.SCRIPT_NAME#" method="post">
<b>Select the encoding</b><br>
<select size="1" name="myEncoding">
<option selected>UU</option>
<option>Base64</option>
<option>Hex</option>
</select>
</form>
<select size="1" name="myAlgorithm" >
  <option selected>CFMX_COMPAT</option>
  <option>AES</option>
  <option>DES</option>
  <option>DESEDE</option>
</select>

<b>Select the algorithm</b>

<b>Input your key</b> (used for CFMX_COMPAT encryption only)
<input type = "Text" name = "myKey" value = "MyKey"/>

<b>Enter string to encrypt</b>
<textarea name = "myString" cols = "40" rows = "5" WRAP = "VIRTUAL"">This string will be encrypted (you can replace it with more typing).</textarea>

<input type = "Submit" value = "Encrypt my String"/>
DeleteClientVariable

**Description**
Deletes a client variable. (To test for the existence of a variable, use IsDefined.)

**Returns**
True, if the variable is successfully deleted; false, otherwise.

**Category**
Other functions

**Function syntax**
DeleteClientVariable("name")

**See also**
GetClientVariablesList

**History**
ColdFusion MX: Changed behavior: if the variable is not present, this function now returns False. (In earlier releases, it threw an error.)

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Name of a client variable to delete, surrounded by double-quotation marks</td>
</tr>
</tbody>
</table>

**Example**

<!--- This view-only example shows DeleteClientVariable --->

<h3>DeleteClientVariable Example</h3>

<p>This view-only example deletes a client variable called "User_ID", if it exists in the list of client variables returned by GetClientVariablesList. This example requires the existence of an Application.cfm file and client management to be in effect.
</p>

<cfset client.somevar = ">
<cfset client.user_id = ">
<p>Client variable list:<cfoutput>#GetClientVariablesList()#</cfoutput>
<cfif ListFindNoCase(GetClientVariablesList(), "User_ID") is not 0>
  <cfset temp = DeleteClientVariable("User_ID")>
  <p>Was variable "User_ID" Deleted? <cfoutput>#temp#</cfoutput>
</cfif>
<p>Amended Client variable list:<cfoutput>#GetClientVariablesList()#</cfoutput>
--->
DirectoryExists

Description
Determined whether a directory exists.

Returns
Yes, if the specified directory exists; No, otherwise.

Category
System functions

Function syntax
DirectoryExists(absolute_path)

See also
FileExists

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>absolute_path</td>
<td>An absolute path</td>
</tr>
</tbody>
</table>

Example

<h3>DirectoryExists Example</h3>
Enter a directory to check for existence.
<form action = "directoryexists.cfm" method="post">
  <input type = "text" name = "yourDirectory">
  <br>
  <input type = "submit" name = "submit">
</form>

<cfif IsDefined("FORM.yourDirectory")>
  <cfif FORM.yourDirectory is not "">
    <cfset yourDirectory = FORM.yourDirectory>
    <cfif DirectoryExists(yourDirectory)>
      <cfoutput>
        <p>Your directory exists. Directory name: #yourDirectory#</p>
      </cfoutput>
    </cfif>
    <cfelse>
      <p>Your directory does not exist.</p>
    </cfelse>
  </cfif>
</cfif>
DollarFormat

Description

Formats a string in U.S. format. (For other currencies, use LSCurrencyFormat or LSEuroCurrencyFormat.)

Returns

A number as a string, formatted with two decimal places, thousands separator, and dollar sign. If number is negative, the return value is enclosed in parentheses. If number is an empty string, returns zero.

Category

Display and formatting functions

Function syntax

DollarFormat(number)

See also

DecimalFormat, NumberFormat

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>number</td>
<td>Number to format</td>
</tr>
</tbody>
</table>

Example

<!--- This example shows the use of DollarFormat --->
...<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(half * (18/100))##DollarFormat(threefourth * (18/100))#
    </pre>
  </cfoutput>
</cfloop>
...
**Duplicate**

**Description**
Returns a clone, also known as a deep copy, of a variable. There is no reference to the original variable.

**Returns**
A clone of a variable.

**Category**
Structure functions, System functions

**Function syntax**
Duplicate(variable_name)

**See also**

**History**
ColdFusion MX: Changed behavior: this function can be used on XML objects.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>variable_name</td>
<td>Name of a variable to duplicate</td>
</tr>
</tbody>
</table>

**Usage**
Use this function to duplicate complex structures, such as nested structures and queries.

*Note:* The `Duplicate` function throws an exception if you attempt to duplicate a CFC.

*Note:* With this function, you cannot duplicate a COM, CORBA, or JAVA object returned from the `cfoject` tag or the `CreateObject` function. If an array element or structure field is a COM, CORBA, or JAVA object, you cannot duplicate the array or structure.

**Example**

```cfc
default
<h3>Duplicate Example</h3>
<cfset s1 = StructNew()>
<cfset s1.nested = StructNew()>
<cfset s1.nested.item = "original">
<cfset copy = StructCopy(s1)>
<cfset clone = Duplicate(s1)>
<cfset s1.nested.item = "modified">
<cfoutput>
<p>The copy contains the modified value: #copy.nested.item#</p>
<p>The duplicate contains the original value: #clone.nested.item#</p>
</cfoutput>
```


Encrypt

Description
Encrypts a string using a specific algorithm and encoding method.

Returns
String; can be much longer than the original string.

Category
Security functions, String functions

Function syntax
Encrypt(string, key[, algorithm[, encoding]])

See also
Decrypt

History
ColdFusion MX 7: Added the algorithm and encoding parameters.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>string</td>
<td>String to encrypt.</td>
</tr>
<tr>
<td>key</td>
<td>String. Key or seed used to encrypt the string.</td>
</tr>
<tr>
<td></td>
<td>• For the CFMX_COMPAT algorithm, any combination of any number of characters; used as a seed used to generate a 32-bit encryption key.</td>
</tr>
<tr>
<td></td>
<td>• For all other algorithms, a key in the format used by the algorithm. For these algorithms, use the GenerateSecretKey function to generate the key.</td>
</tr>
<tr>
<td>algorithm</td>
<td>(Optional) The algorithm to use to decrypt the string. ColdFusion MX installs a cryptography library with the following algorithms:</td>
</tr>
<tr>
<td></td>
<td>• CFMX_COMPAT: the algorithm used in ColdFusion MX and prior releases. This algorithm is the least secure option (default).</td>
</tr>
<tr>
<td></td>
<td>• AES: the Advanced Encryption Standard specified by the National Institute of Standards and Technology (NIST) FIPS-197.</td>
</tr>
<tr>
<td></td>
<td>• BLOWFISH: the Blowfish algorithm defined by Bruce Schneier.</td>
</tr>
<tr>
<td></td>
<td>• DES: the Data Encryption Standard algorithm defined by NIST FIPS-46-3.</td>
</tr>
<tr>
<td></td>
<td>• DESEDE: the &quot;Triple DES&quot; algorithm defined by NIST FIPS-46-3.</td>
</tr>
<tr>
<td></td>
<td>If you install a security provider with additional cryptography algorithms, you can also specify any of its string encryption and decryption algorithms.</td>
</tr>
<tr>
<td>encoding</td>
<td>(Optional; if you specify this parameter, you must also specify the algorithm parameter) The binary encoding in which to represent the data as a string.</td>
</tr>
<tr>
<td></td>
<td>• Base64: the Base64 algorithm, as specified by IETF RFC 2045.</td>
</tr>
<tr>
<td></td>
<td>• Hex: the characters !-F0-9 represent the hexadecimal byte values.</td>
</tr>
<tr>
<td></td>
<td>• UU: the UUEncode algorithm (default).</td>
</tr>
</tbody>
</table>
Usage

This function uses a symmetric key-based algorithm, in which the same key is used to encrypt and decrypt a string. The security of the encrypted string depends on maintaining the secrecy of the key.

For all algorithms except the default algorithm, ColdFusion MX 7 uses the Java Cryptography Extension (JCE) and installs a Sun Java 1.4.2 runtime that includes the Sun JCE default security provider. This provider includes the algorithms listed in the Parameters section. The JCE framework includes facilities for using other provider implementations; however, Macromedia cannot provide technical support for third-party security providers.

The default algorithm, which is the same as was used in ColdFusion 5 and ColdFusion MX, uses an XOR-based algorithm that uses a pseudo-random 32-bit key, based on a seed passed by the user as a function parameter. This algorithm is less secure than the other available algorithms.

Example

The following example encrypts and decrypts a text string. It lets you specify the encryption algorithm and encoding technique. It also has a field for a key seed to use with the CFMX_COMPAT algorithm. For all other algorithms, it generates a secret key.

```cfscript>
/* GenerateSecretKey does not generate key for the CFMX_COMPAT algorithm, so use the key from the form. */
if (Form.myAlgorithm EQ "CFMX_COMPAT")
   theKey=Form.MyKey;
else
   theKey=generateSecretKey(Form.myAlgorithm);
//Encrypt the string
encrypted=encrypt(Form.myString, theKey, Form.myAlgorithm, Form.myEncoding);
//Decrypt it
decrypted=decrypt(encrypted, theKey, Form.myAlgorithm, Form.myEncoding);
</cfscript>

<!--- Display the values used for encryption and decryption, and the results. --->
<b>The algorithm:</b> #Form.myAlgorithm#
<b>The key:</b> #theKey#
<b>The string:</b> #Form.myString#
<b>Encrypted:</b> #encrypted#
<b>Decrypted:</b> #decrypted#
</cfoutput>
</cfif>

<!--- Do the following if the form has been submitted. --->
<cfif IsDefined("Form.myString")>
<!--- Display the values used for encryption and decryption, and the results. --->
<b>The algorithm:</b> #Form.myAlgorithm#
<b>The key:</b> #theKey#
<b>The string:</b> #Form.myString#
<b>Encrypted:</b> #encrypted#
<b>Decrypted:</b> #decrypted#
</cfoutput>
</cfif>
<!-- The input form. -->
<form action="#CGI.SCRIPT_NAME#" method="post">
  <b>Select the encoding</b>
  <select size="1" name="myEncoding">
    <option selected>UU</option>
    <option>Base64</option>
    <option>Hex</option>
  </select>

  <br>
  <b>Select the algorithm</b>
  <select size="1" name="myAlgorithm">
    <option selected>CFMX_COMPAT</option>
    <option>AES</option>
    <option>DES</option>
    <option>DESEDE</option>
  </select>

  <br>
  <b>Input your key</b> (used for CFMX_COMPAT encryption only)
  <input type="Text" name="myKey" value="MyKey">

  <br>
  <b>Enter string to encrypt</b>
  <textarea name="myString" cols="40" rows="5" WRAP="VIRTUAL">This string will be encrypted (you can replace it with more typing).</textarea>

  <input type="Submit" value="Encrypt my String">
</form>
Evaluate

Description
Evaluates one or more string expressions, dynamically, from left to right. (The results of an evaluation on the left can have meaning in an expression to the right.) Returns the result of evaluating the rightmost expression.

Returns
An object; the result of the evaluation(s).

Category
Dynamic evaluation functions

Function syntax
Evaluate(string_expression1 [, string_expression2 [, ... ] ]

See also
DE, IIf

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>string_expression1</td>
<td>Expressions to evaluate</td>
</tr>
<tr>
<td>string_expression2</td>
<td></td>
</tr>
</tbody>
</table>

Usage
String expressions can be complex. If a string expression contains a single- or double-quotation mark, the mark must be escaped.

This function is useful for forming one variable from multiple variables. For example, to reference a column of the query qNames with a variable, var, using an index value to traverse rows, you could use the following code:

```coldfusion
<cfset var=Evaluate("qNames.#colname#[#index#]")>
```

For more information, see Chapter 4, “Using Expressions and Number Signs” in ColdFusion MX Developer's Guide.

Example
```coldfusion
<!--- This example shows the use of DE and Evaluate --->
<h3>Evaluate Example</h3>
<cfif IsDefined("FORM.myExpression")>
  <cftry>
    <!--- Evaluate the expression --->
    <cfset myExpression = Evaluate(FORM.myExpression)>
    <!--- Use DE to output the value of the variable, unevaluated --->
    <cfoutput>
The value of the expression #Evaluate(DE(FORM.MyExpression))# is #myExpression#.
</cfoutput>
  </cftry>
</cfif>
...
Exp

Description
Calculates the exponent whose base is e that represents \texttt{number}. The constant e equals 2.71828182845904, the base of the natural logarithm. This function is the inverse of \texttt{Log}, the natural logarithm of \texttt{number}.

Returns
The constant e, raised to the power of \texttt{number}.

Category
Mathematical functions

Function syntax
\texttt{Exp(number)}

See also
\texttt{Log, Log10}

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>number</td>
<td>Exponent to apply to the base e</td>
</tr>
</tbody>
</table>

Usage
To calculate powers of other bases, use the exponentiation operator (^).

Example

```
<h3>Exp Example</h3>
<cfif IsDefined("FORM.Submit")>
  <cfoutput>
  <p>Your number, #FORM.number#
  <br>#FORM.number# raised to the E power: \texttt{exp(FORM.number)}
  <cfif FORM.number LTE 0>
    <br>You must enter a positive real number to see its natural logarithm
    <cfelse><br>
    The natural logarithm of #FORM.number#: \texttt{log(FORM.number)}
  </cfif>
  <cfif FORM.number LTE 0><br>
  You must enter a positive real number to see its logarithm to base 10
  <cfelse><br>
  The logarithm of #FORM.number# to base 10: \texttt{log10(FORM.number)}
  </cfif>
  </cfoutput>
</cfif>
<cfform action = "exp.cfm">
Enter a number to see its value raised to the E power, its natural logarithm, and the logarithm of number to base 10.
<cfinput type = "Text" name = "number" message = "You must enter a number" validate = "float" required = "No">
<cfinput type = "Submit" name = "Submit">
</cfform>
```
ExpandPath

Description

Creates an absolute, platform-appropriate path that is equivalent to the value of `relative_path`, appended to the base path. This function (despite its name) can accept an absolute or relative path in the `relative_path` parameter.

The base path is the currently executing page’s directory path. It is stored in `pageContext.getServletContext()`.

Returns

A string. If the relative path contains a trailing forward slash or backward slash, the return value contains the same trailing character.

Category

System functions

Function syntax

```
ExpandPath(relative_path)
```

See also

`FileExists`, `GetCurrentTemplatePath`, `GetFileFromPath`

History

ColdFusion MX: Changed behavior for the `relative_path` parameter: this function can now accept an absolute or relative path in the `relative_path` parameter. To resolve a path, this function uses virtual mappings that are defined in the ColdFusion Administrator. This function does not reliably use virtual mappings that are defined in IIS, Apache, or other web servers.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>relative_path</td>
<td>Relative or absolute directory reference or file name, within the current directory, (\ and ..) to convert to an absolute path. Can include forward or backward slashes.</td>
</tr>
</tbody>
</table>

Usage

If the parameter or the returned path is invalid, the function throws an error.

These examples show the valid constructions of `relative_path`:

- `ExpandPath( "*.*")`
- `ExpandPath( "/")`
- `ExpandPath( "\")`
- `ExpandPath( "/mycfpage.cfm")`
- `ExpandPath( "mycfpage.cfm")`
- `ExpandPath( "myDir/mycfpage.cfm")`
- `ExpandPath( "/myDir/mycfpage.cfm")`
- `ExpandPath( "./../mycfpage.cfm")`
Example

<h3>ExpandPath Example - View Only</h3>

<!---
<cfset thisPath = ExpandPath("*.*)">
<cfset thisDirectory = GetDirectoryFromPath(thisPath)>
<cfoutput>
The current directory is: #GetDirectoryFromPath(thisPath)#

<cfif IsDefined(form.yourFile)>
<cfif form.yourFile is not ">
<cfset yourFile = form.yourFile>
<cfif FileExists(ExpandPath(yourFile))>
<p>Your file exists in this directory. You entered the correct file name. #GetFileFromPath(#thisPath#/#yourfile#)#
<CFELSE>
<p>Your file was not found in this directory:
<br>Here is a list of the other files in this directory:
<!--- use CFDIRECTORY to give the contents of the snippets directory, order by name and size --->
<CFDIRECTORY DIRECTORY="#thisDirectory#" NAME="myDirectory" SORT="name ASC, size DESC">
<!--- Output the contents of the CFDIRECTORY as a CFTABLE --->
<CFTABLE QUERY="myDirectory">
<CFCOL HEADER="NAME:" TEXT="#Name#">
<CFCOL HEADER="SIZE:" TEXT="#Size#">
</CFTABLE>
</CFELSE>
</cfif>
</cfif>
<cfelse>
<h3>Please enter a file name</h3>
</CFIF>
</cfoutput>

<form action="expandpath.cfm" METHOD="post">
<h3>Enter the name of a file in this directory <I><FONT SIZE="-1">(try expandpath.cfm)</FONT></I></h3>
<input TYPE="Text" NAME="yourFile">
<input TYPE="Submit" NAME="">
</form>

---
**FileExists**

**Description**
Determine whether a file exists.

**Returns**
Yes, if the file specified in the parameter exists; No, otherwise.

**Category**
System functions, Decision functions

**Function syntax**
FileExists(absolute_path)

**See also**
DirectoryExists, ExpandPath, GetTemplatePath

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>absolute_path</td>
<td>An absolute path</td>
</tr>
</tbody>
</table>

**Usage**
To access a file on a remote system, the account (for Windows) or user (for UNIX and Linux) that is running ColdFusion must have permission to access the file, directory, and remote system. For example, if you run ColdFusion MX in the Server Configuration as a Windows service, by default it runs under the local system account, which does not have sufficient privileges to access remote systems. You can change this, however, on the Log On page of the Services Properties dialog box.

**Example**
<h3>FileExists Example</h3>

```cftmpl
<cfset thisPath = ExpandPath("*.*)>
<cfset thisDirectory = GetDirectoryFromPath(thisPath)>
<cfoutput>
The current directory is: #GetDirectoryFromPath(thisPath)#
<cfif IsDefined("FORM.yourFile")>
<cfif FORM.yourFile is not ">"</cfif>
<cfset yourFile = FORM.yourFile>
<cfif FileExists(ExpandPath(yourfile))>
<p>Your file exists in this directory. You entered the correct file name. #GetFileFromPath("#thisPath#/yourfile")#</p>
<cfelse>
```

FileExists  585
Find

Description
Finds the first occurrence of a *substring* in a *string*, from a specified start position. The search is case-sensitive.

Returns
A number; the position of *substring* in *string*; or 0, if *substring* is not in *string*.

Category
String functions

Function syntax
```
Find(substring, string [, start ])
```

See also
FindNoCase, Compare, FindOneOf, REFind, Replace

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>substring</td>
<td>A string or a variable that contains one. String for which to search.</td>
</tr>
<tr>
<td>string</td>
<td>A string or a variable that contains one. String in which to search.</td>
</tr>
<tr>
<td>start</td>
<td>Start position of search.</td>
</tr>
</tbody>
</table>

Example
```
<cfoutput>
<cfset stringToSearch = "The quick brown fox jumped over the lazy dog.">
#find("the",stringToSearch)#<br>
#find("the",stringToSearch,35)#<br>
#find("no such substring",stringToSearch)#<br>
<cfif stringToSearch>
#findnocase("the",stringToSearch)#<br>
#findnocase("the",stringToSearch,5)#<br>
#findnocase("no such substring",stringToSearch)#<br>
<cfif stringToSearch>
#findoneof("aeiou",stringToSearch)#<br>
#findoneof("aeiou",stringToSearch,4)#<br>
#findoneof("@%^()",stringToSearch)#<br>
</cfoutput>
```
FindNoCase

Description

Finds the first occurrence of a substring in a string, from a specified start position. If substring is not in string, returns zero. The search is case-insensitive.

Returns

The position of substring in string; or 0, if substring is not in string.

Category

String functions

Function syntax

FindNoCase(substring, string [, start ])

See also

Find, CompareNoCase, FindOneOf, REFind, Replace

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>substring</td>
<td>A string or a variable that contains one. String for which to search.</td>
</tr>
<tr>
<td>string</td>
<td>A string or a variable that contains one. String in which to search.</td>
</tr>
<tr>
<td>start</td>
<td>Start position of search.</td>
</tr>
</tbody>
</table>

Example

In the following example, the Find function returns 33 as the first position found because "the" is lowercase. The FindNoCase function returns 1 as the first position because the case is ignored.

```cftexthtml
<cfset stringToSearch = "The quick brown fox jumped over the lazy dog.">

stringToSearch = <cfoutput>#stringToSearch#</cfoutput><br>

Find Function:<br>
Find("the",stringToSearch) returns <cfoutput>#find("the",stringToSearch)#</cfoutput><br>

FindNoCase Function:<br>
FindNoCase("the",stringToSearch) returns
  <cfoutput>#FindNoCase("the",stringToSearch)#</cfoutput>
```
FindOneOf

**Description**

Finds the first occurrence of *any one of a set of characters* in a *string*, from a specified start position. The search is case-sensitive.

**Returns**

The position of the first member of *set* found in *string*; or 0, if no member of *set* is found in *string*.

**Category**

String functions

**Function syntax**

```
FindOneOf(set, string [, start ])
```

**See also**

Find, Compare, REFind

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>set</td>
<td>A string or a variable that contains one or more characters to search for.</td>
</tr>
<tr>
<td>string</td>
<td>A string or a variable that contains one. String in which to search.</td>
</tr>
<tr>
<td>start</td>
<td>Start position of search.</td>
</tr>
</tbody>
</table>

**Example**

```
<cfset stringToSearch = "The quick brown fox jumped over the lazy dog.">
#find("the",stringToSearch)#<br>
#find("the",stringToSearch,35)#<br>
#find("no such substring",stringToSearch)#<br>
<br>
#findnocase("the",stringToSearch)#<br>
#findnocase("the",stringToSearch,5)#<br>
#findnocase("no such substring",stringToSearch)#<br>
<br>
#findOneOf("aeiou",stringToSearch)#<br>
#findOneOf("aeiou",stringToSearch,4)#<br>
#findOneOf("@%^()",stringToSearch)#<br>
```

588 Chapter 3: ColdFusion Functions
**FirstDayOfMonth**

**Description**
Determines the ordinal (day number, in the year) of the first day of the month in which a given date falls.

**Returns**
A number corresponding to a day-number in a year.

**Category**
Date and time functions

**Function syntax**
`FirstDayOfMonth(date)`

**See also**
Day, DayOfWeek, DayOfWeekAsString, DayOfYear, DaysInMonth, DaysInYear

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>date</td>
<td>Date/time object, in the range 100 AD–9999 AD.</td>
</tr>
</tbody>
</table>

**Usage**
When passing a date/time value as a string, you must enclose it in quotation marks. Otherwise, it is interpreted as a number representation of a date/time object.

**Example**

```html
<h3>FirstDayOfMonth Example</h3>

<cfoutput>
The first day of #MonthAsString(Month(Now()))#, #Year(Now())# was day #FirstDayOfMonth(Now())# of the year.
</cfoutput>
```
Fix

Description

Converts a real number to an integer.

Returns

If number is greater than or equal to 0, the closest integer less than number.
If number is less than 0, the closest integer greater than number.

Category

Mathematical functions

Function syntax

Fix(number)

See also

Ceiling, Int, Round

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>number</td>
<td>A number</td>
</tr>
</tbody>
</table>

Example

<h3>Fix Example</h3>

<p>Fix returns the closest integer less than the number if the number is greater than or equal to 0. Fix returns the closest integer greater than the number if number is less than 0.</p>

<cfoutput>
<p>The fix of 3.4 is #Fix(3.4)#</p>
<p>The fix of 3 is #Fix(3)#</p>
<p>The fix of 3.8 is #Fix(3.8)#</p>
<p>The fix of -4.2 is #Fix(-4.2)#</p>
</cfoutput>
FormatBaseN

Description

Converts number to a string, in the base specified by radix.

Returns

String that represents number, in the base radix.

Category

Display and formatting functions, Mathematical functions, String functions

Function syntax

FormatBaseN(number, radix)

See also

InputBaseN

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>number</td>
<td>Number to convert</td>
</tr>
<tr>
<td>radix</td>
<td>Base of the result</td>
</tr>
</tbody>
</table>

Example

<h3>FormatBaseN Example</h3>  
<p>Converts a number to a string in the base specified by Radix.</p>  
<cfoutput>  
<br>FormatBaseN(10,2): #FormatBaseN(10,2)#  
<br>FormatBaseN(1024,16): #FormatBaseN(1024,16)#  
<br>FormatBaseN(125,10): #FormatBaseN(125,10)#  
<br>FormatBaseN(10.75,2): #FormatBaseN(10.75,2)#  
</cfoutput>  
<h3>InputBaseN Example</h3>  
<p>InputBaseN returns the number obtained by converting a string, using base specified by Radix (an integer from 2 to 36).</p>  
<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>
GenerateSecretKey

**Description**

Gets a secure key value for use in the *Encrypt* function.

**Returns**

A string containing the encryption key.

**Category**

Security functions, String functions

**Function syntax**

```
GenerateSecretKey(algorithm)
```

**See also**

Decrypt, Encrypt

**History**

ColdFusion MX 7: Added this function.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| algorithm | The encryption algorithm for which to generate the key. ColdFusion MX installs a cryptography library with the following algorithms:  
  • AES: the Advanced Encryption Standard specified by the National Institute of Standards and Technology (NIST) FIPS-197.  
  • BLOWFISH: the Blowfish algorithm defined by Bruce Schneier.  
  • DES: the Data Encryption Standard algorithm defined by NIST FIPS-46-3.  
  • DESEDE: the "Triple DES" algorithm defined by NIST FIPS-46-3. |

**Usage**

You cannot use the *GenerateSecretKey* function to generate a key for the ColdFusion default encryption algorithm (CFMX_COMPAT) of the *Encrypt* and *Decrypt* functions.

ColdFusion MX 7 uses the Java Cryptography Extension (JCE) and installs a Sun Java 1.4.2 runtime that includes the Sun JCE default security provider. This provider includes the algorithms listed in the Parameters section. The JCE framework includes facilities for using other provider implementations; however, Macromedia cannot provide technical support for third-party security providers.
Example

The following example encrypts and decrypts a text string. It lets you specify the encryption algorithm and encoding technique. It also has a field for a key seed to use with the CFMX_COMPAT algorithm. For all other algorithms, it uses the GenerateSecretKey function to generate a secret key.

<h3>Decrypt Example</h3>

<!--- Do the following if the form has been submitted. --->
<cfif IsDefined("Form.myString")>
  <cfscript>
    /* GenerateSecretKey does not generate keys for the CFMX_COMPAT algorithm, so we use a key from the form.
     */
    if (Form.myAlgorithm EQ "CFMX_COMPAT")
      theKey=Form.MyKey;
    // For all other encryption techniques, generate a secret key. 
    else 
      theKey=generateSecretKey(Form.myAlgorithm);
    //Encrypt the string.
    encrypted=encrypt(Form.myString, theKey, Form.myAlgorithm, Form.myEncoding);
    //Decrypt it.
    decrypted=decrypt(encrypted, theKey, Form.myAlgorithm, Form.myEncoding);
  </cfscript>
  <!--- Display the values used for encryption and decryption, and the results. --->
  <cfoutput>
    <b>The algorithm:</b> #Form.myAlgorithm#<br>
    <b>The key:</b> #theKey#<br>
    <br>
    <b>The string:</b> #Form.myString#<br>
    <br>
    <b>Encrypted:</b> #encrypted#<br>
    <br>
    <b>Decrypted:</b> #decrypted#<br>
  </cfoutput>
</cfif>

<!--- The input form. --->
<form action="#CGI.SCRIPT_NAME#" method="post">
  <b>Select the encoding</b><br>
  <select size="1" name="myEncoding" >
    <option selected>UU</option>
    <option>Base64</option>
    <option>Hex</option>
  </select><br>
  <br>
  <b>Select the algorithm</b><br>
  <select size="1" name="myAlgorithm" >
    <option selected>CFMX_COMPAT</option>
    <option>AES</option>
    <option>DES</option>
  </select><br>
</form>
<option>DESEDE</option>
</select>
<br>
<br>
<b>Input your key</b> (used for CFMX_COMPAT encryption only)<br>
<input type = "Text" name = "myKey" value = "foobar"><br>
<br>
<b>Enter string to encrypt</b><br>
<textarea name = "myString" cols = "40" rows = "5" WRAP = "VIRTUAL">This string will be encrypted (you can replace it with more typing).</textarea>
<input type = "Submit" value = "Encrypt my String">
GetAuthUser

Description

Gets the name of an authenticated user.

Returns

The name of an authenticated user.

Category

Security functions

Function syntax

GetAuthUser()

See also

IsUserInRole, cflogin, cfloginuser; Chapter 16, “Securing Applications” in ColdFusion MX Developer’s Guide

History

ColdFusion MX: Added this function.

Usage

This function works with cflogin authentication or web server authentication. It checks for a logged-in user as follows:

1. It checks for a login made with cfloginuser.
2. If no user was logged in with cfloginuser, it checks for a web server login (cgi.remote_user).

Example

<H3>GetAuthUser Example</H3>

<P>Authenticated User: <cfoutput>#{GetAuthUser()}#</cfoutput>
GetBaseTagData

Description

Used within a custom tag. Finds calling (ancestor) tag by name and accesses its data.

Returns

An object that contains data (variables, scopes, and so on) from an ancestor tag. If there is no
ancestor by the specified name, or if the ancestor does not expose data (for example, cfif), an
exception is thrown.

Category

Other functions

Function syntax

GetBaseTagData(tagname [, instancenumber ])

See also

GetBaseTagList; “High-level data exchange” in Chapter 11, “Creating and Using Custom
CFML Tags,” in ColdFusion MX Developer’s Guide

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Req/Opt</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tagname</td>
<td>Required</td>
<td>Ancestor tag name for which to return data</td>
</tr>
<tr>
<td>instancenumber</td>
<td>Optional</td>
<td>Number of ancestor levels to jump before returning data. The default value is 1 (closest ancestor).</td>
</tr>
</tbody>
</table>

Example

<!--- This example shows the use of GetBaseTagData function. Typically used in custom tags.--->

...<cfif trim(inCustomTag) neq "">
<!--- Get the tag instance data --->
<cfset tagData = GetBaseTagData(inCustomTag)>
<!--- Find the tag's execution mode --->
Located inside the
<cfif tagData.thisTag.executionMode neq 'inactive'>
  template
<cfelse>
  BODY
</cfif>
GetBaseTagList

Description

Gets ancestor tag names, starting with the parent tag.

Returns

A comma-delimited list of uppercase ancestor tag names, as a string. The first list element is the current tag. If the current tag is nested, the next element is the parent tag. If the function is called for a top-level tag, it returns an empty string. If an ancestor does not expose data (see GetBaseTagData), its name might not be returned.

Category

Other functions

Function syntax

GetBaseTagList()

See also


Usage

This function does not display the following tags or end tags in the ancestor tag list:

- cfif, cfelseif, cfelse
- cfswitch, cfcase, cfdefaultcase
- cftry, cfcatch

This function displays the following tags only under the following conditions:

- cfloop: if it uses a query attribute
- cfoutput: if at least one of its children is a complex expression
- cfprocessingdirective: if it has at least one other attribute besides pageencoding

Example

<!--- This example shows the use of GetBaseTagList function. Typically used in custom tags. --->
<cfif thisTag.executionMode is "start">
  <!--- Get the tag context stack
  The list will look something like "CFIF,MYTAGNAME..." --->
  <cfset ancestorList = GetBaseTagList()>
  <br><br>Dump of GetBaseTagList output:<br>
  <cfdump var="#ancestorList#"<br><br>
  <!--- Output current tag name --->
  <cfoutput var="#ancestorList#"></cfoutput> <!--- This is custom tag @ListGetAt(ancestorList,1)#</cfoutput><br>
  <!--- Determine whether this is nested inside a loop --->
  <cfset inLoop = ListFindNoCase(ancestorList, "cfloop")>
  <cfif inLoop>
    Running in the context of a cfloop tag.<br>
  </cfif>
</cfif>
GetBaseTemplatePath

**Description**
Gets the absolute path of an application's base page.

**Returns**
The absolute path of the application base page, as a string.

**Category**
Other functions, System functions

**Function syntax**
GetBaseTemplatePath()

**See also**
GetCurrentTemplatePath, FileExists, ExpandPath

**Example**

<h3>GetBaseTemplatePath Example</h3>

<p>The template path of the current page is: <cfoutput>#{GetBaseTemplatePath()}#</cfoutput>
**GetClientVariablesList**

**Description**
Finds the client variables to which a page has write access.

**Returns**
Comma-delimited list of non-read-only client variables, as a string.

**Category**
List functions, Other functions

**Function syntax**
GetClientVariablesList()

**See also**
DeleteClientVariable

**Usage**
The list of variables returned by this function is compatible with ColdFusion list functions.

**Example**

<!--- This example is view-only. --->

```cfm
<h3>GetClientVariablesList Example</h3>
<p>This view-only example deletes a client variable called "User_ID", if it exists
   in the list of client variables returned by GetClientVariablesList().
<p>This example requires the existence of an Application.cfm file and that client
   management be in effect.
<!---
<cfset client.somevar = ">
<cfset client.user_id = ">
<p>Client variable list:<cfoutput>#GetClientVariablesList()#</cfoutput>
<cfif ListFindNoCase(GetClientVariablesList(), "User_ID") is not 0>
<!--- delete that variable
<cfset temp = DeleteClientVariable("User_ID")>
<p>Was variable "User_ID" Deleted? <cfoutput>#temp#</cfoutput>
</cfif>
<p>Amended Client variable list:<cfoutput>#GetClientVariablesList()#</cfoutput>
<!--- -->
```

---
GetContextRoot

Description

Returns path to the J2EE server context root for the current request.

Returns

The path from the web root to the context root for the current page. The path starts with a forward slash character (/) but does not end with a forward slash character (/). For applications in the default (root) context, returns the empty string.

Category

System functions

History

ColdFusion MX 7: Added this function.

Function syntax

GetContextRoot()

See also

GetPageContext

Usage

This function is equivalent to calling GetPageContext().GetRequest().GetContextPath(). On J2EE configurations, it returns the path from the Web root to the J2EE context root of the ColdFusion MX J2EE application. On server configurations, it returns the empty string, because the context root is at the web root.

This function is useful in applications that might be installed at varying J2EE context roots.

Example

The ColdFusion MX Administrator uses the following line to get the location of the administrator directory:

```cfset request.thisURL = "#getContextRoot()#/CFIDE/administrator/"
```

The Administrator uses the returned value in places where it uses a URL to access Administrator resources, such as images, as in the following line:

```<a href="index.cfm"><img src="#request.thisURL#images/back.gif" width="16" height="16" border="0" alt=""></a>```
GetCurrentTemplatePath

Description

Gets the path of the page that calls this function.

Returns

The absolute path of the page that contains the call to this function, as a string.

Category

System functions

Function syntax

GetCurrentTemplatePath()

See also

GetBaseTemplatePath, FileExists, ExpandPath

Usage

If the function call is made from a page included with a cfinclude tag, this function returns the page path of an included page. Contrast this with the GetBaseTemplatePath function, which returns the path of the top-level page, even if it is called from an included page.

Example

<!--- This example uses GetCurrentTemplatePath to show the template path of the current page --->
<h3>GetCurrentTemplatePath Example</h3>
<p>The template path of the current page is:<br>
<cfoutput>#GetCurrentTemplatePath()#</cfoutput>

GetDirectoryFromPath

Description

Extracts a directory from an absolute path.

Returns

Absolute path, without the filename. The last character is a forward or backward slash, depending on the operating system.

Category

System functions

Function syntax

GetDirectoryFromPath(path)

See also

ExpandPath, GetFileFromPath

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>path</td>
<td>Absolute path (drive, directory, filename, and extension)</td>
</tr>
</tbody>
</table>

Example

```cf
<h3>GetDirectoryFromPath Example</h3>
<cfset thisPath = ExpandPath("*.*)">
<cfset thisDirectory = GetDirectoryFromPath(thisPath)>
<cfoutput>
The current directory is: #GetDirectoryFromPath(thisPath)#
<cfif IsDefined("FORM.yourFile")>
  <cfif FORM.yourFile is not ">"
    <cfset yourFile = FORM.yourFile>
    <cfif FileExists(ExpandPath(yourfile))>
      <p>Your file exists in this directory. You entered the correct file name.
      #GetFileFromPath("#thisPath#/"yourfile")#
    <cfelse>
      <p>Your file was not found in this directory:
    <br>Here is a list of the other files in this directory:
    <!--- use cfdirectory show directory, order by name & size --->
    <cfdirectory directory = "#thisDirectory#"
      name = "myDirectory" SORT = "name ASC, size DESC""
    <!--- Output the contents of the cfdirectory as a CFTABLE --->
    <cftable query = "myDirectory">
      <cfcol header = "NAME:" text = "#Name#">
      <cfcol header = "SIZE:" text = "#Size#">
    </cftable>
  </cfif>
</cfif>
<cfelse>
  <H3>Please enter a file name</H3>
</cfif>
```
<form action="getdirectoryfrompath.cfm" METHOD="post">
   <H3>Enter the name of a file in this directory <I><FONT SIZE="-1">(try expandpath.cfm)</FONT></I></H3>
   <input type="Text" NAME="yourFile">
   <input type="Submit" NAME="">
</form>
GetEncoding

**Description**

Returns the encoding (character set) of the Form or URL scope.

**Returns**

String; the character encoding of the specified scope.

**Category**

International functions, System functions

**Function syntax**

GetEncoding(*scope_name*)

**See also**

SetEncoding, cfcontent, cfprocessingdirective, URLEncode, URLEncodedFormat

**History**

ColdFusion MX: Added this function.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>scope_name</td>
<td>• Form</td>
</tr>
<tr>
<td></td>
<td>• URL</td>
</tr>
</tbody>
</table>

**Usage**

Use this function to determine the character encoding of the URL query string or the fields of a form that was submitted to the current page. The default encoding, if none has been explicitly set, is UTF-8.

For more information, see www.iana.org/assignments/character-sets.

**Example**

```cfml
<!--- This example sends the contents of two fields and interprets them as big5 encoded text. --->
<cfcontent type="text/html; charset=big5">
<form action='$cgi.script_name# method='get'>
<input name='xxx' type='text'>
<input name='yyy' type='text'>
<input type="Submit" value="Submit">
</form>
<cfif IsDefined("URL.xxx")>
<cfscript>
SetEncoding("url", "big5");
WriteOutput("URL.XXX is " & URL.xxx & "\n");
WriteOutput("URL.YYY is " & URL.yyy & "\n");
theEncoding = GetEncoding("URL");
WriteOutput("The URL variables were decoded using " & theEncoding & ".");
</cfscript>
```

---
WriteOutput("The encoding is " & theEncoding);
</cfscript>
</cfif>
GetException

Description
Used with the `cftry` and `cfcatch` tags. Retrieves a Java exception object from a Java object.

Returns
Any Java exception object raised by a previous method call on the Java object.

Category
System functions

Syntax
`GetException(object)`

Parameters
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>A Java object.</td>
</tr>
</tbody>
</table>

Usage
ColdFusion stores a Java exception object for each method call on a Java object. Subsequent method calls reset the exception object. To get the current exception object, you must call `GetException` on the Java object before other methods are invoked on it.

Example
```coldfusion
<!---- Create the Java object reference ---->
<cfobject action = create type = java class = primativetype name = myObj>
<!---- Calls the object's constructor ---->
<cfset void = myObj.init()>
<cftry>
<cfset void = myObj.DoException()>
<cfcatch type = "Any">
<cfset exception = GetException(myObj)>
<!---- User can call any valid method on the exception object.---->
<cfset message = exception.toString()>
<cfoutput>
Error<br>
I got exception <br>
The exception message is: #message# <br>
</cfoutput>
</cfcatch>
</cftry>
```
GetFileFromPath

Description
Extracts a filename from an absolute path.

Returns
Filename, as a string.

Category
System functions

Function syntax
GetFileFromPath(path)

See also
ExpandPath, GetCurrentTemplatePath

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>path</td>
<td>Absolute path (drive, directory, filename, and extension)</td>
</tr>
</tbody>
</table>

Example

<h3>GetFileFromPath Example</h3>
<cfset thisPath = ExpandPath(".*")>
<cfset thisDirectory = GetDirectoryFromPath(thisPath)>
<cfoutput>
The current directory is: #GetDirectoryFromPath(thisPath)#
<cfif IsDefined("FORM.yourFile")>
<cfif FORM.yourFile is not ">">
<cfset yourFile = FORM.yourFile>
<cfif FileExists(ExpandPath(yourFile))>
<p>Your file exists in this directory. You entered the correct file name. #GetFileFromPath("#thisPath#/"yourfile")#</p>
<cfelse>
<p>Your file was not found in this directory:<br>Here is a list of the other files in this directory:<br>!--- use cfdirectory to give the contents of the snippets directory, order by name and size --->
<cfdirectory
directory = "#thisDirectory#"
name = "myDirectory"
sort = "name ASC, size DESC">
!--- Output the contents of the cfdirectory as a cftable --->
cftable query = "myDirectory"
<cfcol header = "NAME:" text = "#Name#">
<cfcol header = "SIZE:" text = "#Size#">
...
GetFunctionList

Description
Displays a list of the functions that are available in ColdFusion.

Returns
A structure of functions.

Category
System functions

Function syntax
GetFunctionList()

Example
<!--- This example shows the use of GetFunctionList. ---->
<cfset fList = GetFunctionList()>
<cfoutput>#$StructCount(fList)# functions<br><br>
</cfoutput>
<cfloop collection = "#fList#" item = "key">
  <cfoutput>#$key#$<br>
</cfoutput>
</cfloop>
GetGatewayHelper

Description

Gets a Java GatewayHelper object that provides methods and properties for use with a ColdFusion event gateway.

Returns

A Java GatewayHelper object.

Category

Extensibility functions

Function syntax

GetGatewayHelper(gatewayID)

See also

SendGatewayMessage

History

ColdFusion MX 7: Added the function.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>gatewayID</td>
<td>Identifier of the gateway that provides the GatewayHelper object. Must be the Gateway ID of one of the ColdFusion event gateway instances configured on the ColdFusion MX Administrator Event Gateways section’s Gateways page.</td>
</tr>
</tbody>
</table>

Usage

The ColdFusion GetGatewayHelper function returns a Java GatewayHelper object that provides event gateway-specific helper methods and properties. To use this function, the event gateway must provide access to a class that implements the GatewayHelper class. For example, an instant messaging event gateway might make buddy-list management functions available in a GatewayHelper object.

An event gateway listener CFC can get the gatewayID value from the CFEvent structure of the incoming message.

You access the GatewayHelper object’s methods and properties using standard ColdFusion Java object access techniques. For more information, see “The role of the GatewayHelper object” in Chapter 42, “Using Event Gateways,” in ColdFusion MX Developer’s Guide.
Example

If an event gateway's helper class includes an addBuddy method that takes a single String parameter, you could use the following code to get the GatewayHelper object and add a buddy to the buddies list:

```cfscript
myHelper = getGatewayHelper(myGatewayID);
status = myHelper.addBuddy("jsmith");
</cfscript>`
GetHttpRequestData

**Description**

Makes HTTP request headers and body available to CFML pages. Useful for capturing SOAP request data, which can be delivered in an HTTP header.

**Returns**

A ColdFusion structure.

**Category**

System functions

**Function syntax**

GetHttpRequestData()

**Returns**

The function returns a structure containing the following entries:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>content</td>
<td>Raw content from form submitted by client, in string or binary format. For content to be considered string data, the FORM request header &quot;CONTENT_TYPE&quot; must start with &quot;text/&quot; or be special case &quot;application/x-www-form-urlencoded&quot;. Other types are stored as a binary object.</td>
</tr>
<tr>
<td>headers</td>
<td>Structure that contains HTTP request headers as value pairs. Includes custom headers, such as SOAP requests.</td>
</tr>
<tr>
<td>method</td>
<td>String that contains the CGI variable Request_Method.</td>
</tr>
<tr>
<td>protocol</td>
<td>String that contains the Server_Protocol CGI variable.</td>
</tr>
</tbody>
</table>

**Usage**

To determine whether data is binary, use `IsBinary(x.content)`. To convert data to a string value, if it can be displayed as a string, use `ToString(x.content)`.

**Example**

The following example shows how this function can return HTTP header information.

```cfml
<cfset x = GetHttpRequestData()>
<cfoutput>
<table cellpadding = "2" cellspacing = "2">
<tr>
  <td><b>HTTP Request item</b></td>
  <td><b>Value</b></td>
</tr>
<cfloop collection = #x.headers# item = "http_item">
  <tr>
    <td>#http_item#</td>
    <td>#StructFind(x.headers, http_item)#</td>
  </tr>
</cfloop>
<tr>
  <td>request_method</td>
  <td>#x.method#</td>
</tr>
</table>
</cfoutput>
```
<tr>
  <td>server_protocol</td>
  <td>#x.protocol#</td></tr>
</table>

<b>http_content --- #x.content#</b>
</cfoutput>
GetHttpTimeString

Description
Gets the current time, in the Universal Time code (UTC).

Returns
The time, as a string, according to the HTTP standard described in RFC 1123 and its underlying RFC, 822. This format is commonly used in Internet protocols, including HTTP.

Category
Date and time functions, International functions

Function syntax
GetHttpTimeString(\textit{date\_time\_object})

See also
GetLocale, GetTimeZoneInfo, SetLocale

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>date_time_object</td>
<td>A ColdFusion date-time object string or Java Date object</td>
</tr>
</tbody>
</table>

Usage
The time in the returned string is UTC, consistent with the HTTP standard.

Example

```coldfuse
<cfoutput>
#GetHttpTimeString(now())#
</cfoutput>
```
GetK2ServerDocCount

Description
This function is deprecated.

Returns
The number of collection metadata items stored in Verity collections.

Category
Full-text search functions, Query functions

Function syntax
GetK2ServerDocCount()

See also
GetK2ServerDocCountLimit

History
ColdFusion MX 6.1: Deprecated this function. It might not work, and it might cause an error, in later releases.
ColdFusion MX: Added this function.

Example
<cfoutput>GetK2ServerDocCount =
  $*#GetK2ServerDocCount()#*$</cfoutput>
GetK2ServerDocCountLimit

Description
This function is deprecated.

Returns
Number of collection metadata items that the K2 server permits, as an integer

Category
Full-text search functions, Query functions

Function syntax
GetK2ServerDocCountLimit()

See also
GetK2ServerDocCount

History
ColdFusion MX 6.1: Deprecated this function. It might not work, and it might cause an error, in later releases.
ColdFusion MX: Added this function.

Usage
If a search generates a larger number of documents than the limit, ColdFusion puts a warning message in the Administrator and in the log file.

Example
GetLocale

Description

Gets the current ColdFusion MX geographic/language locale value.

To set the default display format of date, time, number, and currency values in a ColdFusion application session, you use the SetLocale function.

Returns

The current locale value, as an English string. If a locale has a Java name and a name that ColdFusion MX used prior to the ColdFusion MX 7 release (for example, en_US and English (US)), ColdFusion MX returns the ColdFusion name (for example, English (US)).

Category

Display and formatting functions, International functions, System functions

Function syntax

GetLocale()

See also

GetLocaleDisplayName, SetLocale

History

ColdFusion MX 7: Added support for all Java locales and locale names.
ColdFusion MX: Changed behavior to that described in usage.

Usage

This function returns the locale name as it is represented in ColdFusion MX; for example, Portuguese (Brazilian), or ca_ES. To get a locale name in the language of the locale, use the GetLocaleDisplayName function, which returns português (Brasil) and català (Espanya).

This function determines whether a locale value is set for ColdFusion MX. (The value is set with the SetLocale function.)

- If the ColdFusion MX locale value is present, the function returns it.
- If the ColdFusion MX locale has not been explicitly set, ColdFusion now determines whether the default locale of the ColdFusion server computer operating system is among the locale values it supports. (The default locale is stored in the user environment variables user.language and user.region.)

If the default locale value is not supported, the function returns English (US). ColdFusion sets the locale in the JVM to this value; it persists until the server is restarted or it is reset with the SetLocale function.

This function does not access a web browser’s Accept-Language HTTP header setting.

Note: When ColdFusion is started, it stores the supported locale values in the variable Server.ColdFusion.SupportedLocales. ColdFusion MX supports the locales supported by its Java runtime environment. The SupportedLocales value lists the Java names for the supported locales and the corresponding names that ColdFusion MX used prior to the ColdFusion MX 7 release.
For more information, see “Locales” in Chapter 17, “Developing Globalized Applications,” in ColdFusion MX Developer’s Guide.

Example

<h3>Example: Using SetLocale and GetLocale</h3>

```coldfusion
<!--- For each new request, the locale gets reset to the JVM locale --->
Initial locale's ColdFusion name: #GetLocale()#<br>
<br>
<!--- Do this only if the form was submitted. --->
<cfif isDefined("form.mylocale")>
  <b>Changing locale to #form.mylocale#</b><br>
  <br>
  <!--- Set the locale to the submitted value and save the old ColdFusion locale name. --->
  <cfset oldlocale=SetLocale("#form.mylocale#")>
  <br>
  <!--- Get the current locale. It should have changed. --->
  New locale: #GetLocale()#<br>
</cfif>

<!--- Self-submitting form to select the new locale. --->
<cfform>
  <h3>Please select the new locale:</h3>
  <cfselect name="mylocale">
  <!--- The server.coldfusion.supportedlocales variable is a list of all supported locale names. Use a list cfloop tag to create an HTML option tag for each name in the list. --->
  <cfloop index="i" list="#server.coldfusion.supportedlocales#">
    <option value="#i#">#i#</option>
  </cfloop>
  </cfselect><br>
  <br>
  <cfinput type="submit" name="submitit" value="Change Locale">
</cfform>
```

</cfoutput>
GetLocaleDisplayName

Description
Gets a locale value and displays the name in a manner that is appropriate to a specific locale. By default, gets the current locale in the current locale's language.

Returns
The localized display name of the locale, in the language of the specified locale.

Category
Display and formatting functions, International functions, System functions

Function syntax
GetLocaleDisplayName([locale[, inLocale]])

See also
GetLocale, SetLocale

History
ColdFusion MX 7: Added this function.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>locale</td>
<td>The locale whose name you want. The default is the current ColdFusion locale, or if no ColdFusion locale has been set, the JVM locale.</td>
</tr>
<tr>
<td>inlocale</td>
<td>The locale in which to return the name. The default is the current ColdFusion locale, or if no ColdFusion locale has been set, the JVM locale.</td>
</tr>
</tbody>
</table>

Example
The following example expands on the GetLocale example to show the use of the GetLocaleDisplayName function to display locale names in the current locale and in other locales. It lets you select a locale from all supported locales, changes the ColdFusion MX locale to the selected locales, and displays the old and new locale names.

```html
<html>
<head>
  <title>Displaying locales</title>
</head>

<body>
  <h3>Example: Changing and Displaying Locales</h3>
  <cfoutput>
    <!--- For each new request, the locale gets reset to the JVM locale --->
    Initial locale's ColdFusion name: #GetLocale()#<br>
    Initial locale's display name: #GetLocaleDisplayName()#<br>
    <!--- Do this only if the form was submitted. --->
    <cfif IsDefined("form.mylocale")>
      Changing locale to #form.mylocale#<br>
    </cfif>
  </cfoutput>
</body>
```

618 Chapter 3: ColdFusion Functions
<br>
<!--- Set the locale to the submitted value. --->
<cfset oldlocale=SetLocale("#form.mylocale#")>
<!--- Get the current locale's ColdFusion name. --->
<cfset newlocale=GetLocale()>
New locale's ColdFusion name: #newlocale#<br>
New locale's display name in current locale: #GetLocaleDisplayName()#<br>
New locale's display name in old locale:
  #GetLocaleDisplayName(newlocale, oldlocale)#<br>
New locale's display name in en_US:
  #GetLocaleDisplayName(newlocale, "en_US")#<br>
<br>
Old locale's display name in current locale:
  #GetLocaleDisplayName(oldlocale)#<br>
Old locale's display name in en_US:
  #GetLocaleDisplayName(oldlocale, "en_US")#<br>
</cfif>
<!--- Self-submitting form to select the new locale. --->
<cfform>
<h3>Please select the new locale:</h3>
<cfselect name="mylocale">
<!--- The server.coldfusion.supportedlocales variable is a
  list of all supported locale names. Use a list cfloop tag
  to create an HTML option tag for each name in the list. --->
<cfloop index="i" list="#server.coldfusion.supportedlocales#">
<!--- In the select box, we use the US English display names for
  the locales. You can change en_US to your prefered locale. --->
  <option value="#i#" #GetLocaleDisplayName(i, "en_US")#</option>
</cfloop>
</cfselect>
<br>
<br>
<cfinput type="submit" name="submitit" value="Change Locale">
</cfform>
</cfoutput>
</body>
</html>
GetMetaData

Description
Gets metadata (such as the methods, properties, and parameters of a component) associated with an object that is deployed on the ColdFusion server.

Returns
Structured metadata information: for ColdFusion components (CFCs) and user defined functions (UDFs), a structure; for query objects, an array of structures.

Category
System functions

Function syntax
GetMetaData(object)

See also
CreateObject, QueryAddColumn, QueryNew

History
ColdFusion MX 7: Added support for getting query object metadata.
ColdFusion MX: Added this function.

Parameters
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>A ColdFusion component, user-defined function, or query object. Within a CFC, the parameter can also specify the This scope.</td>
</tr>
</tbody>
</table>

Usage
This function provides information about application data, and lets applications dynamically determine the structure of an object and how to use it. This function is useful for CFCs and query objects. The metadata for a CFC includes information on the component and its functions, arguments, and properties. The getMetaData function also returns metadata for user-defined functions that not part of CFCs.
The following table lists the data returned by the function for supported object types:

<table>
<thead>
<tr>
<th>Object</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Component</td>
<td>displayname</td>
<td>Value of the <code>cfcomponent</code> tag <code>displayname</code> attribute, if any.</td>
</tr>
<tr>
<td></td>
<td>extends</td>
<td>Metadata for the component’s ancestor component. Components that do not explicitly extend another component extend the WEB-INF.cftags.component.</td>
</tr>
<tr>
<td></td>
<td>functions</td>
<td>Array of metadata structures for the component’s functions.</td>
</tr>
<tr>
<td></td>
<td>hint</td>
<td>Value of the <code>cfcomponent</code> tag <code>displayname</code> attribute, if any.</td>
</tr>
<tr>
<td></td>
<td>name</td>
<td>Component name, including the period-delimited path from a component search root such as the web root or a directory specified in the administrator Custom Tag Paths page.</td>
</tr>
<tr>
<td></td>
<td>output</td>
<td>Value of the <code>cfcomponent</code> tag <code>output</code> attribute, if any.</td>
</tr>
<tr>
<td></td>
<td>path</td>
<td>Absolute path to the component.</td>
</tr>
<tr>
<td></td>
<td>properties</td>
<td>Array of structures containing metadata specified by the component’s <code>cfproperty</code> tags, if any.</td>
</tr>
<tr>
<td></td>
<td>type</td>
<td>Always, component.</td>
</tr>
<tr>
<td></td>
<td>userMetadata</td>
<td>User-specified attributes in the <code>cfcomponent</code> tag</td>
</tr>
<tr>
<td>Function</td>
<td>access</td>
<td>Value of the <code>cffunction</code> tag <code>access</code> attribute, if any.</td>
</tr>
<tr>
<td></td>
<td>displayname</td>
<td>Value of the <code>cffunction</code> tag <code>displayname</code> attribute, if any.</td>
</tr>
<tr>
<td></td>
<td>hint</td>
<td>Value of the <code>cffunction</code> tag <code>hint</code> attribute, if any.</td>
</tr>
<tr>
<td></td>
<td>name</td>
<td>Function name.</td>
</tr>
<tr>
<td></td>
<td>output</td>
<td>Value of the <code>cffunction</code> tag <code>output</code> attribute, if any.</td>
</tr>
<tr>
<td></td>
<td>parameters</td>
<td>Array of structures containing metadata for the function parameters.</td>
</tr>
<tr>
<td></td>
<td>returntype</td>
<td>Value of the <code>cffunction</code> tag <code>returntype</code> attribute, if any.</td>
</tr>
<tr>
<td></td>
<td>roles</td>
<td>Value of the <code>cffunction</code> tag <code>output</code> attribute, if any.</td>
</tr>
<tr>
<td></td>
<td>userMetadata</td>
<td>User-specified attributes in the <code>cffunction</code> tag</td>
</tr>
</tbody>
</table>
Note: Use the This scope to access component metadata inside the CFC. The This scope is available at runtime in the component body and in the CFC methods. It is used to read and write variables that are present during the life of the component.

For more information, see “Using introspection to get information about components” in Chapter 10, “Building and Using ColdFusion Components,” in ColdFusion MX Developer’s Guide.

Example

The following example uses the cfdump tag to display metadata for the utilities CFC that is used by the ColdFusion component browser. It also displays the names and data types of the fields in the cfdocexamples database Employees table.

<!--- Create an instance of the Component Explorer utilities CFC. and get its metadata --->
<cfscript>
componentutils = createObject("component", "cfide.componentutils.utils");
utilmetadata = getMetaData(componentutils);
</cfscript>

<h4>Metadata for the CFC component utilities</h4>
<cfdump var="#utilmetadata#">

<table>
<thead>
<tr>
<th>Object</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parameter or Property</td>
<td>default</td>
<td>Value of the cfargument or cfproperty tag default attribute, if any.</td>
</tr>
<tr>
<td></td>
<td>displayname</td>
<td>Value of the cfargument or cfproperty tag displayname attribute, if any.</td>
</tr>
<tr>
<td></td>
<td>hint</td>
<td>Value of the cfargument or cfproperty tag hint attribute, if any.</td>
</tr>
<tr>
<td></td>
<td>name</td>
<td>Function parameter or CFC property name.</td>
</tr>
<tr>
<td></td>
<td>required</td>
<td>Value of the cfargument or cfproperty tag required attribute, if any.</td>
</tr>
<tr>
<td></td>
<td>type</td>
<td>Value of the cfargument or cfproperty tag type attribute, if any.</td>
</tr>
<tr>
<td></td>
<td>userMetadata</td>
<td>User-specified attributes in the argument tag.</td>
</tr>
<tr>
<td>Query</td>
<td>IsCaseSensitive</td>
<td>Boolean value indicating whether character data must be case correct.</td>
</tr>
<tr>
<td></td>
<td>Name</td>
<td>The column name.</td>
</tr>
<tr>
<td></td>
<td>TypeName</td>
<td>The SQL data type (Omitted if the query object is created with QueryNew without specifying types.)</td>
</tr>
</tbody>
</table>

Object Field Description
---
Parameter or Property | A structure containing the following fields:
| default | Value of the cfargument or cfproperty tag default attribute, if any.
| displayname | Value of the cfargument or cfproperty tag displayname attribute, if any.
| hint | Value of the cfargument or cfproperty tag hint attribute, if any.
| name | Function parameter or CFC property name.
| required | Value of the cfargument or cfproperty tag required attribute, if any.
| type | Value of the cfargument or cfproperty tag type attribute, if any.
| userMetadata | User-specified attributes in the argument tag.

Query | An array of structures containing the following elements:
| IsCaseSensitive | Boolean value indicating whether character data must be case correct.
| Name | The column name.
| TypeName | The SQL data type (Omitted if the query object is created with QueryNew without specifying types.)
<cfquery name="getemployees" datasource="cfdocexamples">
SELECT      *
FROM         Employees
</cfquery>
<cfset employeemeta=getMetaData(getemployees)>

<h4>The Employees table has the following columns</h4>
<cfloop index="i" from="1" to="#arrayLen(employeemeta)#">
   <cfoutput>
      #employeemeta[i].name# #employeemeta[i].TypeName#
      #employeemeta[i].isCaseSensitive#<br>
   </cfoutput>
</cfloop>
GetMetricData

Description
Gets server performance metrics.

Returns
ColdFusion structure that contains metric data, depending on the mode value.

Category
System functions

Function syntax
GetMetricData(mode)

History
ColdFusion MX: Deprecated the cachepops parameter. It might not work, and it might cause an error, in later releases.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mode</td>
<td>perf_monitor</td>
<td>Returns internal data, in a structure. To receive data, you must enable PerfMonitor in ColdFusion Administrator before executing the function. On Windows, this data is otherwise displayed in the Windows PerfMonitor.</td>
</tr>
<tr>
<td></td>
<td>simple_load</td>
<td>Returns an integer value that is computed from the state of the server’s internal queues. Indicates the overall server load.</td>
</tr>
<tr>
<td></td>
<td>prev_req_time</td>
<td>Returns the time, in milliseconds, that it took the server to process the previous request.</td>
</tr>
<tr>
<td></td>
<td>avg_req_time</td>
<td>Returns the average time, in milliseconds, that it takes the server to process a request. Changing the setting to 0 prevents the server from calculating the average and removes overhead associated with gathering data. The default value is 120 seconds.</td>
</tr>
</tbody>
</table>

Usage

If mode = "perf_monitor", the function returns a structure with these data fields:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>InstanceName</td>
<td>The name of the ColdFusion server. The default value is cfserver.</td>
</tr>
<tr>
<td>PageHits</td>
<td>Number of HTTP requests received since ColdFusion MX was started.</td>
</tr>
<tr>
<td>ReqQueued</td>
<td>Number of HTTP requests in the staging queue, waiting for processing.</td>
</tr>
<tr>
<td>DBHits</td>
<td>Number of database requests since the server was started.</td>
</tr>
</tbody>
</table>
Example

<!--- This example gets and displays metric data from Windows NT PerfMonitor --->
<cfset pmData = GetMetricData( "PERF_MONITOR" ) >
<cfoutput>
  Current PerfMonitor data is: <p>
  InstanceName: #pmData.InstanceName# <p>
  PageHits: #pmData.PageHits# <p>
  ReqQueued: #pmData.ReqQueue# <p>
  DBHits: #pmData.DBHits# <p>
  ReqRunning: #pmData.ReqRunning# <p>
  ReqTimedOut: #pmData.ReqTimedOut# <p>
  BytesIn: #pmData.BytesIn# <p>
  BytesOut: #pmData.BytesOut# <p>
  AvgQueueTime: #pmData.AvgQueueTime# <p>
  AvgReqTime: #pmData.AvgReqTime# <p>
  AvgDBTime: #pmData.AvgDBTime# <p>
</cfoutput>

Field | Description
--- | ---
ReqRunning | Number of HTTP requests currently running. In the ColdFusion Administrator, you can set the maximum number of requests that run concurrently.
ReqTimedOut | Number of HTTP requests that timed out while in the staging queue or during processing.
BytesIn | Number of bytes in HTTP requests to ColdFusion MX.
BytesOut | Number of bytes in HTTP responses from ColdFusion MX.
AvgQueueTime | For the last two HTTP requests (current and previous), the average length of time the request waited in the staging queue.
AvgReqTime | For the last two HTTP requests (current and previous), the average length of time the server required to process the request.
AvgDBTime | For the last two HTTP requests (current and previous), the average length of time the server took to process CFQueries in the request.
cacheops | This parameter is deprecated. ColdFusion automatically sets its value to -1.
GetPageContext

Description

Gets the current ColdFusion MX PageContext object that provides access to page attributes and configuration, request, and response objects.

Returns

The current ColdFusion MX Java PageContext Java object.

Category

System functions

Function syntax

GetPageContext()

History

ColdFusion MX: Added this function.

Usage

The ColdFusion MX PageContext class is a wrapper class for the Java PageContext object that can resolve scopes and perform case-insensitive variable lookups.

The PageContext object exposes fields and methods that can be useful in J2EE integration. It includes the include and forward methods that provide the equivalent of the corresponding standard JSP tags. You use these methods to call JSP pages and servlets. For example, you use the following code in CFScript to include the JSP page hello.jsp and pass it a name parameter:

GetPageContext().include("hello.jsp?name=Bobby");

When you use GetPageContext to include a JSP page in a CFML page on WebLogic, you may need to flush the output of the CFML page with cfflush before calling the JSP page. Otherwise, the ColdFusion output appears after the JSP output.

For more information, see your Java Server Pages (JSP) documentation.

Example

<!--- this example shows using the page context to set a page variable and access the language of the current locale --->
<cfset pc = GetPageContext()>
<cfset pc.setAttribute("name","John Doe")>
<cfoutput>name: #variables.name#<br></cfoutput>
<cfoutput>Language of the current locale is #pc.getRequest().getLocale().getDisplayLanguage()#</cfoutput>.
GetProfileSections

Description

Gets all the sections of an initialization file.

An initialization file assigns values to configuration variables, also known as entries, that are set when the system boots, the operating system comes up, or an application starts. An initialization file has the suffix INI; for example, boot.ini, Win32.ini.

Returns

An initialization file, as a struct whose format is as follows:

- Each initialization file section name is a key in the struct
- Each list of entries in a section of an initialization file is a value in the struct

If there is no value, returns an empty string.

Category

System functions

Function syntax

GetProfileSections(iniFile)

See also

GetProfileString, SetProfileString

History

ColdFusion MX: Added this function.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>iniFile</td>
<td>Absolute path (drive, directory, filename, extension) of initialization file; for example, C:\boot.ini</td>
</tr>
</tbody>
</table>
GetProfileString

**Description**

Gets an initialization file entry.

An initialization file assigns values to configuration variables, also known as entries, that are set when the system boots, the operating system comes up, or an application starts. An initialization file has the suffix INI; for example, boot.ini, Win32.ini.

**Returns**

An entry in an initialization file, as a string. If there is no value, returns an empty string.

**Category**

System functions

**Function syntax**

GetProfileString(iniPath, section, entry)

**See also**

GetProfileSections, GetProfileString, SetProfileString

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>iniPath</td>
<td>Absolute path (drive, directory, filename, extension) of initialization file; for example, C:\boot.ini</td>
</tr>
<tr>
<td>section</td>
<td>Section of initialization file from which to extract information</td>
</tr>
<tr>
<td>entry</td>
<td>Name of value to get</td>
</tr>
</tbody>
</table>

**Example**

<h3>GetProfileString Example</h3>

Uses GetProfileString to get the value of timeout in an initialization file.

Enter the full path of your initialization file, and submit the form.

<!--- If the form was submitted, it gets the initialization path and timeout value specified in the form --->

```
<cfif Isdefined("Form.Submit")>
  <cfset IniPath = FORM.iniPath>
  <cfset Section = "boot loader">
  <cfset timeout = GetProfileString(IniPath, Section, "timeout")>
  </cfset>
</cfif>

<cfif timeout>
  <h4>Boot Loader</h4>
  Timeout is set to: <cfoutput>#timeout#</cfoutput>.</cfoutput>
</cfif>
```

<form action = "getprofilestring.cfm">
<table cellspacing = "2" cellpadding = "2" border = "0">
  <tr>
    <td>Full Path of Init File</td>
    <td><input type = "Text" name = "IniPath" value = "C:\myboot.ini"></td>
  </tr>
</table>
</form>
<tr>
  <td><input type = "Submit" name = "Submit" value = "Submit"></td>
  <td></td>
</tr></table>
</form>
GetSOAPRequest

Description
Returns an XML object that contains the entire SOAP request. Usually called from within a web service CFC.

Returns
An XML object that contains the entire SOAP request.

Category
XML functions

History
ColdFusion MX 7: Added this function.

Function syntax
GetSOAPRequest()

See also

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>webservice</td>
<td>Optional. A webservice object as returned from the cfoject tag or the CreateObject function. Required if the function is called from the client.</td>
</tr>
</tbody>
</table>

Usage
Call this function to obtain a web service request object after the web service has been invoked. If you call this function from outside a web service CFC without the webservice parameter, it throws the following expression error:

Unable to use getSOAPRequest: not processing a web service request.

If you call this function from within a web service CFC, you can omit the webservice argument. The function executes against the request that it is currently processing.

You can use CFML XML functions to examine the returned XML object.

Example
This example makes a request to execute the echo_me function of the headerservice.cfc web service. For information on implementing the headerservice.cfc web service and also to see the echo_me function and the content of the web service CFC, see the example for either the AddSOAPResponseHeader function or the GetSOAPRequestHeader function.

<!--- Note that you might need to modify the URL in the CreateObject function to match your server and the location of the headerservice.cfc file if it is
different than shown here.
Note, too, that getSOAPRequest is called from the client here, whereas often it
would be called from within the web service CFC. --->

<cfscript>
    ws = CreateObject("webservice",
    "http://localhost/soapheaders/headerservice.cfc?WSDL");
    ws.echo_me("hello world");
    req = getSOAPRequest(ws);
</cfscript>
<cfdump var="#req#"/>
GetSOAPRequestHeader

Description
Obtains a SOAP request header. Call only from within a CFC web service function that is processing a request as a SOAP web service.

Returns
A SOAP request header.

Category
XML functions

History
ColdFusion MX 7: Added this function.

Function syntax
GetSOAPRequestHeader(namespace, name [, asXML])

See also

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>namespace</td>
<td>A String that is the namespace for the header.</td>
</tr>
<tr>
<td>name</td>
<td>A String that is the name of the header.</td>
</tr>
<tr>
<td>asXML</td>
<td>If True, the header is returned as a CFML XML object; if False (default), the header is returned as a Java object.</td>
</tr>
</tbody>
</table>

Usage
If you specify False for the asXML parameter, ColdFusion first attempts to retrieve the header using the data type specified in the header's xsi:type attribute. If the xsi:type attribute is not available, ColdFusion attempts to retrieve the header as a string. If you specify True for the asXML parameter, ColdFusion retrieves the header as raw XML.

This function throws an error if it is invoked in a context that is not a web service request. Use the IsSOAPRequest function to determine whether the CFC is running as a web service.

Example
This example creates a CFC web service that illustrates the operation of the GetSOAPRequestHeader function and also provides a web service that illustrates the operation of other ColdFusion SOAP functions.

Save the following code as headerservice.cfc in a folder called soapheaders under your web root.
Test its operation, and specifically the operation of the GetSOAPRequestHeader function, by executing the examples that invoke this web service. For example, see the example for AddSOAPRequestHeader.
<h3>GetSOAPRequestHeader Example</h3>
<cfcomponent displayName="tester" hint="Test for SOAP headers">
<cffunction name="echo_me"
access="remote"
output="false"
returntype="string"
displayname="Echo Test" hint="Header test">
<cfargument name="in_here" required="true" type="string">
<cfset isSOAP = isSOAPRequest()>
<cfif isSOAP>
<!--- Get the first header as a string and as XML --->
<cfset username = getSOAPRequestHeader("http://mynamespace/", "username")>
<cfset return = "The service saw username: " & username>
<cfset xmlusername = getSOAPRequestHeader("http://mynamespace/", "username", "TRUE")>
<cfset return = return & "<br> as XML: " & xmlusername>

<!--- Get the second header as a string and as XML --->
<cfset password = getSOAPRequestHeader("http://mynamespace/", "password")>
<cfset return = return & "The service saw password: " & password>
<cfset xmlpassword = getSOAPRequestHeader("http://mynamespace/", "password", "TRUE")>
<cfset return = return & "<br> as XML: " & xmlpassword>

<!--- Add a header as a string --->
<cfset addSOAPResponseHeader("http://www.tomj.org/myns/", "returnheader", "AUTHORIZED VALUE", false)>

<!--- Add a second header using a CFML XML value --->
<cfset doc = XmlNew()>
<cfset x = XmlElemNew(doc, "http://www.tomj.org/myns/", "returnheader2")>
<cfset x.XmlText = "hey man, here I am in XML">
<cfset x.XmlAttributes["xsi:type"] = "xsd:string">
<cfset tmp = addSOAPResponseHeader("ignoreNameSpace", "ignoreName", x)>
</cfif>
<cfelse>
<!--- Add a header as a string - Must generate error! --->
<cfset addSOAPResponseHeader("http://www.tomj.org/myns/", "returnheader", "AUTHORIZED VALUE", false)>
--->
<cfset return = "Not invoked as a web service">
</cfif>

<cfreturn return>
</cffunction>
</cfcomponent>
GetSOAPResponse

**Description**

Returns an XML object that contains the entire SOAP response after invoking a web service.

**Returns**

An XML object that contains the entire SOAP response.

**Category**

XML functions

**History**

ColdFusion MX 7: Added this function.

**Function syntax**

GetSOAPResponse(webservice)

**See also**


**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>webservice</td>
<td>A webservice object as returned from the <code>cfobject</code> tag or the <code>CreateObject</code> function.</td>
</tr>
</tbody>
</table>

**Usage**

You must first invoke the web service before attempting to get the response. You can use CFML XML functions to examine the XML response.

**Example**

This example makes a request to execute the `echo_me` function of the headerservice.cfc web service. Following the request, the example calls the `GetSOAPResponse` function to get the SOAP response, and then calls `cfdump` to display its content.

for information on implementing the headerservice.cfc web service and also to see the `echo_me` function and the content of the web service CFC, see the example for either the `AddSOAPResponseHeader` function or the `GetSOAPRequestHeader` function.

<!--- Note that you might need to modify the URL in the CreateObject function to match your server and the location of the headerservice.cfc file if it is different than shown here. --->

```cfs
<cfscript>
    ws = CreateObject("webservice", "http://localhost/soapheaders/headerservice.cfc?WSDL");
    ws.echo_me("hello world");
    resp = getSOAPResponse(ws);
</cfscript>
<cfdump var="#resp#"
GetSOAPResponseHeader

Description

Returns a SOAP response header. Call this function from within code that is invoking a web service after making a web service request.

Returns

A SOAP response header.

Category

XML functions

History

ColdFusion MX 7: Added this function.

Function syntax

GetSOAPResponseHeader(webservice, namespace, name [, asXML])

See also


Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>webservice</td>
<td>A webservice object as returned from the cfobject tag or the CreateObject function.</td>
</tr>
<tr>
<td>namespace</td>
<td>A String that is the namespace for the header.</td>
</tr>
<tr>
<td>name</td>
<td>A String that is the name of the SOAP header.</td>
</tr>
<tr>
<td>asXML</td>
<td>If True, the header is returned as a CFML XML object. If False (default), the header is returned as a Java object.</td>
</tr>
</tbody>
</table>

Usage

If you specify false for the asXML parameter, ColdFusion first attempts to retrieve the header using the data type specified in the header’s xsi:type attribute. If the xsi:type attribute is not available, ColdFusion attempts to retrieve the header as a string. If you specify True for the asXML parameter, ColdFusion retrieves the header as raw XML.

Used within CFML code by a consumer of a web service after it calls the web service with cfinvoke.

Example

There are two parts to this example. The first part is the web service CFC that this function (as well as the other ColdFusion SOAP functions) uses to demonstrate its interaction with a web service. To implement the web service for this function, see the example for either the AddSOAPResponseHeader function or the GetSOAPRequestHeader function.
Execute the following example to see how the `GetSOAPResponseHeader` function operates:

```cfc
<!--- Note that you might need to modify the URL in the CreateObject function to match your server and the location of the headerservice.cfc file if it is different than shown here. Likewise for the cfinvoke tag at the end --->

<h3>GetSOAPResponseHeader Example</h3>
<cfscript>
// Create the web service object
ws = CreateObject("webservice", "http://localhost/soapheaders/headerservice.cfc?WSDL");

// Set the username header as a string
addSOAPRequestHeader(ws, "http://mynamespace/", "username", "tom", false);

// Set the password header as a CFML XML object
doc = XmlNew();
doc.password = XmlElemNew(doc, "http://mynamespace/", "password");
doc.password.XmlText = "My Voice is my Password";
doc.password.XmlAttributes["xsi:type"] = "xsd:string";
addSOAPRequestHeader(ws, "ignoredNameSpace", "ignoredName", doc);

// Invoke the web service operation
ret = ws.echo_me("argument");

// Get the first header as an object (string) and as XML
header = getSOAPResponseHeader(ws, "http://www.tomj.org/myns", "returnheader");
XMLheader = getSOAPResponseHeader(ws, "http://www.tomj.org/myns", "returnheader", true);

// Get the second header as an object (string) and as XML
header2 = getSOAPResponseHeader(ws, "http://www.tomj.org/myns", "returnheader2");
XMLheader2 = getSOAPResponseHeader(ws, "http://www.tomj.org/myns", "returnheader2", true);
</cfscript>
<hr>
<cfoutput>
Soap Header value: #HTMLCodeFormat(header)#<br>
Soap Header XML value: #HTMLCodeFormat(XMLheader)#<br>
Soap Header 2 value: #HTMLCodeFormat(header2)#<br>
Soap Header 2 XML value: #HTMLCodeFormat(XMLheader2)#<br>
Return value: #HTMLCodeFormat(ret)#<br>
</cfoutput>
<hr>
<cfinvoke component="soapheaders.headerservice" method="echo_me" returnvariable="ret" in_here="hi"> </cfinvoke>
<cfoutput>Cfinvoke returned: #ret#</cfoutput>
```
**GetTempDirectory**

**Description**

Gets the path of the directory that ColdFusion uses for temporary files. The directory depends on the account under which ColdFusion is running and other factors. Before using this function in an application, test to determine the directory it returns under your account.

**Returns**

The absolute pathname of a directory, including a trailing slash, as a string.

**Category**

*System functions*

**Function syntax**

GetTempDirectory()

**See also**

GetTempFile

**History**

ColdFusion MX: Changed behavior: on Windows, this function now returns the temporary directory of the embedded Java application server. On other platforms, it returns the temporary directory of the operating system.

**Example**

```html
<h3>GetTempDirectory Example</h3>

<p>The temporary directory for this ColdFusion server is <cfoutput>#GetTempDirectory()#</cfoutput>.

<p>We have created a temporary file called: <cfoutput>#GetTempFile(GetTempDirectory(),"testFile")#</cfoutput>
```
GetTempFile

Description

Creates a temporary file in a directory whose name starts with (at most) the first three characters of prefix.

Returns

Name of a temporary file, as a string.

Category

System functions

Function syntax

GetTempFile(dir, prefix)

See also

GetTempDirectory

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dir</td>
<td>Directory name</td>
</tr>
<tr>
<td>prefix</td>
<td>Prefix of a temporary file to create in the dir directory</td>
</tr>
</tbody>
</table>

Example

<h3>GetTempFile Example</h3>
<p>The temporary directory for this ColdFusion Server is <cfoutput>#GetTempDirectory()#</cfoutput>. We have created a temporary file called: <cfoutput>#GetTempFile(GetTempDirectory(),"testFile")#</cfoutput>
GetTemplatePath

Description

This function is deprecated. Use the GetBaseTemplatePath function instead.

Gets the absolute path of an application's base page.

History

ColdFusion MX: Deprecated this function. It might not work, and it might cause an error, in later releases.
GetTickCount

Description
Returns the current value of an internal millisecond timer.

Returns
A string representation of the system time, in milliseconds.

Category
Date and time functions, System functions

Function syntax
GetTickCount()

Usage
This function is useful for timing CFML code segments or other page processing elements. The value of the counter has no meaning. To generate useful timing values, take the difference between the results of two GetTickCount calls.

Example
<cfset iterationCount = 1000>
<cfset tickBegin = GetTickCount()>
<cfloop Index = i From = 1 To = #iterationCount#></cfloop>
<cfset tickEnd = GetTickCount()>
<cfset loopTime = tickEnd - tickBegin>
<cfoutput>Loop time (#iterationCount# iterations) was: #loopTime# milliseconds</cfoutput>
GetTimeZoneInfo

Description

Gets local time zone information for the computer on which it is called, relative to Universal Time Coordinated (UTC). UTC is the mean solar time of the meridian of Greenwich, England, used as the basis for calculating standard time throughout the world.

ColdFusion stores date and time values as date-time objects: real numbers on a universal time line. It converts an object to a time zone when it formats an object; it converts a date/time value from a time zone to a real number when it parses a value.

Returns

Structure that contains these elements and keys:

• utcTotalOffset: offset of local time, in seconds, from UTC
  ■ A plus sign indicates a time zone west of UTC (such as a zone in North America)
  ■ A minus sign indicates a time zone east of UTC (such as a zone in Germany)
• utcHourOffset: offset, in hours of local time, from UTC
• utcMinuteOffset: offset, in minutes, beyond the hours offset. For North America, this is 0. For countries that are not exactly on the hour offset, the number is between 0 and 60. For example, standard time in Adelaide, Australia is offset 9 hours and 30 minutes from UTC.
• isDSTOn: True, if Daylight Savings Time (DST) is on in the host; False, otherwise

Category

Date and time functions, International functions

Function syntax

GetTimeZoneInfo()

See also

DateConvert, CreateDateTime, DatePart

Example

<h3>GetTimeZoneInfo Example</h3>
<!--- This example shows the use of GetTimeZoneInfo --->
<cfoutput>
The local date and time are #now()#.
</cfoutput>
<cfset info = GetTimeZoneInfo()>
<cfoutput>
<p>Total offset in seconds is #info.utcTotalOffset#.</p>
<p>Offset in hours is #info.utcHourOffset#.</p>
<p>Offset in minutes minus the offset in hours is #info.utcMinuteOffset#.</p>
<p>Is Daylight Savings Time in effect? #info.isDSTOn#.</p>
</cfoutput>
GetToken

Description
Determines whether a token of the list in the delimiters parameter is present in a string.

Returns
The token found at position index of the string, as a string. If index is greater than the number of tokens in the string, returns an empty string.

Category
String functions

Function syntax
GetToken(string, index [, delimiters ])

See also
Left, Right, Mid, SpanExcluding, SpanIncluding

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>string</td>
<td>A string or a variable that contains one. String in which to search.</td>
</tr>
<tr>
<td>index</td>
<td>Positive integer or a variable that contains one. The position of a token.</td>
</tr>
<tr>
<td>delimiters</td>
<td>A string or a variable that contains one. A delimited list of delimiters. Elements may consist of multiple characters. Default list of delimiters: space character, tab character, newline character; or their codes: &quot;chr(32)&quot;, &quot;chr(9)&quot;, chr(10). Default list delimiter: comma character.</td>
</tr>
</tbody>
</table>

Usage
The following examples show how this function works.

Example A: Consider the following code:
GetToken("red,blue:red,black,tan::red,pink,brown::red,three", 2, ":;")
This function call requests element number 2 from the string, using the delimiter ":;". The output is as follows:
red,black,tan

Example B: Consider the following code:
<cfset mystring = "four," & chr(32)# & chr(9)# & chr(10)# & ",five, nine.zero;;" & chr(10)# & "nine,tens::eleven::twelve::thirteen." & chr(32)# & chr(9)# & chr(10)# & ",four">

642 Chapter 3: ColdFusion Functions
<cfoutput>
#mystring#<br><br></cfoutput>

The output is as follows:

four,
.five, nine.zero::<br>
nine.ten:, eleven::twelve::thirteen,<br>
.four<br>

The `GetToken` function recognizes explicit spaces, tabs, or newline characters as the parameter delimiters (To specify a space character, the code is `chr(32)`; a tab character, `chr(9)`; and a newline character, `chr(10)`.)

In the example string `mystring`, there is:

• A forced space between the substrings "four." and ".five"
• A literal space between ".five." and "nine"
• A literal space between "ten:" and "eleven."
• A forced space between "thirteen." and ",four"

In the following call against `mystring`, no spaces are specified in delimiters (it is omitted), so the function uses the space character as the string delimiter:

<br>
<cfoutput>
GetToken(mystring, 3) is : #GetToken(mystring, 3)#</cfoutput><br>

The output of this code is as follows:

GetToken(mystring, 3) is : nine,zero::<br>

The function finds the third delimiter, and returns the substring just before it that is between the second and third delimiter. This substring is "nine,zero:".

**Example C:** Consider the following code:

<cfset mystring2 = "four,"
   & #chr(9) & #chr(10) & ".five,nine.zero::" & #chr(10) & "nine.ten:,eleven::twelve::thirteen." & #chr(9) & #chr(10) & ".four">
<cfoutput>
#mystring2#<br></cfoutput>

The output is as follows:

four, .five,nine.zero::
nine.ten:,eleven::twelve::thirteen, .four
The following is a call against `mystring2`:

```cfc
<cfoutput>
  GetToken(mystring2, 2) is : #GetToken(mystring2, 2)#
</cfoutput>
```

The output is as follows:

```
GetToken(mystring2, 2) is : ,five,nine,zero;:
```

The function finds the second delimiter, and returns the substring just before it that is between the first and second delimiter. This substring is `"five,nine,zero;"`.

**Example**

```cfm
<h3>GetToken Example</h3>
<cfif IsDefined("FORM.yourString")>
  <!--- set delimiter --->
  <cfif FORM.yourDelimiter is not "">
    <cfset yourDelimiter = FORM.yourDelimiter>
  <cfelse>
    <cfset yourDelimiter = " ">
  </cfif>
  <!--- check whether number of elements in list is greater than or equal to the element sought to return --->
  <cfif ListLen(FORM.yourString, yourDelimiter) GTE FORM.returnElement>
    <cfoutput>
      <p>Element #FORM.ReturnElement# in #FORM.yourString#, delimited by "#yourDelimiter#"
      <br/is:#GetToken(FORM.yourString, FORM.returnElement, yourDelimiter)#
    </cfoutput>
  </cfif>
</cfif>
```

...
Hash

Description
Converts a variable-length string to a fixed-length string that can act as a "fingerprint" or unique identifier for the original string. It is not possible to convert the hash result back to the source string.

Returns
A string.

Category
Conversion functions, Security functions, String functions

Function syntax
Hash(string[, algorithm[, encoding]])

History
ColdFusion MX 7: Added the algorithm and encoding parameters.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>string</td>
<td>String to hash.</td>
</tr>
</tbody>
</table>
| algorithm | (Optional) The algorithm to use to hash the string. ColdFusion MX installs a cryptography library with the following algorithms:  
  • CFMX_COMPAT: Generates a hash string identical to that generated by ColdFusion MX and ColdFusion MX 6.1 (default).  
  • MD5: (Default) Generates a 32-character, hexadecimal string, using the MD5 algorithm (The algorithm used in ColdFusion MX and prior releases).  
  • SHA: Generates a 28-character string using the Secure Hash Standard SHA-1 algorithm specified by Nation Institute of Standards and Technology (NIST) FIPS-180-2.  
  • SHA-256: Generates a 44-character string using the SHA-256 algorithm specified by FIPS-180-2.  
  • SHA-384: Generates a 64-character string using the SHA-384 algorithm specified by FIPS-180-2.  
  • SHA-512: Generates an 88-character string using the SHA-1 algorithm specified by FIPS-180-2.  
  If you install a security provider with additional cryptography algorithms, you can also specify any of its hashing algorithms. |
| encoding  | (Optional; to use this attribute you must also specify the algorithm parameter) A string specifying the encoding to use when converting the string to byte data used by the hash algorithm. Must be a character encoding name recognized by the Java runtime. The default value is the value specified by the defaultCharset entry in the neo-runtime.xml file, which is normally UTF-8, ignored when using the CFMX_COMPAT algorithm. |
Usage

The result of this function is useful for comparison and validation. For example, you can store the hash of a password in a database without exposing the password. You can check the validity of the password by hashing the entered password and comparing the result with the hashed password in the database.

ColdFusion MX 7 uses the Java Cryptography Extension (JCE) and installs a Sun Java 1.4.2 runtime that includes the Sun JCE default security provider. This provider includes the algorithms listed in the Parameters section. The JCE framework includes facilities for using other provider implementations; however, Macromedia cannot provide technical support for third-party security providers.

The encoding attribute is normally not required. It provides a mechanism for generating identical hash values on systems with different default encodings. ColdFusion uses a default encoding of UTF-8 unless you modify the defaultCharset entry in the neo-runtime.xml file.

Example

The following example lets you enter a password and compares the hashed password with a hash value saved in the SecureData table of the cfdocexamples database. This table has the following three entries:

<table>
<thead>
<tr>
<th>User ID</th>
<th>Password</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>abc</td>
</tr>
<tr>
<td>14</td>
<td>def</td>
</tr>
<tr>
<td>15</td>
<td>ghi</td>
</tr>
</tbody>
</table>

<!--- Do the following if the form is submitted. --->
<cfif IsDefined("Form.UserID")>

<!--- query the database. --->
<cfquery name = "CheckPerson" datasource = "cfdocexamples">
SELECT PasswordHash
FROM SecureData
WHERE UserID = <cfqueryparam value = "#Form.userID#" cfsqltype = "CF_SQL_CHARVAR">
</cfquery>

<!--- Compare query PasswordHash field and the hashed form password and display the results. --->
<cfif Hash(Form.password, "SHA") is not checkperson.passwordHash>
User ID #Form.userID# or password is not valid. Try again.
<cfelse>
Password is valid for User ID #Form.userID#.
</cfif>
</cfoutput>
</cfif>
<!--- Form for entering ID and password. --->
<form action="CGI.SCRIPT_NAME" method="post">
  <b>User ID: </b><br>
  <input type = "text" name="UserID" ><br>
  <b>Password: </b><br>
  <input type = "text" name="password" ><br>
  <input type = "Submit" value = "Encrypt my String">
</form>
Hour

**Description**

Gets the current hour of the day.

**Returns**

Ordinal value of the hour, in the range 0 - 23.

**Category**

Date and time functions

**Function syntax**

```
Hour(date)
```

**See also**

DatePart, Minute, Second

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>date</td>
<td>Date/time object, in the range 100 AD–9999 AD.</td>
</tr>
</tbody>
</table>

**Usage**

When passing a date/time value as a string, you must enclose it in quotation marks. Otherwise, it is interpreted as a number representation of a date/time object.

**Example**

```cftmpl
<!--- This example shows the use of Hour, Minute, and Second --->
<h3.Hour Example</h3>
<cfoutput>
The time is currently #TimeFormat(Now())#. We are in hour #Hour(Now())#, Minute #Minute(Now())# and Second #Second(Now())# of the day.
</cfoutput>
```
**HTMLCodeFormat**

**Description**

Replaces special characters in a string with their HTML-escaped equivalents and inserts `<pre>` and `</pre>` tags at the beginning and end of the string.

**Returns**

HTML-escaped string `string`, enclosed in `<pre>` and `</pre>` tags. Return characters are removed; line feed characters are preserved. Characters with special meanings in HTML are converted to HTML character entities such as `&gt;`.

**Category**

Display and formatting functions

**Function syntax**

`HTMLCodeFormat(string [, version])`

**See also**

`HTMLEditFormat`

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>string</td>
<td>A string or a variable that contains one.</td>
</tr>
<tr>
<td>version</td>
<td>HTML version to use; currently ignored.</td>
</tr>
<tr>
<td></td>
<td>- <code>1</code>: The latest implementation of HTML</td>
</tr>
<tr>
<td></td>
<td>- <code>2.0</code>: HTML 2.0 (Default)</td>
</tr>
<tr>
<td></td>
<td>- <code>3.2</code>: HTML 3.2</td>
</tr>
</tbody>
</table>

**Usage**

This function converts the following characters to HTML character entities:

<table>
<thead>
<tr>
<th>Text character</th>
<th>Encoding</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;</code></td>
<td><code>&amp;lt;</code></td>
</tr>
<tr>
<td><code>&gt;</code></td>
<td><code>&amp;gt;</code></td>
</tr>
<tr>
<td><code>&amp;</code></td>
<td><code>&amp;amp;</code></td>
</tr>
<tr>
<td><code>&quot;</code></td>
<td><code>&amp;quot;</code></td>
</tr>
</tbody>
</table>

This function typically increases the length of a string. This can cause unpredictable results when performing certain string functions (Left, Right, and Mid, for example) against the expanded string.

The only difference between this function and HTMLEditFormat is that HTMLEditFormat does not surround the text in an HTML `pre` tag.
Example

<!--- This example shows the effects of HTMLCodeFormat and HTMLEditFormat. View it in your browser, then View it using your browser's the View Source command. --->
<cfset testString="This is a test & this is another

This text is in angle brackets"

Previous line was blank!!!">
<cfoutput>
  <h3>The text without processing</h3>
  #testString#<br>
  <h3>Using HTMLCodeFormat</h3>
  #HTMLCodeFormat(testString)#
  <h3>Using HTMLEditFormat</h3>
  #HTMLEditFormat(testString)#
</cfoutput>
**HTMLEditFormat**

**Description**
Replaces special characters in a string with their HTML-escaped equivalents.

**Returns**
HTML-escaped string `string`. Return characters are removed; line feed characters are preserved. Characters with special meanings in HTML are converted to HTML character entities such as `&gt;`.

**Category**
Display and formatting functions

**Function syntax**
`HTMLEditFormat(string [, version ])`

**See also**
HTMLCodeFormat, cfapplication

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>string</code></td>
<td>A string or a variable that contains one.</td>
</tr>
<tr>
<td><code>version</code></td>
<td>HTML version to use; currently ignored.</td>
</tr>
<tr>
<td></td>
<td>• <code>-1</code>: The latest implementation of HTML</td>
</tr>
<tr>
<td></td>
<td>• <code>2.0</code>: HTML 2.0 (Default)</td>
</tr>
<tr>
<td></td>
<td>• <code>3.2</code>: HTML 3.2</td>
</tr>
</tbody>
</table>

**Usage**
This function converts the following characters to HTML character entities:

<table>
<thead>
<tr>
<th>Text character</th>
<th>Encoding</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;</code></td>
<td><code>&lt;&amp;l t ;</code></td>
</tr>
<tr>
<td><code>&gt;</code></td>
<td><code>&amp;g t ;</code></td>
</tr>
<tr>
<td><code>&amp;</code></td>
<td><code>&amp;a m p ;</code></td>
</tr>
<tr>
<td><code>&quot;</code></td>
<td><code>&amp;quot ;</code></td>
</tr>
</tbody>
</table>

This function can usefed to help protect ColdFusion pages that return user-provided data to the client browser from cross-site scripting attacks. However, the `scriptprotect` attribute of the `cfapplication` tag or the equivalent `This.scriptProtect` variable setting in Application.cfc can be preferable in most instances, because you only need to specify it once for an application.

This function typically increases the length of a string. This can cause unpredictable results when performing certain string functions (`Left, Right, and Mid`, for example) against the expanded string.
The only difference between this function and `HTMLCodeFormat` is that `HTMLCodeFormat` surrounds the text in an HTML `pre` tag.

**Example**

```coldfusion
<cfset testString="This is a test & this is another
Previous line was blank!!!">

<cfoutput>
  <h3>The text without processing</h3>
  #testString#
  <br>
  <h3>Using HTMLCodeFormat</h3>
  #HTMLCodeFormat(testString)#
  <h3>Using HTMLEditFormat</h3>
  #HTMLEditFormat(testString)#
</cfoutput>
```
**IIf**

**Description**

Evaluates a Boolean conditional dynamic expression. Depending on whether the expression is true or false, dynamically evaluates one of two string expressions and returns the result. This function is convenient for incorporating a `cfif` tag in-line in HTML.

For general conditional processing, see `cfif`. For error handling, see `cftry`. For more information, see *ColdFusion MX Developer’s Guide*.

**Returns**

If result is `True`, returns the value of `Evaluate(string_expression1)`; otherwise, returns the value of `Evaluate(string_expression2)`.

**Category**

Decision functions, Dynamic evaluation functions

**Function syntax**

```html
IIf(condition, string_expression1, string_expression2)
```

**See also**

`DE`, `Evaluate`

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>condition</td>
<td>An expression that can be evaluated as a Boolean.</td>
</tr>
<tr>
<td>string_expression1</td>
<td>A string or a variable that contains one. Expression to evaluate and return if condition is True.</td>
</tr>
<tr>
<td>string_expression2</td>
<td>A string or a variable that contains one. Expression to evaluate and return if condition is False.</td>
</tr>
</tbody>
</table>

**Usage**

The `IIf` function is a shortcut for the following construct:

```html
<cfif condition>
  <cfset result = Evaluate(string_expression1)>
<cfelse>
  <cfset result = Evaluate(string_expression2)>
</cfif>
```

The expressions `string_expression1` and `string_expression2` must be string expressions, so that they are not evaluated immediately as the parameters of `IIf`. For example:

```html
IIf(y is 0, DE("Error"), x/y)
```

If `y = 0`, this generates an error, because the third expression is the value of `x/0` (invalid expression).

ColdFusion evaluates `string_expression1` and `string_expression2`. To return the string itself, use the `DE` function.
Note: If you use number signs (#) in string_expression1 or string_expression2, ColdFusion evaluates the part of the expression in number signs first. If you misuse the number signs, you can cause unexpected results from the IIf function. For example, if you use number signs around the whole expression in string_expression1, and if there is an undefined variable in string_expression1, the function might fail, with the error 'Error Resolving Parameter,'

If a variable is undefined, ColdFusion throws an error when it processes this function. The following example shows this problem:

```coldfusion
#IIf(IsDefined("Form.Deliver"), DE(Form.Deliver), DE("no"))#
```

This returns "Error resolving parameter FORM.DELIVER".

To avoid this problem, use the DE and Evaluate functions in code such as the following:

```coldfusion
#IIf(IsDefined("Form.Deliver"), Evaluate(DE("Form.Deliver")), DE("no"))#
```

This returns "no"; ColdFusion does not throw an error.

In the following example, LocalVar is undefined; however, if you omit number signs around LocalVar, the code works properly:

```coldfusion
<cfoutput>
#IIf(IsDefined("LocalVar"), "LocalVar", DE("The variable is not defined."))#
</cfoutput>
```

The output is:

The variable is not defined.

The number signs around LocalVar in the following code cause it to fail with the error message 'Error Resolving Parameter', because ColdFusion never evaluates the original condition IsDefined("LocalVar").

Here is another example:

```coldfusion
<cfoutput>
#IIf(IsDefined("LocalVar"), DE("#LocalVar#"), DE("The variable is not defined."))#
</cfoutput>
```

The error message would be as follows:

Error resolving parameter LOCALVAR

The DE function has no effect on the evaluation of LocalVar, because the number signs cause it to be evaluated immediately.

Example

```coldfusion
<h3>IIf Function Example</h3>
<p>IIf evaluates a condition, and does an Evaluate on string expression 1 or string expression 2 depending on the Boolean outcome <I>(True: run expression 1; False: run expression 2)</I>.
</p>
```
<cfoutput>
    #If( Hour(now()) GTE 12,
    DE("It is afternoon or evening"),
    DE("It is morning"))#
</cfoutput>
</b>
IncrementValue

**Description**

Adds one to an integer.

**Returns**

The integer part of *number*, incremented by one.

**Category**

Mathematical functions

**Function syntax**

`IncrementValue(number)`

**See also**

DecrementValue

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>number</td>
<td>Number to increment</td>
</tr>
</tbody>
</table>

**Example**

```
<h3>IncrementValue Example</h3>
<p>Returns the integer part of a number incremented by one.</p>
<p>IncrementValue(0): <cfoutput>#IncrementValue(0)#</cfoutput></p>
<p>IncrementValue("1"): <cfoutput>#IncrementValue("1")#</cfoutput></p>
<p>IncrementValue(123.35): <cfoutput>#IncrementValue(123.35)#</cfoutput></p>
```
**InputBaseN**

**Description**

Converts *string*, using the base specified by *radix*, to an integer.

**Returns**

A number in the range 2-36, as a string.

**Category**

Mathematical functions

**Function syntax**

\[\text{InputBaseN}(\text{string, radix})\]

**See also**

FormatBaseN

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>string</td>
<td>A string or a variable that contains one. String that represents a number, in the base specified by <em>radix</em>.</td>
</tr>
<tr>
<td>radix</td>
<td>Base of the number represented by <em>string</em>, in the range 2–36.</td>
</tr>
</tbody>
</table>

**Example**

<h3>InputBaseN Example</h3>

<p>InputBaseN converts a number to a string in the base specified by Radix.<br>
<br>FormatBaseN converts a number to a string in the base specified by Radix.<br>
<br>FormatBaseN("1010",2): #FormatBaseN("1010",2)#<br>
<br>FormatBaseN("1024",16): #FormatBaseN("1024",16)#<br>
<br>FormatBaseN("125",10): #FormatBaseN("125",10)#<br>
<br>FormatBaseN("10.75",2): #FormatBaseN("10.75",2)#</p>

<h3>InputBaseN Example</h3>

<p>InputBaseN returns the number obtained by converting a string, using the base specified by Radix.<br>
<br>InputBaseN returns the number obtained by converting a string, using the base specified by Radix.<br>
<br>InputBaseN("1010",2): #InputBaseN("1010",2)#<br>
<br>InputBaseN("3ff",16): #InputBaseN("3ff",16)#<br>
<br>InputBaseN("125",10): #InputBaseN("125",10)#<br>
<br>InputBaseN("1010",2): #InputBaseN("1010",2)#
</p>
Insert

Description
Inserts a substring in a string after a specified character position. If position = 0, prefixes the substring to the string.

Returns
A string.

Category
String functions

Function syntax
Insert(substring, string, position)

See also
RemoveChars, Len

Parameters
Parameter Description
substring A string or a variable that contains one. String to insert.
string A string or a variable that contains one. String into which to insert substring.
position Integer or variable; position in string after which to insert substring.

Example
<h3>Insert Example</h3>
<pre><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#.</p>
</cfoutput>
</cfif>
<cfelse>
<cfoutput>
<p>You inserted the substring #FORM.MySubstring# into the string #FORM.MyString#, resulting in the following string: #Insert(FORM.MySubString, FORM.myString,
FORM.insertposition)#</p>
</cfoutput>
</cfif>
</pre>
Int

Description
Calculates the closest integer that is smaller than number. For example, it returns 3 for Int(3.3) and for Int(3.7); it returns -4 for Int(-3.3) and for Int(-3.7).

Returns
An integer, as a string.

Category
Mathematical functions

Function syntax
Int(number)

See also
Ceiling, Fix, Round

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>number</td>
<td>Real number to round down to an integer.</td>
</tr>
</tbody>
</table>

Example

Int returns the closest integer smaller than a number.

Int(11.7) : <cfoutput>#Int(11.7)#</cfoutput>
Int(-11.7) : <cfoutput>#Int(-11.7)#</cfoutput>
Int(0) : <cfoutput>#Int(0)#</cfoutput>
**IsArray**

**Description**
Determines whether a value is an array.

**Returns**
True, if *value* is an array, or a query column object.

**Category**
Array functions, Decision functions

**Function syntax**
```
IsArray(value [, number ])
```

**See also**
Array functions; "Modifying a ColdFusion XML object" in Chapter 35, “Using XML and WDDX,” in *ColdFusion MX Developer’s Guide*

**History**
ColdFusion MX:
- Changed behavior: if the *value* parameter contains a reference to a query result column, this function now returns True. For example: `IsArray(MyQuery['Column1'])` returns True. (In earlier releases, it returns False.)
- Changed behavior: this function can be used on XML objects.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>Variable or array name</td>
</tr>
<tr>
<td>number</td>
<td>Dimension; function tests whether the array has exactly this dimension</td>
</tr>
</tbody>
</table>

**Usage**
Use this function to determine whether a value is an array or query column. This function evaluates a Java array object, such as a vector object, as having one dimension.

**Example**

```cfdm
<cfset MyNewArray = ArrayNew(1)>
<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>
<p>It has #ArrayLen(MyNewArray)# elements.</p>
<p>Contents: #ArrayToList(MyNewArray)#</p>
</cfoutput>
```
IsAuthenticated

Description

This function is obsolete. Use the newer security tools; see “Conversion functions” on page 453 and Chapter 16, “Securing Applications” in ColdFusion MX Developer’s Guide.

History

ColdFusion MX: This function is obsolete. It does not work in ColdFusion MX and later ColdFusion releases.
IsAuthorized

Description
This function is obsolete. Use the newer security tools; see “Conversion functions” on page 453 and Chapter 16, “Securing Applications” in ColdFusion MX Developer's Guide

History
ColdFusion MX: This function is obsolete. It does not work in ColdFusion MX and later releases.
IsBinary

Description

Determines whether a value is stored as binary data.

Returns

True, if the value is binary; False, otherwise.

Category

Decision functions

Function syntax

IsBinary(value)

See also

ToBinary, ToBase64, IsNumeric, YesNoFormat

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>Number or string</td>
</tr>
</tbody>
</table>

Example

<!--- Create a string of all ASCII characters (32-255) and concatenate them together. --->
<cfset charData =""
<cfloop index="data" from="32" to="255">
  <cfset ch=chr(data)>
  <cfset charData=charData & ch>
</cfloop>

<b>The following string is the concatenation of all characters (32 to 255) from the ASCII table.</b><br><br>
<cfoutput><cfhtmlformat(charData)></cfoutput><br>
<br>
<!--- Create a Base 64 representation of this string. --->
<cfset data64=toBase64(charData)>
<!--- Convert string to binary. --->
<cfset binaryData=toBinary(data64)>
<!--- Check to see if it really is a binary value. --->
<cfif IsBinary(binaryData)>
  The binaryData variable is binary!<br>
</cfif>
<!--- Convert binary data back to Base 64. --->
<cfset another64=toBase64(binaryData)>
<cfif Not IsBinary(another64)>
  The another64 variable is NOT binary!<br>
</cfif>
<!--- Compare another64 with data64 to ensure that they are equal. --->
<cfif another64 eq data64>
  Base 64 representation of binary data is identical to the Base 64 representation of string data.\n\n\n
<cfelse>
  <h3>Conversion error.</h3>
</cfif>
IsBoolean

Description
Determines whether a value can be converted to Boolean

Returns
True, if the parameter can be converted to Boolean; False, otherwise.

Category
Decision functions

Function syntax
IsBoolean(value)

See also
IsNumeric, IsValid, YesNoFormat

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>Number or string</td>
</tr>
</tbody>
</table>

Example

<h3>IsBoolean Example</h3>

<cfif IsDefined("FORM.theTestValue")>
  <cfif IsBoolean(FORM.theTestValue)>
    <h3>The expression <cfoutput>#DE(FORM.theTestValue)#</cfoutput> is Boolean</h3>
  </cfif>
  <cfelse>
    <h3>The expression <cfoutput>#DE(FORM.theTestValue)#</cfoutput> is not Boolean</h3>
  </cfelse>
</cfif>

<form action = "isBoolean.cfm">
  <p>Enter an expression, and discover whether it can be evaluated to a Boolean value.

  <input type = "Text" name = "TheTestValue" value = "1">
  <input type = "Submit" value = "Is it Boolean?" name = "">
</form>
IsCustomFunction

Description
Determines whether a name represents a custom function.

Returns
True, if name can be called as a custom function; False, otherwise.

Category
Decision functions

Function syntax
IsCustomFunction(name)

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Name of a custom function. Must not be in quotation marks. If not a defined variable or function name, ColdFusion generates an error.</td>
</tr>
</tbody>
</table>

Usage
The IsCustomFunction function returns True for any function that can be called as a custom function, including functions defined using CFScript function declarations and cffunction tags, and functions that are ColdFusion component methods. For CFC methods, you must first instantiate the component.

Note: To prevent undefined variable exceptions, always precede IsCustomFunction with an IsDefined test, as shown in the example.

Example
<h3>IsCustomFunction Example</h3>
<cfscript>
function realUDF() {
    return 1;
}
</cfscript>
<cfset X = 1>
<!--- Example that fails existence test --->
<cfif IsDefined("Foo") AND IsCustomFunction(Foo)>
    Foo is a UDF.<br>
</cfif>

<!--- Example that passes existence test but fails IsCustomFunction --->
<cfif IsDefined("X") AND IsCustomFunction(X)>
    X is a UDF.<br>
</cfif>

<!--- Example that passes both tests--->
<cfif IsDefined("realUDF") AND IsCustomFunction(realUDF)>
    realUDF is a function.<br>
</cfif>
<!-- Example using a CFC, defined in TestCFC.cfc-->
<cfobject component="TestCFC" name="myTestCFCObject">
  <CFIF IsDefined("myTestCFCObject.testFunc") AND
    IsCustomFunction(myTestCFCObject.testFunc)>
    myTestCFCObject.testFunc is a function.
  </CFIF>
</cfobject>
**IsDate**

**Description**
Determines whether a string or Java object can be converted to a date/time value.

**Returns**
True, if string can be converted to a date/time value; False, otherwise. ColdFusion converts the Boolean return value to its string equivalent, "Yes" or "No."

**Category**
Date and time functions, Decision functions

**Function syntax**
IsDate(string)

**See also**
CreateDate, IsNumericDate, IsValid, LSDateFormat, LSIsDate, ParseDateTime

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>string</td>
<td>A string or a variable that contains one.</td>
</tr>
</tbody>
</table>

**Usage**
This function checks against U.S. date formats only. For other date support, see LSDateFormat.
A date/time object falls in the range 100 AD–9999 AD.

**Example**
```html
<h3>IsDate Example</h3>
<cfif IsDefined("FORM.theTestValue")>
  <cfif IsDate(FORM.theTestValue)>
    <h3>The string <cfoutput>#DE(FORM.theTestValue)#</cfoutput> is a valid date</h3>
  </cfif>
  <cfelse>
    <h3>The string <cfoutput>#DE(FORM.theTestValue)#</cfoutput> is not a valid date</h3>
  </cfif>
</cfif>

<form action = "isDate.cfm">
  Enter a string, find whether it can be evaluated to a date value.
  <p><input type = "Text" name = "TheTestValue" value = "<cfoutput>#Now()#" />
  <input type = "Submit" value = "Is it a Date?" name = ">
</form>
```
IsDebugMode

Description
Determines whether debugging output is enabled.

Returns
True, if debugging mode is set in the ColdFusion Administrator; False if debugging mode is disabled.

Category
Decision functions

Function syntax
IsDebugMode()

See also
cfsetting

Description
If debugging output is enabled in ColdFusion Administrator and has not been overridden by setting the cfsetting tag showDebugOutput attribute to No, the IsDebugMode function returns Yes; No, otherwise.

Example
<h3>IsDebugMode Example</h3>
<cfif IsDebugMode()>
<h3>Debugging is set in the ColdFusion Administrator</h3>
<cfelse>
<h3>Debugging is disabled</h3>
</cfif>
IsDefined

Description
Evaluates a string value to determine whether the variable named in it exists.
This function is an alternative to the ParameterExists function, which is deprecated.

Returns
True, if the variable is found, or, for a structure, if the specified key is defined; False, otherwise.
The return value is False for an array or structure element referenced using bracket notation. For example, IsDefined("myArray[3]") always returns False, even if the array element myArray[3] has a value.

Category
Decision functions

Function syntax
IsDefined("variable_name")

See also
Evaluate

History
ColdFusion MX: Changed behavior: this function can process only the following constructs:
• A simple variable
• A named variable with dot notation
• A named structure with dot notation (for example: mystruct.key)

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>variable_name</td>
<td>String, enclosed in quotation marks. Name of variable to test for.</td>
</tr>
</tbody>
</table>

Usage
When working with scopes that ColdFusion exposes as structures, the StructKeyExists function can sometimes replace this function. The following lines are equivalent:

```cfc
if(isDefined("form.myVariable"))
if(structKeyExists(form,"myVariable"))
```

Example

```cfc
<cfif IsDefined("form.myString")>
<p>The variable form.myString has been defined, so show its contents. This construction allows us to place a form and its resulting action code on the same page and use IsDefined to control the flow of execution.</p>
<p>The value of "form.myString" is <b><i>#form.myString#</i></b></p>
<cfoutput>#form.myString#</cfoutput></cfif><cfelse>
<cfoutput>&form.myString&lt;/cfoutput></cfif></cfelse>
```
During the first time through this template, the variable "form.myString" has not yet been defined, so we do not try to evaluate it.
</cfif>

<form action="#CGI.Script_Name" method="POST">
<input type="Text" name="MyString" value="My sample value">
<input type="Submit" name="">
</form>
IsK2ServerABroker

Description
This function is deprecated.

Returns
True, if K2Broker is in configured with ColdFusion; False, otherwise.

Category
Decision functions, Full-text search functions, Query functions

Function syntax
IsK2ServerABroker()

See also

History
ColdFusion MX 6.1: Deprecated this function. It might not work, and it might cause an error, in later releases.
ColdFusion MX: Added this function.

Example
<cfoutput>IsK2ServerABroker = $*#IsK2ServerABroker()#*$</cfoutput>
**IsK2ServerDocCountExceeded**

**Description**

This function is deprecated.

**Returns**

True, if the document count limit is exceeded; False, otherwise.

**Category**

Decision functions, Full-text search functions, Query functions

**Function syntax**

`IsK2ServerDocCountExceeded()`

**See also**

`GetK2ServerDocCountLimit, IsK2ServerABroker`

**History**

ColdFusion MX 6.1: Deprecated this function. It might not work, and it might cause an error, in later releases.

ColdFusion 5: Added this function.

**Example**

```
```
IsK2ServerOnline

Description
This function is deprecated because the K2Server is always running when ColdFusion is running.

Returns
True, if the K2Server is available to perform a search; False, otherwise.

Category
Decision functions, Full-text search functions, Query functions

Function syntax
IsK2ServerOnline()

See also
IsK2ServerABroker

History
ColdFusion MX 6.1: Deprecated this function. It might not work, and it might cause an error, in later releases.
ColdFusion MX: Added this function.

Example
<cfoutput>IsK2ServerOnline = $*#IsK2ServerOnline()#*$</cfoutput>
IsLeapYear

Description
Determines whether a year is a leap year.

Returns
True, if year is a leap year; False, otherwise.

Category
Date and time functions, Decision functions

Function syntax
IsLeapYear(year)

See also
DaysInYear

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>year</td>
<td>Number representing a year</td>
</tr>
</tbody>
</table>

Example
<h3>IsLeapYear Example</h3>

<cfif IsDefined("FORM.theTestValue")>
  <cfif IsLeapYear(FORM.theTestValue)>
    <h3>The year value #DE(FORM.theTestValue)# is a Leap Year</h3>
  </cfelseif>
  <cfelse>
    <h3>The year value #DE(FORM.theTestValue)# is not a Leap Year</h3>
  </cfelse>
</cfif>

<form action = "isLeapYear.cfm">
<p>Enter a year value, and find out whether it is a Leap Year.
<p><input type = "Text" name = "TheTestValue" value = 
  "#Year(Now())#">
<input type = "Submit" value = "Is it a Leap Year?" name = ""></form>
IsNumeric

Description

Determines whether a string can be converted to a numeric value. Supports numbers in U.S. number format. For other number support, use LSIsNumeric.

Returns

True, if string can be converted to a number; False, otherwise.

Category

Decision functions

Function syntax

IsNumeric(string)

See also

IsBinary, IsValid

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>string</td>
<td>A string or a variable that contains one.</td>
</tr>
</tbody>
</table>

Example

```html
<h3>IsNumeric Example</h3>
<cfif IsDefined("FORM.theTestValue")>
  <cfif IsNumeric(FORM.theTestValue)>
    <h3>The string #DE(FORM.theTestValue)# can be converted to a number</h3>
  </cfif>
  <cfelse>
    <h3>The string #DE(FORM.theTestValue)# cannot be converted to a number</h3>
  </cfif>
</cfif>

<form action = "isNumeric.cfm">
  Enter a string, and find out whether it can be evaluated to a numeric value.
  <p><input type = "Text" name = "TheTestValue" value = "123">
  <input type = "Submit" value = "Is it a Number?" name = "">
</form>
```
IsNumericDate

Description
Evaluates whether a real number is a valid representation of a date (date/time object).

Returns
True, if the parameter represents a valid date/time object; False, otherwise.

Category
Date and time functions, Decision functions

Function syntax
IsNumericDate(number)

See also
IsDate, ParseDateTime

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>number</td>
<td>A real number</td>
</tr>
</tbody>
</table>

Usage
ColdFusion, by default, evaluates any input parameter and attempts to convert it to a real number before it passes the parameter to the IsNumericDate function. As a result, parameter values such as 12/12/03 and \{ts '2003-01-14 10:04:13'\} return True, because ColdFusion converts valid date string formats to date/time objects, which are real numbers.

Example
<h3>IsNumericDate Example</h3>
<cfif IsDefined("form.theTestValue")>
<!--- test if the value represents a number or a pre-formatted Date value --->
<cfif IsNumeric(form.theTestValue) or IsDate(form.theTestValue)>
<!--- if this value is a numericDate value, then pass --->
<cfif IsNumericDate(form.theTestValue)>
<h3>The string <cfoutput>#DE(form.theTestValue)#</cfoutput> can be converted to a valid numeric date</h3>
<cfelse>
<h3>The string <cfoutput>#DE(form.theTestValue)#</cfoutput> can not be converted to a valid numeric date</h3>
</cfif>
<cfelse>
<h3>The string <cfoutput>#DE(form.theTestValue)#</cfoutput> is not a valid numeric date</h3>
</cfif>
</cfif>

<form action="#cgi.script_name#" method="POST">
<p>Enter a value, and discover if it can be evaluated to a date value.</p>
</form>
<p>
<input type="Text" name="TheTestValue" value="<CFOUTPUT>#Now()#</CFOUTPUT>">
<input type="Submit" value="Is it a Date?" name="">
</form>
IsObject

Description
Determine whether a value is an object.

Returns
True, if the value represents a ColdFusion object. False if the value is any other type of data, such as an integer, string, date, or struct.

Category
Decision functions

Function syntax
IsObject(value)

See also
IsDate, IsNumeric, IsNumericDate, IsQuery, IsSimpleValue, IsStruct, IsWDDX, IsXmlDoc, IsXmlElem, IsXmlRoot

History
ColdFusion MX: Added this function.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>A value, typically the name of a variable.</td>
</tr>
</tbody>
</table>

Usage
This function returns False for query and XML objects.

Example
<!--- to use this example, create a color.cfc component as follows: --->
<!---
<cfcomponent>
  <cffunction name="myFunction" access="public" returntype="string">
    <!--- Create a structure object --->
    <cfset myColor="#Blue#">
    <cfreturn myColor>
  </cffunction>
</cfcomponent>
--->

<!--- Create an instance of the color.cfc component --->
<cfobject name="getColor" component="color">
  <cfif IsObject(getColor)>
    <!--- Invoke the myFunction method --->
    <cfinvoke
      component="#getColor#
      method="myFunction"
    >
  </cfif>
</cfobject>
<cfif IsDefined("myColor")>
  <!--- Output the returned variable --->
  The value of myColor = <cfoutput>#myColor#</cfoutput><p>
</cfif>
</cfif>
IsProtected

Description

This function is obsolete. Use the newer security tools; see “Conversion functions” on page 453 and Chapter 16, “Securing Applications,” in ColdFusion MX Developer's Guide

History

ColdFusion MX: This function is obsolete. It does not work in ColdFusion MX and later releases.
IsQuery

Description
Determines whether value is a query.

Returns
True, if value is a query.

Category
Decision functions, Query functions

Function syntax
IsQuery(value)

See also
QueryAddRow

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>Query variable</td>
</tr>
</tbody>
</table>

Example
<!--- Shows an example of IsQuery and IsSimpleValue -->
<h3>IsQuery Example</h3>
<!--- define a variable called "GetEmployees" -->
<CFPARAM name = "GetEmployees" DEFAULT = "#Now()#"

<p>Before the query is run, the value of GetEmployees is
<cfoutput>#GetEmployees#</cfoutput>
<cfif IsSimpleValue(GetEmployees)>
<p>GetEmployees is currently a simple value
</cfif>
<!--- make a query on the snippets datasource -->
<cfquery name = "GetEmployees" datasource = "cfdocexamples"
SELECT *
FROM employees
</cfquery>

<p>After the query is run, GetEmployees contains a number of rows
that look like this (display limited to three rows):
<cfoutput QUERY = "GetEmployees" MaxRows = "3">
<pre>#Emp_ID# #FirstName# #LastName#</pre>
</cfoutput>
<cfif IsQuery(GetEmployees)>
<p>GetEmployees is no longer a simple value, but the name of a query
</cfif>
**IsSimpleValue**

**Description**
Determine the type of a value.

**Returns**
True, if value is a string, number, Boolean, or date/time value; False, otherwise.

**Category**
Decision functions

**Function syntax**
IsSimpleValue(value)

**See also**
IsValid

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>Variable or expression</td>
</tr>
</tbody>
</table>

**Example**

```cftml
<!--- Shows an example of IsQuery and IsSimpleValue ---->
<h3>IsSimpleValue Example</h3>
<!--- define a variable called "GetEmployees" ---->
<cfparam name = "GetEmployees" default = "#Now()#">  

<p>Before the query is run, the value of GetEmployees is</p>
<cfoutput>#GetEmployees#</cfoutput>
<cfif IsSimpleValue(GetEmployees)>
<p>GetEmployees is currently a simple value</p>
</cfif>

<!--- make a query on the snippets datasource ---->
<cfquery name = "GetEmployees" datasource = "cfdocexamples">
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):</p>
<cfoutput QUERY = "GetEmployees" MaxRows = "3">  
<pre>#Emp_ID# #FirstName# #LastName#</pre>
</cfoutput>
<cfif IsQuery(GetEmployees)>
<p>GetEmployees is no longer a simple value, but the name of a query</p>
</cfif>
```
IsSOAPRequest

Description
Determines whether a CFC is being called as a web service.

Returns
True if CFC is being called as a web service; False, otherwise.

Category
XML functions

History
ColdFusion MX 7: Added this function.

Function syntax
IsSOAPRequest()

See also

Usage
Call this function within a CFC to determine if it is running as a web service.

Example
This example creates a CFC web service that illustrates the operation of the IsSOAPRequest function and also provides a web service that illustrates the operation of other ColdFusion SOAP functions.

Save the following code as headerservice.cfc in a folder called soapheaders under your web root. Test its operation, and specifically the operation of the IsSOAPRequest function, by executing the examples that invoke this web service. For example, see the example for AddSOAPRequestHeader.

<h3>IsSOAPRequest Example</h3>
<cfcomponent displayName="tester" hint="Test for SOAP headers">
<cffunction name="echo_me" access="remote" output="false" returntype="string" displayname="Echo Test" hint="Header test">
<cfargument name="in_here" required="true" type="string">
<cfset isSOAP = isSOAPRequest()>
<cfif isSOAP>
<!--- Get the first header as a string and as XML --->
<cfset username = getSOAPRequestHeader("http://mynamespace/", "username")>
<cfset return = "The service saw username: " & username>
<cfset xmlusername = getSOAPRequestHeader("http://mynamespace/", "username", "TRUE")>
<cfset return = return & "<br> as XML: " & xmlusername>

<!--- Get the second header as a string and as XML --->
<cfset password = getSOAPRequestHeader("http://mynamespace/", "password")>
<cfset return = return & "The service saw password: " & password>
<cfset xmlpassword = getSOAPRequestHeader("http://mynamespace/", "password", "TRUE")>
<cfset return = return & "<br> as XML: " & xmlpassword>

<!--- Add a header as a string --->
<cfset addSOAPResponseHeader("http://www.tomj.org/myns", "returnheader", "AUTHORIZED VALUE", false)>

<!--- Add a second header using a CFML XML value --->
<cfset doc = XmlNew()>
<cfset x = XmlElemNew(doc, "http://www.tomj.org/myns", "returnheader2")>
<cfset x.XmlText = "hey man, here I am in XML">
<cfset x.XmlAttributes[xsi:type] = "xsd:string">
<cfset tmp = addSOAPResponseHeader("ignoredNameSpace", "ignoredName", x)>

<cfelse>
<!--- Add a header as a string - Must generate error! --->
<cfset addSOAPResponseHeader("http://www.tomj.org/myns", "returnheader", "AUTHORIZED VALUE", false)>

<cfreturn return = "Not invoked as a web service">
</cfif>

<cfreturn return>
</cffunction>
</cfcomponent>
IsStruct

Description
Determines whether a variable is a structure.

Returns
True, if variable is a ColdFusion structure or is a Java object that implements the java.lang.Map
interface. The return value is False if the object in variable is a user-defined function (UDF).

Category
Decision functions, Structure functions

Function syntax
IsStruct(variable)

See also
Structure functions; “Modifying a ColdFusion XML object” in Chapter 35, “Using XML and
WDDX,” in ColdFusion MX Developer’s Guide

History
ColdFusion MX: Changed behavior: this function can be used on XML objects.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>variable</td>
<td>Variable name</td>
</tr>
</tbody>
</table>

Example
<!--- This view-only example shows the use of IsStruct. --->
This file is similar to addemployee.cfm, which is called by StructNew,
StructClear, and StructDelete. It is an example of a custom tag used to
add employees. Employee information is passed through the employee
structure (the EMPINFO attribute). In UNIX, you must also add the Emp_ID.
<!---
<cfswitch expression = "#ThisTag.ExecutionMode#">
<cfcase value = "start">
<cfif IsStruct(attributes.EMPINFO)>
<cfoutput>Error. Invalid data.</cfoutput>
<cfelse>
<cfquery name = "AddEmployee" datasource = "cfdocexamples">
INSERT INTO Employees
(FirstName, LastName, Email, Phone, Department)
VALUES
<cfoutput>
'"#StructFind(attributes.EMPINFO, "firstname")#',
'"#StructFind(attributes.EMPINFO, "lastname")#',
'"#StructFind(attributes.EMPINFO, "email")#',
'"#StructFind(attributes.EMPINFO, "phone")#',
'"#StructFind(attributes.EMPINFO, "department")#'
</cfoutput>
</cfif>
<cfoutput><hr>Employee Add Complete</cfoutput>
</cfcase>
</cfswitch> --->
**IsUserInRole**

**Description**
Determines whether an authenticated user belongs to the specified Role.

**Returns**
True, if the authenticated user, belongs to the specified Role; False, otherwise.

**Category**
Security functions, Decision functions

**Function syntax**

```
IsUserInRole("role_name")
```

**See also**
GetAuthUser, cflogin, cfloginuser; Chapter 16, “Securing Applications,” in *ColdFusion MX Developer’s Guide*

**History**
ColdFusion MX: Added this function.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>role_name</td>
<td>Name of a security role</td>
</tr>
</tbody>
</table>

**Usage**
Role names are case-sensitive.

To check if a user is in multiple roles, specify them in a comma-delimited list, such as "Admin,HR". Lists with multiple roles cannot contain spaces as separators; for example, do not use "Admin, HR".

**Example**

```
<cfif IsUserInRole("Admin")>
  <cfoutput>Authenticated user is an administrator</cfoutput>
</cfif>
<cfelse IsUserInRole("User")>
  <cfoutput>Authenticated user is a user</cfoutput>
</cfif>
```
IsValid

Description
Tests whether a value meets a validation or data type rule.

Returns
True, if the value conforms to the rule; False, otherwise.

Category
Decision functions

Function syntax
IsValid(type, value)
isValid("range", value, min, max)
isValid("regex" or "regular_expression", value, pattern)

See also
 cfparam, cfform, IsBoolean, IsDate, IsNumeric, IsSimpleValue; "Validating data with the IsValid function and the cfparam tag" in Chapter 28, “Validating Data,” in ColdFusion MX Developer’s Guide

History
ColdFusion MX 7: Added this function.
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>The valid format for the data; one of the following. For detailed information on validation algorithms, see “Validating form data using hidden fields” in Chapter 28, “Validating Data,” in ColdFusion MX Developer’s Guide.</td>
</tr>
<tr>
<td></td>
<td>• any: any simple value. Returns false for complex values, such as query objects; equivalent to the IsSimpleValue function.</td>
</tr>
<tr>
<td></td>
<td>• array: an ColdFusion array; equivalent to the IsArray function.</td>
</tr>
<tr>
<td></td>
<td>• binary: a binary value; equivalent to the IsBinary function.</td>
</tr>
<tr>
<td></td>
<td>• boolean: a Boolean value: yes, no, true, false, or a number; equivalent to the IsBoolean function.</td>
</tr>
<tr>
<td></td>
<td>• creditcard: a 13-16 digit number conforming to the mod10 algorithm.</td>
</tr>
<tr>
<td></td>
<td>• date or time: any date-time value, including dates or times; equivalent to the IsDate function.</td>
</tr>
<tr>
<td></td>
<td>• email: a valid email address.</td>
</tr>
<tr>
<td></td>
<td>• eurodate: any date-time value, including US date formats and time values,</td>
</tr>
<tr>
<td></td>
<td>• float or numeric: a numeric value; equivalent to the IsNumeric function.</td>
</tr>
<tr>
<td></td>
<td>• guid: a Universally Unique Identifier of the form “xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx” where ‘x’ is a hexadecimal number.</td>
</tr>
<tr>
<td></td>
<td>• integer: an integer.</td>
</tr>
<tr>
<td></td>
<td>• query: a query object; equivalent to the IsQuery function.</td>
</tr>
<tr>
<td></td>
<td>• range: a numeric range, specified by the min and max attributes.</td>
</tr>
<tr>
<td></td>
<td>• regex or regular_expression: matches input against pattern attribute.</td>
</tr>
<tr>
<td></td>
<td>• ssn or social_security_number: A U.S. social security number.</td>
</tr>
<tr>
<td></td>
<td>• string: a string value, including single characters and numbers</td>
</tr>
<tr>
<td></td>
<td>• struct: a structure; equivalent to the IsStruct function.</td>
</tr>
<tr>
<td></td>
<td>• telephone: a standard US telephone number.</td>
</tr>
<tr>
<td></td>
<td>• URL: an http, https, ftp, file, mailto, or news URL.</td>
</tr>
<tr>
<td></td>
<td>• UUID: a ColdFusion Universally Unique Identifier, formatted ’xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx’, where ’x’ is a hexadecimal number. See CreateUUID.</td>
</tr>
<tr>
<td></td>
<td>• USdate: a U.S. date of the format mm/dd/yy, with 1-2 digit days and months, 1-4 digit years.</td>
</tr>
<tr>
<td></td>
<td>• variableName: a string formatted according to ColdFusion variable naming conventions.</td>
</tr>
<tr>
<td></td>
<td>• zipcode: U.S., 5- or 9-digit format ZIP codes.</td>
</tr>
<tr>
<td>value</td>
<td>The value to test</td>
</tr>
<tr>
<td>min</td>
<td>The minimum valid value; used only for range validation</td>
</tr>
<tr>
<td>max</td>
<td>The maximum valid value; used only for range validation</td>
</tr>
<tr>
<td>pattern</td>
<td>A JavaScript regular expression that the parameter must match; used only for regex or regular_expression validation.</td>
</tr>
</tbody>
</table>

Usage

The isValid function lets you assure that validation is performed on the server. You can use the cffparam tag to perform equivalent validation.
Example

The following example checks whether a user has submitted a numeric ID and a valid email address and phone number. If any of the submitted values does not meet the validation test, it displays an error message.

```cfml
<cfif isDefined("form.saveSubmit")>
  <cfif isValid("integer", form.UserID) and isValid("email", form.emailAddr)
    and isValid("telephone", form.phoneNo)>
    <cfoutput>
      <!--- Application code to update the database goes here --->
      <h3>The email address and phone number for user #Form.UserID# have been added</h3>
    </cfoutput>
  </cfif>
<cfelse>
  <h3>You must supply a valid User ID, phone number, and email address.</h3>
</cfif>
<cfform action="#CGI.SCRIPT_NAME#">
  User ID:<cfinput type="Text" name="UserID"><br>
  Phone: <cfinput type="Text" name="phoneNo"><br>
  email: <cfinput type="Text" name="emailAddr"><br>
  <cfinput type="submit" name="saveSubmit" value="Save Data"><br>
</cfform>
```
IsWDDX

Description
Determines whether a value is a well-formed WDDX packet.

Returns
True, if the value is a well-formed WDDX packet; False, otherwise.

Category
Decision functions, XML functions

Syntax
IsWDDX(value)

See also

History
ColdFusion MX: Changed behavior: if the value parameter is not a WDDX packet, ColdFusion returns False. (In earlier releases, ColdFusion threw an error.)

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>A WDDX packet</td>
</tr>
</tbody>
</table>

Usage
This function processes a WDDX packet with a validating XML parser, which uses the WDDX Document Type Definition (DTD).

To prevent CFWDDX deserialization errors, you can use this function to validate WDDX packets from unknown sources.

Example

```xml
<cfset packet=""
    <wddxPacket version='1.0'>
    <header></header>
    <data>
    <struct>
    <var name='ARRAY'>
        <array length='3'>
            <string>one</string>
            <string>two</string>
            <string>three</string>
        </array>
    </var>
    <var name='NUMBER'>
        <string>5</string>
    </var>
    <var name='STRING'>
        <string>hello</string>
    </var>
    </struct>
    </data>
</wddxPacket>"
```
IsWDDX() returns #IsWDDX(packet)#<br>
</cfoutput>
IsXML

Description
Determine whether a string is well-formed XML text.

Returns
True, if the function parameter is a string that contains well-formed XML text; False, otherwise.

Category
Decision functions, XML functions

Function syntax
IsXML(value)

See also
IsXmlAttribute, IsXmlDoc, IsXmlElem, IsXmlNode, IsXmlRoot, XmlParse, XmlValidate;
Chapter 35, "Using XML and WDDX" in ColdFusion MX Developer's Guide

History
ColdFusion MX 7: Added this function.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>A string containing the XML document text</td>
</tr>
</tbody>
</table>

Usage
This function determines whether text is well-formed XML, that is, it conforms to all XML syntax and structuring rules. The string does not have to be a complete XML document. The function does not validate against a Document Type Definition (DTD) or XML Schema.

Example
The following example creates two strings, and tests whether they are well-formed XML text:

```cfml
<!--- A well formed XML string --->
<cfset xmlString1='<order id="4323251">
    <customer firstname="Philip" lastname="Cramer" accountNum="21"/>
    <items>
        <item id="43">
            <quantity>1</quantity>
            <unitprice>15.95</unitprice>
        </item>
    </items>
</order>'

<!--- An invalid XML string, missing the </item> close tag --->
<cfset xmlString2='<order id="4323251">
    <customer firstname="Philip" lastname="Cramer" accountNum="21"/>
    <items>
    </items>
</order>'
```

694 Chapter 3: ColdFusion Functions
<item id="43">
    <quantity>1</quantity>
    <unitprice>15.95</unitprice>
</items>
</order>

<!--- Test the strings to see if they are well formed XML --->
<cfoutput>
xmlString1 contains the following text:<br><br>
#HTMLCodeFormat(xmlstring1)#
Is it well formed XML text? #IsXML(xmlString1)#<br><br>
xmlString2 contains the following text:<br><br>
#HTMLCodeFormat(xmlstring2)#
Is it well formed XML text? #IsXML(xmlString2)#
</cfoutput>
IsXmlAttribute

Description
Determines whether the function parameter is an XML Document Object Model (DOM) attribute node.

Returns
True, if the function argument is an XML attribute node; False, otherwise.

Category
Decision functions, XML functions

Function syntax
IsXmlAttribute(value)

See also
IsXML, IsXmlDoc, IsXmlElem, IsXmlNode, IsXmlNode, IsXmlRoot, XmlGetNodeType, XmlValidate,
Chapter 35, “Using XML and WDDX” in ColdFusion MX Developer’s Guide

History
ColdFusion MX 7: Added this function.

Parameters
Parameter | Description
--- | ---
value | Name of an XML attribute

Usage
This function determines whether the parameter is an XML DOM attribute node, a node with an XMLType value of ATTRIBUTE. It is useful for determining whether a value returned by the XmlSearch function is an XML attribute.

The DOM, and therefore ColdFusion MX, treats XML attributes as properties of an element and does not directly expose them as DOM nodes. For this reason, the XmlNodeAttributes entries in ColdFusion XML document objects do not represent DOM attribute nodes, and tests such as the following always return False:

IsXmlAttribute(xmlNode.XMLAttributes);
IsXmlAttribute(xmlNode.XMLAttributes.attribute);

The XmlSearch function does return attributes as XML DOM attribute nodes. For example, the following line returns an array of attribute nodes containing the quantity attributes in the xmlObject document object:

quantities = XmlSearch(xmlObject, '@quantity');

Example
The following example creates an XML document object and gets parts of it. It then tests whether these parts are attribute nodes.
<!-- Create an XML document object --->
<cfxml variable="xmlobject">
  <order id="4323251">
    <customer firstname="Philip" lastname="Cramer" accountNum="21"/>
    <items>
      <item id="43">
        <quantity>1</quantity>
        <unitprice>15.95</unitprice>
      </item>
    </items>
  </order>
</cfxml>

<!-- Get an array with all lastname quantity DOM attribute nodes
    (In this example there is only one entry) --->
<cfset lastnames = XmlSearch(xmlobject, '//@lastname')>

<!-- Test objects to see if they are attributes --->
<cfoutput>
  <h3>Are the following XML Attribute nodes?</h3>
  <!-- The order element id attribute.
        This a simple variable, not a DOM attribute node.--->
  node.xmlobject.order.XmlAttributes.id:
  IsXmlAttribute(xmlobject.order.XmlAttributes.id)<br>
  <!-- The items element --->
  xmlobject.order.items: IsXmlAttribute(xmlobject.order.items)<br>
  lastnames[1] returned by XmlSearch:
  IsXmlAttribute(lastnames[1])<br>
</cfoutput>
IsXmlDoc

**Description**
Determines whether the function parameter is a ColdFusion XML document object.

**Returns**
True, if the function argument is an XML document object; False, otherwise.

**Category**
Decision functions, XML functions

**Function syntax**
IsXmlDoc(value)

**See also**
IsXML, IsXmlAttribute, IsXmlElem, IsXmlNode, IsXmlRoot, XmlValidate; Chapter 35, “Using XML and WDDX” in ColdFusion MX Developer’s Guide

**History**
ColdFusion MX: Added this function.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>Name of an XML document object</td>
</tr>
</tbody>
</table>

**Example**
The following example creates an XML Document object and a Java object and tests whether they are XML document objects:

<!--- Create an XML document object --->
<cfxml variable="xmlobject">
  <order id="4323251">
    <customer firstname="Philip" lastname="Cramer" accountNum="21"/>
    <items>
      <item id="43">
        <quantity>1</quantity>
        <unitprice>15.95</unitprice>
      </item>
    </items>
  </order>
</cfxml>

<!--- Create a Java object --->
<cfobject type="JAVA" action="create" class="java.lang.Error" name="javaobject"/>

<!--- Test the objects --->
<cfoutput>
Is xmlobject an XML document object? #IsXmlDoc(xmlobject)#<br>
Is javaobject an XML document object? #IsXmlDoc(javaobject)#<br>
</cfoutput>
**IsXmlElem**

**Description**
Determined whether the function parameter is an XML document object element.

**Returns**
True, if the function argument is an XML document object element; False, otherwise.

**Category**
Decision functions, XML functions

**Function syntax**
`IsXmlElem(value)`

**See also**
`IsXML, IsXmlAttribute, IsXmlDoc, IsXmlNode, IsXmlRoot, XmlGetNodeType, XmlValidate`;
Chapter 35, “Using XML and WDDX” in *ColdFusion MX Developer’s Guide*

**History**
ColdFusion MX: Added this function.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>Name of an XML document object element</td>
</tr>
</tbody>
</table>

**Example**
The following example tests whether an XML document object, the document root, and an element are elements:

``` CFML
<!--- Create an XML document object --->
<cfxml variable="xmlobject">
<order id="4323251">
  <customer firstname="Philip" lastname="Cramer" accountNum="21"/>
  <items>
    <item id="43">
      <quantity>1</quantity>
      <unitprice>15.95</unitprice>
    </item>
  </items>
</order>
</cfxml>

<!--- Test parts of the document object to see if they are elements --->
<cfoutput>
  <h3>Are the following XML document object elements?</h3>
  xmlobject: #IsXmlElem(xmlobject)#<br>
  xmlobject.XMLRoot: #IsXmlElem(xmlobject.XMLRoot)#<br>
  xmlobject.order.items: #IsXmlElem(xmlobject.order.items)#
</cfoutput>
```
IsXmlNode

Description
Determines whether the function parameter is an XML document object node.

Returns
True, if the function argument is an XML document object node, including an element; False, otherwise.

Category
Decision functions, XML functions

Function syntax
IsXmlNode(value)

See also
IsXML, IsXmlAttribute, IsXmlDoc, IsXmlElem, IsXmlRoot, XmlGetNodeType, XmlSearch, XmlValidate; Chapter 35, “Using XML and WDDX” in ColdFusion MX Developer’s Guide

History
ColdFusion MX 7: Added this function.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>Name of an XML document object node.</td>
</tr>
</tbody>
</table>

Usage
This function returns True for the following components of an XML document object:
- The document object
- Elements in the object
- XmlNode objects in an element’s XmlNodes array

It also returns True for XML node objects returned by the XmlSearch function. It does not return True for most entries in an element, including XmlText, XmlComment, XmlCdata, or the XmlAttributes array (or individual XML attributes).

Example
The following example tests whether an XML document object, an element, an attribute in the object, and an attribute returned by an XmlSearch function are nodes:

<!--- Create an XML document object --->
<cfxml variable="xmlobject"/>
<?xml version="1.0" encoding="UTF-8"?>
<order id="4323251">
  <customer firstname="Philip" lastname="Cramer" accountNum="21"/>
  <items>
    <item id="43"/>
<quantity>1</quantity>
<unitprice>15.95</unitprice>
</item>
</items>
</order>
</cfxml>

<!--- use XmlSearch to get an attribute node. --->
<cfset lastnames = XmlSearch(xmlobject, '//@lastname')>

<!--- Test the objects to see if they are XML nodes--->
<cfoutput>
<h3>Are the following XML nodes?</h3>
<cfobject>
<cfobject.order.items>
<cfobject.order.XmlAttributes.id>
<cfobject.order.XmlAttributes.id>
</cfoutput>
IsXmlRoot

Description
Determine whether the function parameter is the root element of an XML document object.

Returns
True, if the function argument is the root object of an XML document object; False, otherwise.

Category
Decision functions, XML functions

Function syntax
IsXmlRoot(value)

See also
IsXML, IsXmlAttribute, IsXmlDoc, IsXmlElem, IsXmlNode, XmlGetNodeType, XmlValidate;
Chapter 35, “Using XML and WDDX” in ColdFusion MX Developer’s Guide

History
ColdFusion MX: Added this function.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>Name of an XML document object</td>
</tr>
</tbody>
</table>

Example
The following example tests whether an XML document object, its root element, and a child element are XML root elements:

```plaintext
<!--- Create an XML document object --->
<cfxml variable="xmlobject">
  <?xml version="1.0" encoding="UTF-8"?>
  <order id="4323251">
    <customer firstname="Philip" lastname="Cramer" accountNum="21"/>
    <items>
      <item id="43">
        <quantity>1</quantity>
        <unitprice>15.95</unitprice>
      </item>
    </items>
  </order>
</cfxml>

<!--- Test objects to see if they are XML root elements --->
<cfoutput>
  <h3>Are the following the XML Root?</h3>
  xmlobject: #IsXmlRoot(xmlobject)#
  xmlobject.order: #IsXmlRoot(xmlobject.order)#
  <!--- The order element id attribute --->
  xmlobject.order.XmlAttributes.id: #IsXmlRoot(xmlobject.order.XmlAttributes.id)#
</cfoutput>
```
JavaCast

Description
Converts the data type of a ColdFusion variable to pass as an argument to an overloaded method of a Java object. Use only for scalar and string arguments.

Returns
The variable, as type type.

Category
String functions

Function syntax
JavaCast(type, variable)

See also
CreateObject, cfobject

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| type      | Data type to which to convert variable:  
|           | • boolean  
|           | • int       
|           | • long      
|           | • float     
|           | • double    
|           | • string    
|           | • null      
| variable  | A ColdFusion variable that holds a scalar or string type. Must be ** if type is null. |

Usage
Use after creating a Java object with the cfobject tag, before calling one of its methods. If the method takes more than one overloaded argument, you must call JavaCast for each one. Use JavaCast only when a method is overloaded (because its arguments can take more than one data type, not because the method can take a variable number of arguments).

JavaCast cannot be used to cast between complex objects, nor to cast to a super-class.

Use the result of this function only on calls to Java objects, as shown in the following example:

```<cfscript>
x = CreateObject("java", "test.Hello");
x.init();
ret = x.sayHello(JavaCast("null", ""));
</cfscript>

Warning: Do not assign the results of JavaCast("null", "") to a ColdFusion variable. Unexpected results will occur.
Because there is not a one-to-one correspondence between internally stored ColdFusion types and Java scalar types, some conversions cannot be performed.

Example
The method fooMethod in the class fooClass takes one overloaded argument. The fooClass class is defined as follows:

```java
public class fooClass {
    public fooClass () {
    }
    public String fooMethod(String arg) {
        return "Argument was a String";
    }
    public String fooMethod(int arg) {
        return "Argument was an Integer";
    }
}
```

Within ColdFusion, you use the following code:

```coldfusion
<cfobject
    action="create"
    type = "java"
    class = "fooClass"
    name = obj>
<!--- ColdFusion can treat this as a string or a real number --->
<cfset x = 33>
Perform an explicit cast to an int and call fooMethod:<br>
<cfset myInt = JavaCast("int", x)>
<cfoutput>#obj.fooMethod(myInt)#</cfoutput>
<br><br>
Perform an explicit cast to a string and call fooMethod:<br>
<cfset myString = javaCast("String", x)>
<cfoutput>#obj.fooMethod(myString)#</cfoutput>
```
JSStringFormat

Description

Escapes special JavaScript characters, such as single-quotation mark, double-quotation mark, and newline.

Returns

A string that is safe to use with JavaScript.

Category

String functions

Function syntax

JSStringFormat(string)

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>string</td>
<td>A string or a variable that contains one.</td>
</tr>
</tbody>
</table>

Usage

Escapes special JavaScript characters, so you can put arbitrary strings safely into JavaScript.

Example

<!--- This example shows the use of the JSStringFormat function. ---->
<h3>JSStringFormat</h3>
<cfset stringValue = "An example string value with a tab chr(8),
a newline (chr10) and some ""quoted"" 'text'">
<p>This is the string we have created:<br>
<cfoutput>#stringValue#</cfoutput></p>
<cfset jsStringValue = JSStringFormat(#stringValue#)>
<!--- Generate an alert from the JavaScript string jsStringValue. ---->
<SCRIPT>
s = "<cfoutput>#jsStringValue#</cfoutput>";
alert(s);
</SCRIPT>
**LCase**

**Description**
Converts the alphabetic characters in a string to lowercase.

**Returns**
A string, converted to lowercase.

**Category**
String functions

**Function syntax**
```
LCase(string)
```

**See also**
UCase

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>string</td>
<td>A string or a variable that contains one</td>
</tr>
</tbody>
</table>

**Example**
```
<h3>LCase Example</h3>

<cfif IsDefined("FORM.sampleText")>
  <cfif FORM.sampleText is not ">
    <cfoutput>
      <p>Your text, <b>#FORM.sampleText#</b>, returned in lowercase is <b>#LCase(FORM.sampleText)#</b>.</p>
    </cfoutput>
  </cfif>
<cfelse>
  <p><b><i>Please enter some text.</i></b></p>
</cfif>

<p>Enter your text. Press "submit" to see it returned in lowercase: </p>
<form method="post" action="#cgi.script_name#"
   name="lcaseForm">
  <input type = "Text" name = "SampleText" value = "SAMPLE">
  <input type = "Submit" name = "" value = "submit">
</form>
```
**Left**

**Description**

Returns the leftmost count characters in a string.

**Returns**

String; the first count characters in the string parameter.

**Category**

String functions

**Function syntax**

```
Left(string, count)
```

**See also**

Right, Mid, Len

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>string</td>
<td>A string or a variable that contains one.</td>
</tr>
<tr>
<td>count</td>
<td>A positive integer or a variable that contains one. Number of characters to return.</td>
</tr>
</tbody>
</table>

**Example**

```html
<h3>Left Example</h3>

<cfif IsDefined("Form.myText")>
<!--[if Len Returns 0 (zero), then show error message. -->]
<cfif Len(Form.myText)>
  <cfif Len(Form.myText) LTE Form.RemoveChars>
    <cfoutput><p style="color: red; font-weight: bold">Your string #Form.myText# only has #Len(Form.myText)# characters. You cannot output the #Form.removeChars# leftmost characters of this string because it is not long enough.</p></cfoutput>
  <cfelse>
    <cfoutput><p>Your original string: <strong>#Form.myText#</strong></p>
    <p>Your changed string, showing only the <strong>#Form.removeChars#</strong> leftmost characters: <strong>#Left(Form.myText, Form.removeChars)#</strong></p></cfoutput>
  </cfif>
</cfif>
<cfelse>
  <p style="color: red; font-weight: bold">Please enter a string of more than 0 (zero) characters.</p>
</cfif>

<form action="#CGI.ScriptName#" method="POST">
<p>Type in some text</p>
<input type="Text" name="myText"></input>
<p>How many characters from the left do you want to show?</p>
```
```
<select name="RemoveChars">
<option value="1">1
<option value="3" selected>3
<option value="5">5
<option value="7">7
<option value="9">9</select>
<input type="Submit" name="Submit" value="Remove characters"></p>
</form>
Len

Description
Determines the length of a string or binary object.

Returns
Number; length of a string or a binary object.

Category
String functions

Function syntax
Len(string or binary object)

See also
ToBinary, Left, Right, Mid

History
ColdFusion MX: Changed Unicode support: ColdFusion supports the Java UCS-2 representation of Unicode character values 0–65535. (ColdFusion 5 and earlier releases supported ASCII values 1–255. When calculating a length, some string-processing functions processed the ASCII 0 (NUL) character, but did not process subsequent characters of the string.)

Parameters
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>string</td>
<td>A string, the name of a string, or a binary object</td>
</tr>
</tbody>
</table>

Example

```
<cfif IsDefined("Form.MyText")>
  <cfif Len(FORM.myText)>
    <cfoutput><p>Your string, <strong>#FORM.myText#</strong>, has <strong>#Len(FORM.myText)#</strong> characters.</cfoutput>
  </cfif>
  <cfelse>
    <p style="color: red; font-weight: bold">Please enter a string of more than 0 characters.</p>
  </cfif>
</cfif>

<form action = "#CGI.SCRIPT_NAME#" method="POST">
  <p>Type in some text to see the length of your string.</p>
  <input type = "Text" name = "MyText"><br />
  <input type = "Submit" name="Submit" value = "Count characters"><br>
</form>
```
ListAppend

Description

Concatenates a list or element to a list.

Returns

A copy of the list, with value appended. If delimiter = "", returns a copy of the list, unchanged.

Category

List functions

Function syntax

ListAppend(list, value [, delimiters ])

See also

ListPrepend, ListInsertAt, ListGetAt, ListLast, ListSetAt; “Lists” in Chapter 3, “Using ColdFusion Variables,” in ColdFusion MX Developer’s Guide

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>list</td>
<td>A list or a variable that contains one.</td>
</tr>
<tr>
<td>value</td>
<td>An element or a list of elements.</td>
</tr>
<tr>
<td>delimiters</td>
<td>A string or a variable that contains one. Character(s) that separate list elements. The default value is comma. If this parameter contains more than one character, ColdFusion uses only the first character.</td>
</tr>
</tbody>
</table>

Usage

ColdFusion inserts a delimiter character before value.

The following table shows examples of ListAppend processing:

<table>
<thead>
<tr>
<th>Statement</th>
<th>Output</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>ListAppend('elem1,elem2', '')</td>
<td>elem1,elem2</td>
<td>Appended element is empty; delimiter is last character in list; list length is 2.</td>
</tr>
<tr>
<td>ListAppend('', 'elem1,elem2')</td>
<td>elem1,elem2</td>
<td>List length is 2.</td>
</tr>
<tr>
<td>ListAppend('one___two&quot;, &quot;three&quot;, &quot;___&quot;)</td>
<td>&quot;one___two_three&quot;</td>
<td>Inserted the first character of delimiters before &quot;three.&quot;</td>
</tr>
</tbody>
</table>

Example

<h3>ListAppend Example</h3>

<!--- First, query to get some values for our list elements--->
<cfquery name = "GetParkInfo" datasource = "cfdocexamples">
SELECT PARKNAME,CITY,STATE
FROM PARKS WHERE PARKNAME LIKE ‘AL’
</cfquery>
<cfset temp = ValueList(GetParkInfo.ParkName)>
<cfoutput>
<p>The original list: #temp#</p>
</cfoutput>
<!--- now, append a park name to the list --->
<cfset temp2 = ListAppend(Temp, "ANOTHER PARK")>
...
ListChangeDelims

Description
Changes a list delimiter.

Returns
A copy of the list, with each delimiter character replaced by new_delimiter.

Category
List functions

Function syntax
ListChangeDelims(list, new_delimiter [, delimiters ])

See also
ListFirst, ListQualify; “Lists” in Chapter 3, “Using ColdFusion Variables,” in ColdFusion MX Developer’s Guide

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>list</td>
<td>A list or a variable that contains one.</td>
</tr>
<tr>
<td>new_delimiter</td>
<td>Delimiter string or a variable that contains one. Can be an empty string. ColdFusion processes the string as one delimiter.</td>
</tr>
<tr>
<td>delimiters</td>
<td>A string or a variable that contains one. Character(s) that separate list elements. The default value is comma. If this parameter contains more than one character, ColdFusion processes each occurrence of each character as a delimiter.</td>
</tr>
</tbody>
</table>

Example
<h3>ListChangeDelims Example</h3>
<p>ListChangeDelims lets you change the delimiters of a list.
<!--- First, query to get some values for our list elements--->
<CFQUERY NAME="GetParkInfo" DATASOURCE="cfdocexamples">
SELECT PARKNAME,CITY,STATE
FROM Parks
WHERE PARKNAME LIKE 'BA%'
</CFQUERY>
<CFSET temp = ValueList(GetParkInfo.ParkName)>
<cfoutput>
<p>The original list: <p>#temp#</cfoutput>
<!--- Change the delimiters in the list --->
<CFSET temp2 = ListChangeDelims(Temp, "|:P|", ",")>
<cfoutput>
<p>After executing the statement
  <strong>ListChangeDelims(Temp, "|:P|", ",")</strong>, the updated list: <p>#temp2#</cfoutput>
**ListContains**

**Description**

Determines the index of the first list element that contains a specified substring.

**Returns**

Index of the first list element that contains `substring`. If not found, returns zero.

**Category**

List functions

**Function syntax**

```coldfusion
ListContains(list, substring [, delimiters ])
```

**See also**


**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>list</td>
<td>A list or a variable that contains one.</td>
</tr>
<tr>
<td>substring</td>
<td>A string or a variable that contains one. The search is case-sensitive.</td>
</tr>
<tr>
<td>delimiters</td>
<td>A string or a variable that contains one. Character(s) that separate list elements. The default value is comma. If this parameter contains more than one character, ColdFusion processes each occurrence of each character as a delimiter.</td>
</tr>
</tbody>
</table>

**Usage**

ColdFusion ignores empty list elements; thus, the list "a,b,c,,d" has four elements.

**Example**

```coldfusion
<!--- This example shows differences between ListContains and ListFind --->
<!--- Create a list composed of the elements one, two, three. ---->
<cfset aList = "one">
<cfset aList = ListAppend(aList, "two")>
<cfset aList = ListAppend(aList, "three")>
<p>Here is a list: <cfoutput>#aList#</cfoutput>
<p><strong>ListContains</strong> checks for substring "wo" in the list elements:
<cfoutput><p>&nbsp;&nbsp;&nbsp;Substring "wo" is in element #ListContains(aList, "wo")# of list.</cfoutput>
<p>ListFind cannot check for a substring within an element; therefore, in the code, it does not find substring "wo" (it returns 0):
<cfoutput><p>&nbsp;&nbsp;&nbsp;Substring "wo" is in #ListFind(aList, "wo")# of the list.</cfoutput>
<p>If you specify a string that exactly equals an entire list element, such
as "two", both ListContains and ListFind find it, in the second element:

<p>&nbsp;&nbsp;&nbsp;<strong>ListContains</strong>:<br>
The string "two" is in <b>element #ListContains(aList, "two")#</b> of the list.<br></p>
<p>&nbsp;&nbsp;&nbsp;<strong>ListFind</strong>:<br>The string "two" is in <b>element #ListFind(aList, "two")#</b> of the list.<br></p>
**ListContainsNoCase**

**Description**
Determines the index of the first list element that contains a specified substring.

**Returns**
Index of the first list element that contains *substring*, regardless of case. If not found, returns zero.

**Category**
List functions

**Function syntax**
```
ListContainsNoCase(list, substring [, delimiters ])
```

**See also**

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>list</td>
<td>A list or a variable that contains one.</td>
</tr>
<tr>
<td>substring</td>
<td>A string or a variable that contains one. The search is case-insensitive.</td>
</tr>
<tr>
<td>delimiters</td>
<td>A string or a variable that contains one. Character(s) that separate list elements. The default value is comma. If this parameter contains more than one character, ColdFusion processes each occurrence of each character as a delimiter.</td>
</tr>
</tbody>
</table>

**Usage**
ColdFusion ignores empty list elements; thus, the list "a,b,c,,,d" has four elements.

**Example**
```
<h3>ListContainsNoCase Example</h3>
<cfif IsDefined("form.letter")>
<!--- First, query to get some values for our list --->
<cfquery name="GetParkInfo" datasource="cfdocexamples">
   SELECT PARKNAME,CITY,STATE
   FROM Parks
   WHERE PARKNAME LIKE '#form.letter#%'
</cfquery>
<cfset tempList = #ValueList(GetParkInfo.City)#>
<cfif ListContainsNoCase(tempList, form.yourCity) is not 0>
There are parks in your city!
<cfelse>
<p>Sorry, there were no parks found for your city. Try searching under a different letter.</p>
</cfif>
</cfif>
```

ListContainsNoCase  715
ListDeleteAt

Description

Deletes an element from a list.

Returns

A copy of the list, without the specified element.

Category

List functions

Function syntax

ListDeleteAt(list, position [, delimiters ])

See also


Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>list</td>
<td>A list or a variable that contains one.</td>
</tr>
<tr>
<td>position</td>
<td>A positive integer or a variable that contains one. Position at which to delete element. The first list position is 1.</td>
</tr>
<tr>
<td>delimiters</td>
<td>A string or a variable that contains one. Character(s) that separate list elements. The default value is comma. If this parameter contains more than one character, ColdFusion processes each occurrence of each character as a delimiter.</td>
</tr>
</tbody>
</table>

Usage

To use this and other functions with the default delimiter (comma), you can code as follows:

```cfc...
<cfset temp2 = ListDeleteAt(temp, "3")>
```

To specify another delimiter, you code as follows:

```cfc...
<cfset temp2 = ListDeleteAt(temp, "3", ";")>
```

ColdFusion ignores empty list elements; thus, the list “a,b,c,,d” has four elements.

Example

```cfc...
<!--- First, query to get some values for our list elements--->
<cfquery name="GetParkInfo" datasource="cfdocexamples">
  SELECT PARKNAME, CITY, STATE
  FROM Parks
  WHERE PARKNAME LIKE 'CHI%'
</cfquery>
<cfset temp = ValueList(GetParkInfo.ParkName)>
<cfset deleted_element = ListGetAt(temp, "3", ",")>
<cfoutput>
<p>The original list: #temp#
```


<!--- Delete the third element from the list --->
<CFSET temp2 = ListDeleteAt(Temp, "3")>
<cfoutput>
The changed list: #temp2#
This list element:<br>#deleted_element# is no longer present
at position three of the list.</cfoutput>
ListFind

Description
Determines the index of the first list element in which a specified value occurs. Case-sensitive.

Returns
Index of the first list element that contains value, with matching case. If not found, returns zero. The search is case-sensitive.

Category
List functions

Function syntax
ListFind(list, value [, delimiters ])

See also

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>list</td>
<td>A list or a variable that contains one item.</td>
</tr>
<tr>
<td>value</td>
<td>A string, a number, or a variable that contains one. Item for which to search. The search is case-sensitive.</td>
</tr>
<tr>
<td>delimiters</td>
<td>A string or a variable that contains one. Character(s) that separate list elements. The default value is comma. If this parameter contains more than one character, ColdFusion processes each occurrence of each character as a delimiter.</td>
</tr>
</tbody>
</table>

Usage
ColdFusion ignores empty list elements; thus, the list "a,b,c,,d" has four elements.

Example
<!--- Uses ListFind and ListFindNoCase to see if a substring exists in a list --->
<form action="./listfind.cfm" method="POST">
<p>Try changing the case in Leary's last name:
<br><input type="Text" size="25" name="myString" value="Leary">
<br>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="cfdocexamples">
    SELECT FirstName, RTrim(LastName) AS LName, Phone, Department
    FROM Employees
</cfquery>

<cfset myList = ValueList(SearchEmpLastName.LName)>

<!--- Is this case-sensitive or case-insensitive searching --->
<cfif form.type is "ListFind">
    <cfset temp = ListFind(myList, form.myString)>
    <cfif temp is 0>
        <h3>An employee with that exact last name was not found</h3>
    </cfif>
    <cfelse>
        <cfoutput>
            <p>Employee #ListGetAt(ValueList(SearchEmpLastName.FirstName), temp)# #ListGetAt(ValueList(SearchEmpLastName.LName), temp)#, of the #ListGetAt(ValueList(SearchEmpLastName.Department), temp)# Department, can be reached at #ListGetAt(ValueList(SearchEmpLastName.Phone), temp)#.</p>
            <p>This was the first employee found under this case-sensitive last name search.</p>
        </cfoutput>
    </cfif>
</cfif>

<cfif temp is 0>
    <h3>An employee with that exact last name was not found</h3>
</cfelse>

<cfelse>
    <cfoutput>
        <p>Employee #ListGetAt(ValueList(SearchEmpLastName.FirstName), temp)# #ListGetAt(ValueList(SearchEmpLastName.LName), temp)#, of the #ListGetAt(ValueList(SearchEmpLastName.Department), temp)# Department, can be reached at #ListGetAt(ValueList(SearchEmpLastName.Phone), temp)#.</p>
        <p>This was the first employee found under this case-insensitive last name search.</p>
    </cfoutput>
</cfif>
</cfif>
ListFindNoCase

Description
Determines the index of the first list element in which a specified value occurs.

Returns
Index of the first list element that contains value. If not found, returns zero. The search is case-insensitive.

Category
List functions

Function syntax
ListFindNoCase(list, value [ , delimiters ])

See also

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>list</td>
<td>A list or a variable that contains one.</td>
</tr>
<tr>
<td>value</td>
<td>Number or string for which to search. The search is case-insensitive.</td>
</tr>
<tr>
<td>delimiters</td>
<td>A string or a variable that contains one. Character(s) that separate list elements. The default value is comma. If this parameter contains more than one character, ColdFusion processes each occurrence of each character as a delimiter.</td>
</tr>
</tbody>
</table>

Usage
ColdFusion ignores empty list elements; thus, the list "a,b,c,,d” has four elements.

Example
<!--- Uses ListFind and ListFindNoCase to see if a substring exists in a list --->
<form action="./listfind.cfm" method="POST">
<p>Try changing the case in Leary's last name:
<input type="Text" size="25" name="myString" value="Leary">  
<p>Pick a search type:
<select name="type">
<option value="ListFind" selected>Case-Sensitive
<option value="ListFindNoCase">Case-Insensitive
</select>
<input type="Submit" name="" value="Search Employee List">  
</form>

<!--- wait to have a string for searching defined --->
<cfif IsDefined("form.myString") and IsDefined("form.type")>
<cfquery name="SearchEmpLastName" datasource="cfdocexamples" data source="cfdocexamples">
SELECT FirstName, RTrim(LastName) AS LName, Phone, Department
FROM Employees
</cfquery>

<cfset myList = ValueList(SearchEmpLastName.LName)>
<!--- Is this case-sensitive or case-insensitive searching --->
<cfif form.type is "ListFind">
    <cfset temp = ListFind(myList, form.myString)>
    <cfif temp is 0>
        <h3>An employee with that exact last name was not found</h3>
    </cfif>
    <cfelse>
        <cfoutput>
            <p>Employee #ListGetAt(ValueList(SearchEmpLastName.FirstName), temp)#
                #ListGetAt(ValueList(SearchEmpLastName.LName), temp)#, of the
                #ListGetAt(ValueList(SearchEmpLastName.Department), temp)# Department,
                can be reached at #ListGetAt(ValueList(SearchEmpLastName.Phone),
                temp)#.
            </p>
            <p>This was the first employee found under this case-sensitive last name search.</p>
        </cfoutput>
    </cfif>
</cfif>
<cfelse>
    <cfset temp = ListFindNoCase(myList, form.myString)>
    <cfif temp is 0>
        <h3>An employee with that exact last name was not found</h3>
    </cfif>
    <cfelse>
        <cfoutput>
            <p>Employee #ListGetAt(ValueList(SearchEmpLastName.FirstName), temp)#
                #ListGetAt(ValueList(SearchEmpLastName.LName), temp)#, of the
                #ListGetAt(ValueList(SearchEmpLastName.Department), temp)# Department,
                can be reached at #ListGetAt(ValueList(SearchEmpLastName.Phone),
                temp)#.
            </p>
            <p>This was the first employee found under this case-insensitive last name search.</p>
        </cfoutput>
    </cfif>
</cfif>
</cfif>
**ListFirst**

**Description**

 Gets the first element of a list.

**Returns**

 The first element of a list. If the list is empty, returns an empty string.

**Category**

 List functions

**Function syntax**

 `ListFirst(list [, delimiters ])`

**See also**

 `ListGetAt, ListLast, ListQualify`; “Lists” in Chapter 3, “Using ColdFusion Variables,” in ColdFusion MX Developer’s Guide

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>list</td>
<td>A list or a variable that contains a list.</td>
</tr>
<tr>
<td>delimiters</td>
<td>A string or a variable that contains one. Character(s) that separate list elements. The default value is comma. If this parameter contains more than one character, ColdFusion processes each occurrence of each character as a delimiter.</td>
</tr>
</tbody>
</table>

**Usage**

 ColdFusion ignores empty list elements; thus, the list "a,b,c,,d" has four elements.

**Example**

```coldfusion
<h3>ListFirst Example</h3>
<!--- Find a list of users who wrote messages --->
<cfquery name = "GetMessageUser" datasource = "cfdocexamples">
    SELECT Username, Subject, Posted 
    FROM Messages
</cfquery>

<cfset temp = ValueList(GetMessageUser.Username)>
<p>Before editing the list, it is:&nbsp;</p><cfoutput>#ValueList(GetMessageUser.Username)#</cfoutput>. <p>(Users who posted more than once are listed more than once.)</p> <!--- Show the first user in the list --->
<p>The first user in the list is: #ListFirst(temp)#</p> <p>The rest of the list is:&nbsp;</p><cfoutput>#ListRest(temp)#</cfoutput>. <p>(Users who posted more than once are listed more than once.)</p> <p>The last user in the list is: #ListLast(temp)#</p>
```
ListGetAt

Description

Gets a list element at a specified position.

Returns

Index of the list element at position position.

Category

List functions

Function syntax

ListGetAt(list, position [, delimiters ])

See also

ListFirst, ListLast, ListQualify, ListSetAt; “Lists” in Chapter 3, “Using ColdFusion Variables,” in ColdFusion MX Developer’s Guide

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>list</td>
<td>A list or a variable that contains one.</td>
</tr>
<tr>
<td>position</td>
<td>A positive integer or a variable that contains one. Position at which to get element. The first list position is 1.</td>
</tr>
<tr>
<td>delimiters</td>
<td>A string or a variable that contains one. Character(s) that separate list elements. The default value is comma. If this parameter contains more than one character, ColdFusion processes each occurrence of each character as a delimiter.</td>
</tr>
</tbody>
</table>

Usage

If you use list functions on strings that are delimited by a delimiter character and a space, a returned list element might contain a leading space; you use the trim function to remove such spaces from a returned element. For example, consider this list:
<cfset myList = "one hundred, two hundred, three hundred">

To get a value from this list, use the trim function to remove the space before the returned value:
<cfset MyValue = #trim(listGetAt(myList, 2))#>

With this usage, MyValue = "two hundred", not "two hundred", and spaces within a list element are preserved.

ColdFusion ignores empty list elements; thus, the list "a,b,c,,d" has four elements.

Example

<h3>ListGetAt Example</h3>

<cfquery name = "GetMessageUser" datasource = "cfdocexamples">
   SELECT Username, Subject, Posted
   FROM Messages
<cfset temp = ValueList(GetMessageUser.Username)>
<!--- loop through the list and show it with ListGetAt --->
<h3>This list of usernames who have posted messages numbers</h3>
<cfoutput>#ListLen(temp)#</cfoutput> users.</h3>
<ul>
<cfloop From = "1" To = "#ListLen(temp)#" index = "Counter">
<cfoutput><li>Username #Counter#: #ListGetAt(temp, Counter)#</cfoutput>
</cfloop>
</ul>
**ListInsertAt**

**Description**
Inserts an element in a list.

**Returns**
A copy of the list, with *value* inserted at the specified position.

**Category**
List functions

**Function syntax**
```coldfusion
ListInsertAt(list, position, value [,. delimiters ])
```

**See also**

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>list</td>
<td>A list or a variable that contains one.</td>
</tr>
<tr>
<td>position</td>
<td>A positive integer or a variable that contains one. Position at which to insert element. The first list position is 1.</td>
</tr>
<tr>
<td>value</td>
<td>An element or a list of elements.</td>
</tr>
<tr>
<td>delimiters</td>
<td>A string or a variable that contains one. Character(s) that separate list elements. The default value is comma. If this parameter contains more than one character, ColdFusion processes each occurrence of each character as a delimiter.</td>
</tr>
</tbody>
</table>

**Usage**
When inserting an element, ColdFusion inserts a delimiter. If *delimiters* contains more than one delimiter, ColdFusion uses the first delimiter in the string; if *delimiters* is omitted, ColdFusion uses a comma.

ColdFusion ignores empty list elements; thus, the list "a,b,c,,,d" has four elements.

**Example**
```coldfusion
<cfquery name = "GetParkInfo" datasource = "cfdocexamples">
SELECT PARKNAME,CITY,STATE 
FROM PARKS 
WHERE PARKNAME LIKE 'DE%'
</cfquery>
<cfset temp = ValueList(GetParkInfo.ParkName)>
<cfset insert_at_this_element = ListGetAt(temp, "3", ",")>
<cfoutput>
<p>The original list: #temp#
</cfoutput>
<cfset temp2 = ListInsertAt(Temp, "3", "my Inserted Value")>
```

---

ListInsertAt 725
ListLast

Description

Gets the last element of a list.

Returns

The last element of the list.

Category

List functions

Function syntax

ListLast(list [, delimiters ])

See also


Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>list</td>
<td>A list or a variable that contains a list.</td>
</tr>
<tr>
<td>delimiters</td>
<td>A string or a variable that contains one. Character(s) that separate list elements. The default value is comma. If this parameter contains more than one character, ColdFusion processes each occurrence of each character as a delimiter; you cannot specify a multicharacter delimiter.</td>
</tr>
</tbody>
</table>

Usage

If you use list functions on strings that separated by a delimiter character and a space, a returned list element might contain a leading space; use the trim function to remove leading and trailing spaces from a returned element. For example, consider this list:

```cfset myList = "one hundred, two hundred, three hundred"
```

To get a value from this list, use the trim function to remove the space before the returned value:

```cfset MyValue = #trim(ListLast(myList)#>
```

With this usage, the MyValue variable gets the value "three hundred", not " three hundred", and spaces within a list element are preserved.

ColdFusion ignores empty list elements; thus, the list "a,b,c,d" has four elements.

Example

```<h3>ListFirst, ListLast, and ListRest Example</h3>
<cfset temp = ValueList(GetMessageUser.Username)>
```
Before editing the list, it is: 
<cfoutput>$ValueList(GetMessageUser.Username)$</cfoutput>. (Users who posted more than once are listed more than once.)

--- Show the first user in the list ---
The first user in the list is: <cfoutput>$ListFirst(temp)$</cfoutput>
The rest of the list is: 
<cfoutput>$ListRest(temp)$</cfoutput>. (Users who posted more than once are listed more than once.)
The last user in the list is: <cfoutput>$ListLast(temp)$</cfoutput>
ListLen

Description

Determines the number of elements in a list.

Integer; the number of elements in a list.

Category

List functions

Function syntax

ListLen(list [, delimiters ])

See also


Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>list</td>
<td>A list or a variable that contains one.</td>
</tr>
<tr>
<td>delimiters</td>
<td>A string or a variable that contains one. Character(s) that separate list elements. The default value is comma. If this parameter contains more than one character, ColdFusion processes each occurrence of each character as a delimiter.</td>
</tr>
</tbody>
</table>

Usage

ColdFusion ignores empty list elements; thus, the list "a,b,c,,,d" has four elements.

Here are some examples of ListLen processing:

<table>
<thead>
<tr>
<th>Statement</th>
<th>Output</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>ListLen('a,b,c,,,d')</td>
<td>4</td>
<td>Third element is &quot;c&quot;</td>
</tr>
<tr>
<td>ListLen('a,b,c,,,d',',')</td>
<td>4</td>
<td>Fourth element is &quot;d&quot;</td>
</tr>
<tr>
<td>ListLen('elem_1___elem_2___elem_3')</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>ListLen('elem<em>1</em><strong>elem<em>2</em></strong>elem*3')</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>ListLen('elem_1___elem_2___elem_3', '_')</td>
<td>6</td>
<td></td>
</tr>
</tbody>
</table>

Example

<!--- ListLen Example--->
<cfquery name = "GetMessageUser" datasource = "cfdocexamples">
  SELECT Username, Subject, Posted
  FROM Messages
</cfquery>
<cfset temp = ValueList(GetMessageUser.Username)>
<!--- Loop through the list and show it with ListGetAt --->
<h3>This is a list of usernames who have posted messages

728  Chapter 3: ColdFusion Functions
<cfoutput>#{ListLen(temp)}#</cfoutput> users.</h3>
<ul>
<cfloop From = "1" To = "#{ListLen(temp)}#" INDEX = "Counter">
  <cfoutput><li>Username #{Counter}:
      #{ListGetAt(temp, Counter)}#</cfoutput>
</cfloop>
</ul>
ListPrepend

Description

Inserts an element at the beginning of a list.

Returns

A copy of the list, with \texttt{value} inserted at the first position.

Category

List functions

Function syntax

\texttt{ListPrepend(list, value [, delimiters])}

See also

ListAppend, ListInsertAt, ListSetAt; "Lists" in Chapter 3, “Using ColdFusion Variables,” in ColdFusion MX Developer's Guide

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>list</td>
<td>A list or a variable that contains one.</td>
</tr>
<tr>
<td>value</td>
<td>An element or a list of elements.</td>
</tr>
<tr>
<td>delimiters</td>
<td>A string or a variable that contains one. Character(s) that separate list elements. The default value is comma. If this parameter contains more than one character, ColdFusion only uses the first character and ignores the others.</td>
</tr>
</tbody>
</table>

Usage

When prepending an element to a list, ColdFusion inserts a delimiter. If \texttt{delimiters} contains more than one delimiter character, ColdFusion uses the first delimiter in the string; if \texttt{delimiters} is omitted, ColdFusion uses a comma.

ColdFusion ignores empty list elements; thus, the list "a,b,c,,d" has four elements.

If the \texttt{delimiters} parameter is the empty string (""), ColdFusion returns the contents of the \texttt{value} parameter.

Example

\begin{verbatim}
<!--- This example shows ListPrepend --->
<cfquery name = "GetParkInfo" datasource = "cfdocexamples">
  SELECT PARKNAME,CITY,STATE
  FROM PARKS
  WHERE PARKNAME LIKE 'DE%'
</cfquery>
<cfset temp = ValueList(GetParkInfo.ParkName)>
<cfset first_element = ListFirst(temp)>
<cfoutput><p>The original list: #temp#</cfoutput>
<!--- now, insert an element at position 1--->
<cfset temp2 = ListPrepend(Temp, "my Inserted Value")>
\end{verbatim}
ListQualify

Description

Inserts a string at the beginning and end of list elements.

Returns

A copy of the list, with qualifier before and after the specified element(s).

Category

List functions

Function syntax

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

See also


History

ColdFusion MX: Changed behavior: as the elements parameter value, you must specify "all" or "char"; otherwise, ColdFusion throws an exception. (In earlier releases, the function ignored an invalid value, and used "all"; this was inconsistent with other functions.)

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>list</td>
<td>A list or a variable that contains one.</td>
</tr>
<tr>
<td>qualifier</td>
<td>A string or a variable that contains one. Character or string to insert before and after the list elements specified in the elements parameter.</td>
</tr>
<tr>
<td>delimiters</td>
<td>A string or a variable that contains one. Character(s) that separate list elements. The default value is comma. If this parameter contains more than one character, ColdFusion uses the first character as the delimiter and ignores the remaining characters.</td>
</tr>
</tbody>
</table>
| elements | • all: all elements  
• char: elements that are composed of alphabetic characters |

Usage

The new list might not preserve all of the delimiters in the list.

ColdFusion ignores empty list elements; thus, the list "a,b,c,,d" has four elements.

Example

```<cfquery name = "GetEmployeeNames" datasource = "cfdocexamples">  
SELECT FirstName, LastName  
FROM Employees  
</cfquery>  

<h3>ListQualify Example</h3>  
<p>This example uses ListQualify to put the full names of the employees in the query within quotation marks.</p>```
<cfset myArray = ArrayNew(1)>

<!--- loop through query; append these names successively to the last element --->
<cfloop query = "GetEmployeeNames">
   <cfset temp = ArrayAppend(myArray, "#FirstName# #LastName#")></cfloop>

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

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

<cfoutput>
<p>The contents of the unqualified list are as follows:</p>
#myList#
</cfoutput>

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

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

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

<!--- output the array as a list --->
<cfoutput>
<p>The contents of the second qualified list are:</p>
<p>#qualifiedList2#</p>
</cfoutput>
ListRest

Description

Gets a list, without its first element.

Returns

A copy of list, without the first element. If list has one element, returns an empty list.

Category

List functions

Function syntax

ListRest(list [, delimiters ])

See also


Parameters

Parameter | Description
---|---
list | A list or a variable that contains one.
delimiters | A string or a variable that contains one. Character(s) that separate list elements. The default value is comma. If this parameter contains more than one character, ColdFusion processes each occurrence of each character as a delimiter.

Usage

If the list begins with one or more empty entries, this function drops them, as well as the first element.

ColdFusion ignores empty list elements; thus, the list “a,b,c,,d” has four elements.

Example

<h3>ListFirst, ListLast, and ListRest Example</h3>
<!--- Find a list of users who wrote messages --->
<cfquery name = "GetMessageUser" datasource = "cfdocexamples">
  SELECT Username, Subject, Posted
  FROM  Messages
</cfquery>
<cfset temp = ValueList(GetMessageUser.Username)>
<p>Before editing the list, it is: &nbsp;</p>
<cfoutput>&lt;cfset temp = ValueList(GetMessageUser.Username)&gt;</cfoutput>.<p>(Users who posted more than once are listed more than once.)</p>
<cfset FirstUser = ListFirst(temp)>
<p>The first user in the list is: &nbsp;</p>
<cfoutput>&lt;cfoutput>#ListFirst(temp)#</cfoutput>.</p>(Users who posted more than once are listed more than once.)</p>
<cfset RestOfList = ListRest(temp)>
<p>The rest of the list is: &nbsp;</p>
<cfoutput>&lt;cfoutput>#ListRest(temp)#</cfoutput>.</p>(Users who posted more than once are listed more than once.)</p>
<cfset LastUser = ListLast(temp)>
<p>The last user in the list is: &nbsp;</p>
<cfoutput>&lt;cfoutput>#ListLast(temp)#</cfoutput>
**ListSetAt**

**Description**
Replaces the contents of a list element.

**Returns**
A copy of a list, with a new value assigned to the element at a specified position.

**Category**
List functions

**Function syntax**
```
ListSetAt(list, position, value [, delimiters ])
```

**See also**
ListDeleteAt, ListGetAt, ListInsertAt; “Lists” in Chapter 3, “Using ColdFusion Variables,” in ColdFusion MX Developer's Guide

**History**
ColdFusion MX: Changed delimiter modification: ColdFusion MX does not modify delimiters in the list. (In earlier releases, in some cases, replaced delimiters with the first character in the delimiters parameter.)

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>list</td>
<td>A list or a variable that contains one.</td>
</tr>
<tr>
<td>position</td>
<td>A positive integer or a variable that contains one. Position at which to set a value. The first list position is 1.</td>
</tr>
<tr>
<td>value</td>
<td>An element or a list of elements.</td>
</tr>
<tr>
<td>delimiters</td>
<td>A string or a variable that contains one. Character(s) that separate list elements. The default value is comma. If this parameter contains more than one character, ColdFusion processes each occurrence of each character as a delimiter.</td>
</tr>
</tbody>
</table>

**Usage**
When assigning an element to a list, ColdFusion inserts a delimiter. If delimiters contains more than one delimiter, ColdFusion uses the first delimiter in the string, or, if delimiters was omitted, a comma.

ColdFusion ignores empty list elements; thus, the list “a,b,c,,d” has four elements.

**Example**
```
<h3>ListSetAt Example</h3>
<!---- Find a list of users who wrote messages ---->
<cfquery name = "GetMessageUser" datasource = "cfdocexamples">
SELECT Username, Subject, Posted
FROM Messages
```
</cfquery>

<cfset temp = ValueList(GetMessageUser.Subject)>

<!--- loop through the list and show it with ListGetAt --->
<h3>This is a list of <cfoutput>#ListLen(temp)#</cfoutput> subjects posted in messages.</h3>

<cfset ChangedElement = ListGetAt(temp, 2)>
<cfset TempToo = ListSetAt(temp, 2, "I changed this subject", ",")>
<ul>
<cfloop From = "1" To = "#ListLen(temptoo)#" INDEX = "Counter">
  <cfoutput><li>(#Counter#) SUBJECT: #ListGetAt(temptoo, Counter)#</cfoutput>
</cfloop></ul>
<p>Note that element 2, "<cfoutput>#changedElement#</cfoutput>", has been altered to "I changed this subject" using ListSetAt.
ListSort

Description
Sorts list elements according to a sort type and sort order.

Returns
A copy of a list, sorted.

Category
List functions

Function syntax
ListSort(list, sort_type [, sort_order] [, delimiters ])

See also

History
ColdFusion MX: Changed the order in which sorted elements are returned: in a textnocase, descending sort, this function might return elements in a different sort order than in earlier releases. If sort_type = "textnocase" and sort_order = "desc", ColdFusion MX processes elements that differ only in case differently from earlier releases. ColdFusion MX outputs the elements in the reverse of the ascending order. Earlier releases do not change order of elements that differ only in case. Both operations are correct. The new operation ensures that an ascending and descending sort output elements in exactly reverse order.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>list</td>
<td>A list or a variable that contains one.</td>
</tr>
<tr>
<td>sort_type</td>
<td>• numeric: sorts numbers</td>
</tr>
<tr>
<td></td>
<td>• text: sorts text alphabetically, taking case into account (also known as case sensitive). All letters of one case precede the first letter of the other case:</td>
</tr>
<tr>
<td></td>
<td>- aabzABZ, if sort_order = &quot;asc&quot; (ascending sort)</td>
</tr>
<tr>
<td></td>
<td>- ZBAzbaa, if sort_order = &quot;desc&quot; (descending sort)</td>
</tr>
<tr>
<td></td>
<td>• textnocase: sorts text alphabetically, without regard to case (also known as case-insensitive). A letter in varying cases precedes the next letter:</td>
</tr>
<tr>
<td></td>
<td>- aAaBbBzzZ, in an ascending sort; preserves original intra-letter order</td>
</tr>
<tr>
<td></td>
<td>- ZzzBbBaAa, in a descending sort; reverses original intra-letter order</td>
</tr>
</tbody>
</table>

For example, in a textnocase, desc sort of d,a,a,b,A, the following occurs:
• ColdFusion MX returns d,b,A,a,a
• Earlier ColdFusion releases return d,b,a,a,A
(In a textnocase, asc sort, all ColdFusion releases return a,a,A,b,d.)
ColdFusion ignores empty list elements; thus, the list “a,b,c,,,d” has four elements.

**Example**

```coldfusion
<h3>ListSort Example</h3>

<!--- Find a list of users who wrote messages --->
<cfquery name = "GetMessageUser" datasource = "cfdocexamples">
SELECT   Username, Subject, Posted
FROM     Messages
</cfquery>

<cfset myList = ValueList(GetMessageUser.UserName)>
<p>Here is the unsorted list. </p>
<cfoutput>#myList#</cfoutput>

<p>Here is the list sorted alphabetically:</p>
<cfset sortedList = ListSort(myList, "Text")>
<cfoutput>#sortedList#</cfoutput>

<p>Here is a numeric list that is to be sorted in descending order.</p>
<cfset sortedNums = ListSort("12,23,107,19,1.65","Numeric", "Desc")>
<cfoutput>#sortedNums#</cfoutput>

<p>Here is a list that must be sorted numerically, since it contains negative and positive numbers, and decimal numbers.</p>
<cfset sortedNums2 = ListSort("23.75:-34.471:100:-9745","Numeric", "ASC", "::")>
<cfoutput>#sortedNums2#</cfoutput>

<p>Here is a list to be sorted alphabetically without consideration of case.</p>
<cfset sortedMix = ListSort("hello;123,HELLO:jeans,-345,887;ColdFusion:coldfusion", ":textNoCase": "ASC", "::")>
<cfoutput>#sortedMix#</cfoutput>
```
ListToArray

**Description**
Copies the elements of a list to an array.

**Returns**
An array

**Category**
Array functions, Conversion functions, List functions

**Function syntax**
\[
\text{ListToArray}(\text{list} [, \text{delimiters} ])
\]

**See also**
ArrayToList; Chapter 5, “Using Arrays and Structures,” in *ColdFusion MX Developer’s Guide*

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>list</td>
<td>A list or a variable that contains one. You define a list variable with a cfset statement.</td>
</tr>
<tr>
<td>delimiters</td>
<td>A string or a variable that contains one. ColdFusion treats each character in the string as a delimiter. The default value is comma.</td>
</tr>
</tbody>
</table>

**Usage**
ColdFusion ignores empty list elements; thus, the list ”a,b,c,,,d” has four elements.

ColdFusion treats each character in the delimiters parameter as a separate delimiter. Therefore, if the parameter is ”,+” ColdFusion will break the list at either a comma or a plus sign.

**Example**

```coldfusion
<h3>ListToArray Example</h3>
<!---- Find a list of users who wrote messages --->
<cfquery name = "GetMessageUser" datasource = "cfdocexamples">
SELECT Username, Subject, Posted
FROM Messages
</cfquery>
<cfset myList = ValueList(GetMessageUser.UserName)>
<p>My list is a list with <cfoutput>#ListLen(myList)#</cfoutput> elements.
<cfset myArrayList = ListToArray(myList)>
<p>My array list is an array with <cfoutput>#ArrayLen(myArrayList)#</cfoutput> elements.
</cfoutput> elements.
```
**ListValueCount**

**Description**

Counts instances of a specified value in a list. The search is case-sensitive.

**Returns**

The number of instances of *value* in the list.

**Category**

List functions, String functions

**Function syntax**

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

**See also**


**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>list</td>
<td>A list or a variable that contains one.</td>
</tr>
<tr>
<td>value</td>
<td>String or number, or a variable that contains one. Item for which to search. The search is case-sensitive.</td>
</tr>
<tr>
<td>delimiters</td>
<td>A string or a variable that contains one. Character(s) that separate list elements. The default value is comma. If this parameter contains more than one character, ColdFusion processes each occurrence of each character as a delimiter.</td>
</tr>
</tbody>
</table>

**Example**

```cfquery name = "SearchByDepartment" datasource = "cfdocexamples">
SELECT Department
FROM Employees
</cfquery>
<h3>ListValueCount Example</h3>
<p>This example uses ListValueCount to count employees in a department.</p>
<form action = "listvaluecount.cfm">
<p>Select a department:</p>
<select name = "departmentName">
<option value = "Accounting">Accounting</option>
<option value = "Administration">Administration</option>
<option value = "Engineering">Engineering</option>
<option value = "Sales">Sales</option>
</select>
</form>
sales
</option>
</select>
<input type = "Submit" name = "Submit" value = "Search Employee List">
</form>

<!-- wait to have a string for searching defined -->
<cfif IsDefined("FORM.Submit") and IsDefined("FORM.departmentName")>
<cfset myList = ValueList(SearchByDepartment.Department)>
<cfset numberInDepartment = ListValueCount(myList, FORM.departmentName)>

<cfif numberInDepartment is 0>
  <h3>There are no employees in &lt;cfoutput&gt;#FORM.departmentName#&lt;/cfoutput&gt;&lt;/h3&gt;
<cfelseif numberInDepartment is 1>
  <cfoutput&gt;&lt;p&gt;There is only one person in #FORM.departmentName#. &lt;/p&gt;&lt;/cfoutput&gt;
<cfelse>
  <cfoutput&gt;&lt;p&gt;There are #numberInDepartment# people in #FORM.departmentName#. &lt;/p&gt;&lt;/cfoutput&gt;
</cfif>
</cfif>
**ListValueCountNoCase**

**Description**

Counts instances of a specified value in a list. The search is case-insensitive.

**Returns**

The number of instances of *value* in the list.

**Category**

List functions

**Function syntax**

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

**See also**


**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>list</td>
<td>A list or a variable that contains one.</td>
</tr>
<tr>
<td>value</td>
<td>String or number, or a variable that contains one. Item for which to search. The search is case-insensitive.</td>
</tr>
<tr>
<td>delimiters</td>
<td>A string or a variable that contains one. Character(s) that separate list elements. The default value is comma. If this parameter contains more than one character, ColdFusion processes each occurrence of each character as a delimiter.</td>
</tr>
</tbody>
</table>

**Example**

```plaintext
<cfquery name = "SearchByDepartment" datasource = "cfdocexamples">
SELECT Department
FROM Employees
</cfquery>

<h3>ListValueCountNoCase Example</h3>
<p>This example uses ListValueCountNoCase to count employees in a department.</p>
<form action = "listvaluecountnocase.cfm">
<p>Select a department:</p>
<select name = "departmentName">
<option value = "Accounting">Accounting</option>
<option value = "Administration">Administration</option>
<option value = "Engineering">Engineering</option>
</select>
</form>
```
<option value = "Sales">
  Sales
</OPTION>
</select>
</select>
<input type = "Submit" name = "Submit" value = "Search Employee List">
</form>
<!--- wait to have a string for searching defined --->
<cfif IsDefined("FORM.Submit") and IsDefined("FORM.departmentName")>
  <cfset myList = ValueList(SearchByDepartment.Department)>
  <cfset numberInDepartment = ListValueCountNoCase(myList, FORM.departmentName)>

  <cfif numberInDepartment is 0>
    <h3>There are no employees in <cfoutput>#FORM.departmentName#</cfoutput></h3>
  </cfif>
  <cfelseif numberInDepartment is 1>
    <cfoutput><p>There is only one person in #FORM.departmentName#.</p></cfoutput>
  </cfelseif>
  <cfelse>
    <cfoutput><p>There are #numberInDepartment# people in #FORM.departmentName#.</p></cfoutput>
  </cfelseif>
</cfif>
LJustify

Description
Left justifies characters in a string of a specified length.

Returns
A copy of a string, left-justified.

Category
Display and formatting functions, String functions

Function syntax
LJustify(string, length)

See also
CJustify, RJustify

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>string</td>
<td>A string or a variable that contains one</td>
</tr>
<tr>
<td>length</td>
<td>Length of field in which to justify string</td>
</tr>
</tbody>
</table>

Example
<!--- This example shows how to use LJustify --->
<cfparam name = "jstring" default = "">

<cfif IsDefined("FORM.justifyString")>
  <cfset jstring = LJustify(FORM.justifyString, 35)>
</cfif>
<html>
<head>
<title>LJustify Example</title>
</head>
<body>

<h3>LJustify Function</h3>
<p>Enter a string, and it will be left justified within the sample field

<form action = "ljustify.cfm">
  <p><input type = "Text" value = "<cfoutput>#jString#</cfoutput>" size = 35 name = "justifyString">
</form>
Log

Description
Calculates the natural logarithm of a number. Natural logarithms are based on the constant e (2.71828182845904).

Returns
The natural logarithm of a number.

Category
Mathematical functions

Function syntax
Log(number)

See also
Exp, Log10

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>number</td>
<td>Positive real number for which to calculate the natural logarithm</td>
</tr>
</tbody>
</table>

Example
<h3>Log Example</h3>

<cfif IsDefined("FORM.number")>
  <cfoutput>
    <p>Your number, #FORM.number#</p>
    <br>#FORM.number# raised to the E power: #exp(FORM.number)#
    <cfif FORM.number LTE 0><br>Enter a positive real number to get its natural logarithm<br></cfif>
    <cfelse><br>The natural logarithm of #FORM.number#: #log(FORM.number)#</cfelse>
    <cfif FORM.number LTE 0><br>Enter a positive real number to get its logarithm to base 10<br></cfif>
    <cfelse><br>The logarithm of #FORM.number# to base 10: #log10(FORM.number)#</cfelse>
  </cfoutput>
</cfif>

<cfform action = "log.cfm">
Enter a number to see its value raised to the E power, its natural logarithm, and the logarithm of number to base 10.
<input type = "Text" name = "number" message = "You must enter a number" validate = "float" required = "No">
<input type = "Submit" name = "">
</cfform>
Log10

Description
Calculates the logarithm of number, to base 10.

Returns
Number; the logarithm of number, to base 10.

Category
Mathematical functions

Function syntax
Log10(number)

See also
Exp, Log

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>number</td>
<td>Positive real number for which to calculate the logarithm</td>
</tr>
</tbody>
</table>

Example

```html
<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<br/>
    <cfelse><br/>The natural logarithm of #FORM.number#: #log(FORM.number)#<br/>
    </cfif>
    <cfif #FORM.number# LTE 0><br/>You must enter a positive real number to see the logarithm of that number to base 10<br/>
    <cfelse><br/>The logarithm of #FORM.number# to base 10: #log10(FORM.number)#
    </cfif>
  </cfoutput>
</cfif>
<cfform action = "log10.cfm">
  Enter a number to find its value raised to the E power, its natural logarithm, and the logarithm of number to base 10.
  <cinput type = "Text" name = "number" message = "You must enter a number" validate = "float" required = "No">
  <input type = "Submit" name = "">
</cfform>
```
LSCurrencyFormat

Description
Formats a number in a locale-specific currency format. For countries that use the euro, the result depends on the JVM.

Returns
A formatted currency value.

Category
Display and formatting functions, International functions

Function syntax
LSCurrencyFormat(number [, type])

See also

History
ColdFusion MX: Changed formatting behavior: this function might return different formatting than in earlier releases. If a negative number is passed to it, it returns a negative number. If type = "local", it returns the value in the current locale's standard format. If type = "international", it returns the value in the current locale's international standard format. This function uses Java standard locale formatting rules on all platforms.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>number</td>
<td>Currency value</td>
</tr>
<tr>
<td>type</td>
<td>• local: the currency format and currency symbol used in the locale.&lt;br&gt;- With JDK 1.3, the default for Euro Zone countries is their local currency.&lt;br&gt;- With JDK 1.4, the default for Euro Zone countries is the euro.&lt;br&gt;• international: the international standard currency format and currency symbol of the locale.&lt;br&gt;• none: the currency format used in the locale; no currency symbol</td>
</tr>
</tbody>
</table>

Usage
This function uses Java standard locale formatting rules on all platforms.

Note: With a Sun 1.3.1-compliant JVM, use the LSEuroCurrencyFormat function to format euro currency values.
### Currency output

The following table shows sample currency output. For locales that use Euro, the Local and International columns contain two entries. The first is entry is the result with a Sun the 1.4.1-compliant JVM, the second entry is the result with a 1.3.1-compliant JVM.

<table>
<thead>
<tr>
<th>Locale</th>
<th>Type = Local</th>
<th>Type = International</th>
<th>Type = None</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chinese (China)</td>
<td>¥100,000.00</td>
<td>CNY100,000.00</td>
<td>100,000.00</td>
</tr>
<tr>
<td>Chinese (Hong Kong)</td>
<td>HK$100,000.00</td>
<td>HKD100,000.00</td>
<td>100,000.00</td>
</tr>
<tr>
<td>Chinese (Taiwan)</td>
<td>NT$100,000.00</td>
<td>TWD100,000.00</td>
<td>100,000.00</td>
</tr>
<tr>
<td>Dutch (Belgian)</td>
<td>100,000.00 €</td>
<td>BEF100,000.00</td>
<td>100,000.00</td>
</tr>
<tr>
<td>Dutch (Standard)</td>
<td>€100,000.00</td>
<td>NLG100,000.00</td>
<td>100,000.00</td>
</tr>
<tr>
<td>English (Australian)</td>
<td>$100,000.00</td>
<td>AUD100,000.00</td>
<td>100,000.00</td>
</tr>
<tr>
<td>English (Canadian)</td>
<td>$100,000.00</td>
<td>CAD100,000.00</td>
<td>100,000.00</td>
</tr>
<tr>
<td>English (New Zealand)</td>
<td>$100,000.00</td>
<td>NZD100,000.00</td>
<td>100,000.00</td>
</tr>
<tr>
<td>English (UK)</td>
<td>£100,000.00</td>
<td>GBP100,000.00</td>
<td>100,000.00</td>
</tr>
<tr>
<td>English (US)</td>
<td>$100,000.00</td>
<td>USD100,000.00</td>
<td>100,000.00</td>
</tr>
<tr>
<td>French (Belgian)</td>
<td>100,000.00 €</td>
<td>EUR100,000.00</td>
<td>100,000.00</td>
</tr>
<tr>
<td>French (Canadian)</td>
<td>100 000,00 $</td>
<td>CAD100 000.00</td>
<td>100 000.00</td>
</tr>
<tr>
<td>French (Standard)</td>
<td>100 000,00 €</td>
<td>EUR100 000.00</td>
<td>100 000,00</td>
</tr>
<tr>
<td>French (Swiss)</td>
<td>SFr. 100’000.00</td>
<td>CHF100’000.00</td>
<td>100’000.00</td>
</tr>
<tr>
<td>German (Austrian)</td>
<td>€100,000.00</td>
<td>EUR100,000.00</td>
<td>100,000.00</td>
</tr>
<tr>
<td>German (Standard)</td>
<td>100,000.00 €</td>
<td>EUR100,000.00</td>
<td>100,000.00</td>
</tr>
<tr>
<td>German (Swiss)</td>
<td>SFr. 100’000.00</td>
<td>CHF100’000.00</td>
<td>100’000.00</td>
</tr>
<tr>
<td>Italian (Standard)</td>
<td>€100,000.00</td>
<td>EUR100,000.00</td>
<td>100,000.00</td>
</tr>
<tr>
<td>Italian (Swiss)</td>
<td>SFr. 100’000.00</td>
<td>CHF100’000.00</td>
<td>100’000.00</td>
</tr>
<tr>
<td>Japanese</td>
<td>¥100,000</td>
<td>JPY100,000</td>
<td>JPY100,000</td>
</tr>
<tr>
<td>Korean</td>
<td>₩100,000</td>
<td>KRW100,000</td>
<td>100,000</td>
</tr>
<tr>
<td>Norwegian (Bokmal)</td>
<td>kr 100 000.00</td>
<td>NOK100 000.00</td>
<td>100 000.00</td>
</tr>
<tr>
<td>Norwegian (Nynorsk)</td>
<td>kr 100 000.00</td>
<td>NOK100 000.00</td>
<td>100 000.00</td>
</tr>
<tr>
<td>Portuguese (Brazilian)</td>
<td>R$100,000.00</td>
<td>BRC100,000.00</td>
<td>100,000.00</td>
</tr>
</tbody>
</table>
Note: ColdFusion maps Spanish (Modern) to the Spanish (Standard) format.

To set the default display format of date, time, number, and currency values, use the `SetLocale` function.

**Example**

```coldfusion
<h3>LSCurrencyFormat Example</h3>
<p>LSCurrencyFormat returns a currency value using the locale convention. Default value is "local."

<!--- loop through list of locales; show currency values for 100,000 units --->
<cfloop LIST = "#Server.Coldfusion.SupportedLocales#" index = "locale" delimiters = ",">
<cfset oldlocale = SetLocale(locale)>
<cfoutput>
<p>
<b><i>#locale#</i></b>
<br>
Local: #LSCurrencyFormat(100000, "local")#
<br>
International: #LSCurrencyFormat(100000, "international")#
<br>
None: #LSCurrencyFormat(100000, "none")#
<br>
<hr noshade>
</cfoutput>
</cfloop>
```
**LSDateFormat**

**Description**

Formats the date part of a date/time value in a locale-specific format.

**Returns**

A formatted date/time value. If no mask is specified, the value is formatted according to the locale setting of the client computer.

**Category**

Date and time functions, Display and formatting functions, International functions

**Function syntax**

`LSDateFormat(date [, mask])`

**See also**

`LSParseDateTime, LSTimeFormat, DateFormat, SetLocale; “Handling data in ColdFusion MX” in Chapter 17, “Developing Globalized Applications,” in ColdFusion MX Developer’s Guide`

**History**

ColdFusion MX:

- Changed formatting behavior: this function might return different formatting than in earlier releases. This function uses Java standard locale formatting rules on all platforms.
- Added support for the following mask parameter options: short, medium, long, and full.
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>date</td>
<td>A date/time object, in the range 100 AD–9999 AD.</td>
</tr>
</tbody>
</table>
| mask      | Characters that show how ColdFusion displays the date:  
• d: Day of month. Digits; no leading zero for single-digit days  
• dd: Day of month. Digits; leading zero for single-digit days  
• ddd: Day of week, abbreviation  
• dddd: Day of week. Full name  
• m: Month. Digits; no leading zero for single-digit months  
• mm: Month. Digits; leading zero for single-digit months  
• mmm: Month. Abbreviation (if appropriate)  
• mmmm: Month. Full name  
• y: Year. Last two digits; no leading zero for years less than 10  
• yy: Year. Last two digits; leading zero for years less than 10  
• yyyy: Year. Four digits  
• gg: Period/era string. Not processed. Reserved for future use  
The following conform to Java locale-specific time encoding standards. Their exact formats depend on the locale:  
• short: dd, mm, and yy separated by / marks  
• medium: text format using mmm, d, and yyyy  
• long: text format using mmmm, d, and yyyy  
• full: text format using ddd, mmmm, d, and yyyy  
The default value is medium  
For more information on formats, see LSParseDateTime.

Usage

This function uses Java standard locale formatting rules on all platforms.

When passing date/time value as a string, enclose it in quotation marks. Otherwise, it is interpreted as a number representation of a date/time object.

To calculate a difference between time zones, use the GetTimeZoneInfo function.

Example

```cfml
<h3>LSDateFormat Example</h3>
<p>LSDateFormat formats the date part of a date/time value using the locale convention.
</p>
<cfloop 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>
</cfoutput>
</cfloop>
```

For more information on formats, see LSDateTime.
<hr noshade>
</cfoutput>
</cfloop>
LSEuroCurrencyFormat

Description

Formats a number in a locale-specific currency format.

Returns

A formatted currency value. For countries in the Euro currency zone, the function uses the locale’s rule’s for formatting currency in euros.

Category

Display and formatting functions, International functions

Function syntax

LSEuroCurrencyFormat(currency-number [, type ])

See also


History

ColdFusion MX: Changed formatting behavior: this function might return different formatting than in earlier releases. This function uses Java locale formatting rules on all platforms, except that it uses the rule detailed in the Usage section for countries in the Euro currency zone. As a result, it format currencies for non Euro zone locales using the country's currency, not euros.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>currency-number</td>
<td>Currency value.</td>
</tr>
</tbody>
</table>
| type       | • local: the currency format used in the locale. (Default.)  
• international: the international standard currency format of the locale. For example, EUR10.00  
• none: the currency format used in the locale; no currency symbol |

Usage

This function uses euro currency formatting rules for all JVM versions, as follows:

• If the country of the current locale belongs to the Euro Zone (whose members have converted to the euro) the formatted output for the local type includes the Euro currency sign (€); for the international type, the output includes the euro currency symbol (EUR). If the value is negative, the format includes a negative sign before the value or parentheses around the value, according to the formatting rules of the current locale.

• If the country of the current locale is not in the Euro Zone, the currency sign or symbol of the current locale displays. If the value is negative, the format includes a negative sign before the value or parentheses around the value, according to the formatting rules of the current locale.

For a list of the locale options that ColdFusion supports, and information on setting the default display format of date, time, number, and currency values, see SetLocale on page 835.
# Currency output

The following table shows examples of currency output:

<table>
<thead>
<tr>
<th>Locale</th>
<th>Type = Local</th>
<th>Type = International</th>
<th>Type = None</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chinese (China)</td>
<td>¥100,000.00</td>
<td>CNY100,000.00</td>
<td>100,000.00</td>
</tr>
<tr>
<td>Chinese (Hong Kong)</td>
<td>HK$100,000.00</td>
<td>HKD100,000.00</td>
<td>100,000.00</td>
</tr>
<tr>
<td>Chinese (Taiwan)</td>
<td>NT$100,000.00</td>
<td>TWD100,000.00</td>
<td>100,000.00</td>
</tr>
<tr>
<td>Dutch (Belgian)</td>
<td>€100,000.00</td>
<td>EUR100,000.00</td>
<td>100,000.00</td>
</tr>
<tr>
<td>Dutch (Standard)</td>
<td>€100,000.00</td>
<td>EUR100,000.00</td>
<td>100,000.00</td>
</tr>
<tr>
<td>English (Australian)</td>
<td>£100,000.00</td>
<td>AUD100,000.00</td>
<td>100,000.00</td>
</tr>
<tr>
<td>English (Canadian)</td>
<td>£100,000.00</td>
<td>CAD100,000.00</td>
<td>100,000.00</td>
</tr>
<tr>
<td>English (New Zealand)</td>
<td>£100,000.00</td>
<td>NZD100,000.00</td>
<td>100,000.00</td>
</tr>
<tr>
<td>English (UK)</td>
<td>£100,000.00</td>
<td>GBP100,000.00</td>
<td>100,000.00</td>
</tr>
<tr>
<td>English (US)</td>
<td>$100,000.00</td>
<td>USD100,000.00</td>
<td>100,000.00</td>
</tr>
<tr>
<td>French (Belgian)</td>
<td>€100,000.00</td>
<td>EUR100,000.00</td>
<td>100,000.00</td>
</tr>
<tr>
<td>French (Canadian)</td>
<td>€100,000.00</td>
<td>CAD100,000.00</td>
<td>100,000.00</td>
</tr>
<tr>
<td>French (Standard)</td>
<td>€100,000.00</td>
<td>EUR100,000.00</td>
<td>100,000.00</td>
</tr>
<tr>
<td>French (Swiss)</td>
<td>CHF100,000.00</td>
<td>CHF100,000.00</td>
<td>100,000.00</td>
</tr>
<tr>
<td>German (Austrian)</td>
<td>€100,000.00</td>
<td>EUR100,000.00</td>
<td>100,000.00</td>
</tr>
<tr>
<td>German (Standard)</td>
<td>€100,000.00</td>
<td>EUR100,000.00</td>
<td>100,000.00</td>
</tr>
<tr>
<td>German (Swiss)</td>
<td>CHF100,000.00</td>
<td>CHF100,000.00</td>
<td>100,000.00</td>
</tr>
<tr>
<td>Italian (Standard)</td>
<td>€100,000.00</td>
<td>EUR100,000.00</td>
<td>100,000.00</td>
</tr>
<tr>
<td>Italian (Swiss)</td>
<td>CHF100,000.00</td>
<td>CHF100,000.00</td>
<td>100,000.00</td>
</tr>
<tr>
<td>Japanese</td>
<td>¥100,000</td>
<td>JPY100,000</td>
<td>JPY100,000</td>
</tr>
<tr>
<td>Korean</td>
<td>₩100,000</td>
<td>KRW100,000</td>
<td>100,000</td>
</tr>
<tr>
<td>Norwegian (Bokmal)</td>
<td>kr 100,000</td>
<td>NOK100,000</td>
<td>100,000</td>
</tr>
<tr>
<td>Norwegian (Nynorsk)</td>
<td>kr 100,000</td>
<td>NOK100,000</td>
<td>100,000</td>
</tr>
<tr>
<td>Portuguese (Brazilian)</td>
<td>R$100,000</td>
<td>BRL 100,000</td>
<td>100,000</td>
</tr>
<tr>
<td>Portuguese (Standard)</td>
<td>€100,000.00</td>
<td>EUR100,000.00</td>
<td>100,000.00</td>
</tr>
<tr>
<td>Spanish (Mexican)</td>
<td>$100,000.00</td>
<td>MXN100,000</td>
<td>100,000.00</td>
</tr>
<tr>
<td>Spanish (Modern)</td>
<td>€100,000.00</td>
<td>EUR100,000.00</td>
<td>100,000.00</td>
</tr>
<tr>
<td>Spanish (Standard)</td>
<td>€100,000.00</td>
<td>EUR100,000.00</td>
<td>100,000.00</td>
</tr>
<tr>
<td>Swedish</td>
<td>kr 100,000</td>
<td>SEK100,000</td>
<td>100,000</td>
</tr>
</tbody>
</table>

**Note:** ColdFusion uses the Spanish (Standard) formats for Spanish (Modern) and Spanish (Standard).
The following example shows how the function formats negative values. The format includes a negative sign before the value, or parentheses around the value, according to the formatting rules of the current locale.

<table>
<thead>
<tr>
<th>Input value</th>
<th>Output if locale = French (Standard)</th>
<th>Output if locale = English (US)</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1234.56</td>
<td>-1234.56 €</td>
<td>($1,234.56)</td>
</tr>
</tbody>
</table>

Example

```coldfusion
<h3>LSEuroCurrencyFormat Example</h3>
<p>LSEuroCurrencyFormat returns a currency value using the locale convention. Default value is "local."
</p>
<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>
</cfoutput></cfloop>
LSIsCurrency

Description
Determine whether a string is a valid representation of a currency amount in the current locale.

Returns
True, if the parameter is formatted as a valid currency amount, including the appropriate currency indicator. The return value is True for amounts in the local, international, or none currency formats.

Category
Display and formatting functions, Decision functions, International functions

Function syntax
LSIsCurrency(string)

See also
GetLocale, SetLocale, LSCurrencyFormat

History
ColdFusion MX: Changed formatting behavior: this function might return a different result than in earlier releases. This function uses Java standard locale formatting rules on all platforms; the results might vary depending upon the JVM; for example, Sun JVM 1.4.1 requires euro format the local currency if the current locale’s country belongs to the Euro Zone.

Parameters
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>string</td>
<td>A currency string or a variable that contains one.</td>
</tr>
</tbody>
</table>

Usage
For examples of ColdFusion code and output that shows differences between earlier ColdFusion releases and ColdFusion MX in accepting input formats and displaying output, see LSCurrencyFormat.

Note: If the locale belongs to a Euro zone country and the currency is a correctly formatted euro value for the locale, this function returns True for all JVMs, including Sun 1.3.1. As a result, with 1.3.1-compliant JVMs, the LSIsCurrency function does not ensure that LSParseCurrency returns a value. If a currency uses the older country-specific format for Euro Zone locales, the LSIsCurrency function returns False for newer JVMs, such as Sun 1.4.1, and True for older JVMs, such as Sun 1.3.1.

Note: To set the default display format of date, time, number, and currency values, use the SetLocale function.

Example
<h3>LSIsCurrency Example</h3>

<cfif IsDefined("FORM.locale")>
<!--- if locale is defined, set locale to that entry --->
<cfset NewLocale = SetLocale(FORM.locale)>

LSIsCurrency 755
<p>Is the value "<cfoutput>#FORM.myValue#</cfoutput>" a proper currency value for <cfoutput>#GetLocale()#</cfoutput>?
<p>Answer: <cfoutput>#$LSIsCurrency(FORM.myValue)#</cfoutput></cfif>
<p><form action = "LSIsCurrency.cfm">
<p>Select a locale for which you would like to check a currency value:
<p><cfset serverLocale = GetLocale()>
LSIsDate

Description
Determine whether a string is a valid representation of a date/time value in the current locale.

Returns
True, if the string can be formatted as a date/time value in the current locale; False, otherwise.

Category
Date and time functions, Display and formatting functions, International functions

Function syntax
LSIsDate(string)

See also
CreateDateDateTime, GetLocale, IsNumericDate, LSDateFormat, ParseDateTime, SetLocale;
“Handling data in ColdFusion MX” in Chapter 17, “Developing Globalized Applications,” in ColdFusion MX Developer’s Guide

History
ColdFusion MX:
• Changed formatting behavior: this function might return a different result than in earlier releases. This function uses Java standard locale formatting rules on all platforms.
• Changed behavior: this function accepts a dash or hyphen character only in the Dutch(Standard) and Portuguese (Standard) locales. If called this way (for example, LsIsDate("3-1-2002") in any other locale, this function returns False. (Earlier releases returned Tru.
• Changed behavior: when using the SUN JRE 1.3.1 on an English(UK) locale, this function returns False for a date that has a one-digit month or day (for example, 1/1/01). To work around this, insert a zero in a one-digit month or day (for example, 01/01/01).

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>string</td>
<td>A string or a variable that contains one</td>
</tr>
</tbody>
</table>

Usage
A date/time object is in the range 100 AD–9999 AD.

To set the default display format of date, time, number, and currency values, use the SetLocale function.

Example

```<h3>LSIsDate Example</h3>
<cfif IsDefined("FORM.locale")>
  <!--- if locale is defined, set locale to that entry --->
  <cfset NewLocale = SetLocale(FORM.locale)>
```
<p>Is the value "<cfoutput>#FORM.myValue#</cfoutput>" a proper date value for <cfoutput>#GetLocale()#</cfoutput>?
<p>Answer: <cfoutput>#LSIsDate(FORM.myValue)#</cfoutput>
</cfif>
<p><form action = "LSIsDate.cfm">
<p>Select a locale for which you would like to check a date value:
<!--- check the current locale for server --->
<cfset serverLocale = GetLocale()>
</cfif>
LSIsNumeric

Description
Determines whether a string is a valid representation of a number in the current locale.

Returns
True, if the string represents a number the current locale; False, otherwise.

Category
Decision functions, International functions, String functions

Function syntax
LSIsNumeric(string)

See also
GetLocale, SetLocale; “Handling data in ColdFusion MX” in Chapter 17, “Developing Globalized Applications,” in ColdFusion MX Developer’s Guide

History
ColdFusion MX: Changed formatting behavior: this function might return a different result than in earlier releases. This function uses Java standard locale formatting rules on all platforms.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>string</td>
<td>A string or a variable that contains one</td>
</tr>
</tbody>
</table>

Usage
To set the default display format of date, time, number, and currency values, use the SetLocale function.

Example
<h3>LSIsNumeric Example</h3>

<cfif IsDefined("FORM.locale")>
  <!--- if locale is defined, set locale to that entry --->
  <cfset NewLocale = SetLocale(FORM.locale)>
  <p>Is the value "<cfoutput>#FORM.myValue#</cfoutput>" a proper numeric value for <cfoutput>#GetLocale()#</cfoutput>?
  <p>Answer: <cfoutput>#LSIsNumeric(FORM.myValue)#</cfoutput>
</cfif>

<p><form action = "LSIsNumeric.cfm">
  Select a locale for which to check a numeric value:
  ...

  LSIsNumeric  759
**LSNumberFormat**

**Description**

Formats a number in a locale-specific format.

**Returns**

A formatted number.

- If no mask is specified, it returns the number formatted as an integer
- If no mask is specified, truncates the decimal part; for example, it truncates 34.57 to 35
- If the specified mask cannot correctly mask a number, it returns the number unchanged
- If the parameter value is "" (an empty string), it returns 0.

**Category**

Display and formatting functions, International functions

**Function syntax**

```
LSNumberFormat(number [, mask ])
```

**See also**

GetLocale, SetLocale; “Handling data in ColdFusion MX” in Chapter 17, “Developing Globalized Applications,” in ColdFusion MX Developer’s Guide

**History**

ColdFusion MX:

- Changed behavior: if the specified mask format cannot correctly mask a number, this function returns the number unchanged. (In earlier releases, it truncated the number or threw an error.) (If no mask is specified, ColdFusion MX truncates the decimal part as ColdFusion 5 does. For example, it truncates 1234.567 to 1235.)
- Changed formatting behavior: this function might return different formatting than in earlier releases. This function uses Java standard locale formatting rules on all platforms.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>number</td>
<td>Number to format</td>
</tr>
<tr>
<td>mask</td>
<td>LSNumberFormat mask characters apply, except: dollar sign, comma, and dot are mapped to their locale-specific equivalents.</td>
</tr>
</tbody>
</table>

The following table lists the LSNumberFormat mask characters:

<table>
<thead>
<tr>
<th>Character</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>_ (Underscore)</td>
<td>Digit placeholder.</td>
</tr>
<tr>
<td>9</td>
<td>Digit placeholder. (Shows decimal places more clearly than _. )</td>
</tr>
<tr>
<td>.</td>
<td>Location of a mandatory decimal point (or locale-appropriate symbol).</td>
</tr>
</tbody>
</table>
Note: If you do not specify a sign for the mask, positive and negative numbers do not align in columns. To put a plus sign or space before positive numbers and a minus sign before negative numbers, use the plus or hyphen mask character, respectively.

Usage

This function uses Java standard locale formatting rules on all platforms.

The position of symbols in format masks determines where the codes take effect. For example, if you put a dollar sign at the far left of a format mask, ColdFusion displays a dollar sign at the left edge of the formatted number. If you separate the dollar sign on the left edge of the format mask by at least one underscore, ColdFusion displays the dollar sign just to the left of the digits in the formatted number.

These examples show how symbols determine formats:

<table>
<thead>
<tr>
<th>Number</th>
<th>Mask</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.37</td>
<td>$______</td>
<td>&quot;$ 4.37&quot;</td>
</tr>
<tr>
<td>4.37</td>
<td>_$_____</td>
<td>&quot;$4.37&quot;</td>
</tr>
</tbody>
</table>

The positioning can also show where to put a minus sign for negative numbers:

<table>
<thead>
<tr>
<th>Number</th>
<th>Mask</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>-4.37</td>
<td>-_____</td>
<td>&quot;- 4.37&quot;</td>
</tr>
<tr>
<td>-4.37</td>
<td>-_____</td>
<td>&quot;-4.37&quot;</td>
</tr>
</tbody>
</table>

The positions for a symbol are: far left, near left, near right, and far right. The left and right positions are determined by the side of the decimal point on which the code character is shown. For formats that do not have a fixed number of decimal places, you can use a caret (^) to separate the left fields from the right.
An underscore determines whether the code is placed in the far or near position. Most code characters' effect is determined by the field in which they are located. This example shows how to specify where to put parentheses to display negative numbers:

<table>
<thead>
<tr>
<th>Number</th>
<th>Mask</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.21</td>
<td>C(<em>&quot;</em>&quot;)</td>
<td>&quot;(3.21)&quot;</td>
</tr>
<tr>
<td>3.21</td>
<td>C(&quot;_&quot;)</td>
<td>&quot;(3.21)&quot;</td>
</tr>
<tr>
<td>3.21</td>
<td>C(&quot;_&quot;)</td>
<td>&quot;(3.21)&quot;</td>
</tr>
<tr>
<td>3.21</td>
<td>C(&quot;_&quot;)</td>
<td>&quot;(3.21)&quot;</td>
</tr>
</tbody>
</table>

To set the default display format of date, time, number, and currency values, use the `SetLocale` function.

When converting from string to double, to prevent rounding errors, this function adds a rounding factor of 1.55431234752E-014 to the converted number. For example, without adding the rounding factor, converting the string value 1.275 to double with two digits of precision results in a value of 1.27499999999999999, which would be rounded up to 1.27. By adding the rounding factor, the conversion correctly results in a value of 1.28.

If you round off a double, such as 1.99499999999999999999999999999999, where the last decimal is 10E-14, the rounding factor can cause an incorrect result.

Example

```coldfusion
<h3>LSNumberFormat Example</h3>
<p>LSNumberFormat returns a number value using the locale convention.
</p>
<cfset LIST = "#Server.Coldfusion.SupportedLocales#"
index = "locale" delimiters = "," >
<cfloop LIST = #LIST# index = "locale">
<cfoutput><p><b><i>#locale#</i></b><br>
#LSNumberFormat(-1234.5678, "_________")#<br>
#LSNumberFormat(-1234.5678, "_________.___")#<br>
#LSNumberFormat(1234.5678, "_________")#<br>
#LSNumberFormat(1234.5678, "_________.___")#<br>
#LSNumberFormat(-1234.5678, "$__(_________")#<br>
#LSNumberFormat(-1234.5678, "$__(_________.___)")#<br>
#LSNumberFormat(1234.5678, "+_________")#<br>
#LSNumberFormat(1234.5678, "-_________")#<br>
</cfoutput>
</cfloop>
```
LSParseCurrency

Description

Converts a locale-specific currency string into a formatted number. Attempts conversion by comparing the string with each the three supported currency formats (none, local, international) and using the first that matches.

Returns

A formatted number (string representation of a number) that matches the value of the parameter.

Category

International functions, String functions

Function syntax

LSParseCurrency(string)

See also

LSParseEuroCurrency, LSCurrencyFormat, LSEuroCurrencyFormat, LSIsCurrency;

History

ColdFusion MX: Changed formatting behavior: this function might return different formatting than in earlier releases. This function uses Java standard locale formatting rules on all platforms.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>string</td>
<td>A locale-specific string a variable that contains one</td>
</tr>
</tbody>
</table>

Usage

This function uses the locale formatting rules of the JVM (specified in the ColdFusion Administrator Java and JVM page) on all platforms. These rules changed between Sun JVM 1.3.1 and JVM 1.4.1:

- JVM 1.3.1 requires that the local and international versions of currencies of countries in the Euro zone be formatted using the older, country-specific designations, such as 100.000,00 DM or DEM100.000,00 for the German (Standard) locale. Use the LSParseEuroCurrency function to parse euro currencies in these locales with JVM 1.3.1.
- JVM 1.4.1 requires that currencies for Euro zone countries be expressed as euros; for example 100.000,00 € or EUR100.000,00.

Note: The LSIsCurrency function always returns True if the locale is in the Euro currency zone and the currency is expressed in euros, including when using JVM 1.3.1. As a result, with older JVMs, LSIsCurrency does not ensure that LSParseCurrency returns a value.

To set the default display format of date, time, number, and currency values, use the SetLocale function.
For a list of the locale-specific formats used to parse the currency, see LSCurrencyFormat.

Example

```cfml
<h3>LSParseCurrency Example</h3>
<p>LSParseCurrency converts a locale-specific currency string to a number. Attempts conversion through each of the three default currency formats.
</p><cfloop LIST = "#Server.Coldfusion.SupportedLocales#" index = "locale" delimiters = ",">
<cfset oldlocale = SetLocale(locale)>
<cfoutput><p><b><i>#locale#</i></b><br>
Local: #LSCurrencyFormat(123456.78, "local")#
Parsed local Currency:
#LSParseCurrency(LSCurrencyFormat(123456,"local"))#
International: #LSCurrencyFormat(123456.78999, "international")#
Parsed International Currency:
#LSParseCurrency(LSCurrencyFormat(123456.78999,"international"))#
None: #LSCurrencyFormat(123456.78999, "none")#
Parsed None formatted currency:
#LSParseCurrency(LSCurrencyFormat(123456.78999,"none"))#
</cfoutput>
</cfloop>
```
LSParseDateTime

Description

Converts a string that is a valid date/time representation in the current locale into a date/time object.

Returns

A date/time object.

Category

Date and time functions, Display and formatting functions, International functions, String functions

Function syntax

LSParseDateTime(date/time-string)

See also

LSDateFormat, ParseDateTime, SetLocale, GetLocale; “Locales” in Chapter 17, “Developing Globalized Applications,” in ColdFusion MX Developer’s Guide

History

ColdFusion MX:

• Changed formatting behavior: this function might not parse string formats that worked with earlier releases. This function uses Java standard locale formatting rules on all platforms.

• Changed how the date/time-string parameter value is processed: ColdFusion processes the date/time-string parameter value time zone information differently than in earlier releases, as described in the Usage section.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>date/time-string</td>
<td>A string a variable that contains one, in a format that is readable in the current locale.</td>
</tr>
</tbody>
</table>

Usage

This function can parse any date, time, or date/time combination that conforms to Java standard locale formatting rules for the current locale.

The following table lists some of the date/time values you can pass to this function in the English (US) locale. You can also pass only the date or the time parts of these formats:

<table>
<thead>
<tr>
<th>Format</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>m/dd/yy h:mm:ss</td>
<td>1/30/02 7:02:33</td>
</tr>
<tr>
<td>m/dd/yy h:mm tt</td>
<td>1/30/02 7:02 AM</td>
</tr>
<tr>
<td>m/dd/yyyy h:mm</td>
<td>1/30/2002 7:02 AM</td>
</tr>
</tbody>
</table>
Chapter 3: ColdFusion Functions

Valid dates are in the range 100 AD–9999 AD. Two digit years in the range 00-29 are interpreted as being 2000-2029. Two digit years in the range 30-99 are interpreted as being 1930-1999

This function corrects for differences between the current time zone and any time zone specified in the input parameter.

- If a time zone specified in the date/time-string parameter is different from the time zone setting of the computer, ColdFusion adjusts the time value to its equivalent in the computer time zone.
- If a time zone is not specified in the date/time-string parameter, ColdFusion does not adjust the time value.

**Note:** This function does not accept POP dates, which include a time zone offset value.

**Example**

```coldfusion
<h3>LSParseDateTime Example - returns a locale-specific date/time object</h3>
<!--- loop through a list of locales and show date values for Now()--->
<cfloop LIST = "#Server.Coldfusion.SupportedLocales#" index = "locale" delimiters = ",">
    <cfset oldlocale = SetLocale(locale)>
    <cfoutput><p><b><i>#locale#</i></b><br>
    Locale-specific formats: <br>
    #LSDateFormat(Now(), "mmm-dd-yyyy")# #LSTimeFormat(Now())#<br>
    #LSDateFormat(Now(), "mmmm dd, 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>
    </p>
</cfoutput>
</cfloop>
```

<table>
<thead>
<tr>
<th>Format</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>mmm dd, yyyy h:mm:ss tt</code></td>
<td>Jan 30, 2002 7:02:12 AM</td>
</tr>
<tr>
<td><code>mmmm dd, yyyy h:mm:ss tt zzz</code></td>
<td>January 30, 2002 7:02:23 AM PST</td>
</tr>
<tr>
<td><code>ddd, mmm dd, yyyy h:mm:ss</code></td>
<td>Wed, Jan 30, 2002 07:02:12</td>
</tr>
<tr>
<td><code>dddd, mmmm dd, yyyy h:mm:ss tt zzz</code></td>
<td>Wednesday, January 30, 2002 7:02:12 AM PST</td>
</tr>
</tbody>
</table>
LSParseEuroCurrency

Description

Formats a locale-specific currency string as a number. Attempts conversion through each of the default currency formats (none, local, international). Ensures correct handling of euro currency for Euro zone countries.

Returns

A formatted number that matches the value of the string.

Category

International functions, String functions

Function syntax

LSParseEuroCurrency(currency-string)

See also


History

ColdFusion MX: Changed formatting behavior: this function might return different formatting than in earlier releases. This function uses Java locale formatting rules on all platforms, except that it uses the rule detailed in the Usage section for countries in the Euro currency zone.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>currency-string</td>
<td>Locale-specific string or a variable that contains one.</td>
</tr>
</tbody>
</table>

Usage

This function determines whether the current locale’s country belongs to the Euro Zone, whose members have converted to the euro; if so, the currency-string parameter must be formatted in euros on all JVMs, including Sun JVM 1.3.1. If the country is not in the Euro zone, the string must follow the locale formatting rules of the JVM. For examples of valid currency formats in all supported locales, see LSEuroCurrencyFormat on page 752.

For a list of the locale options that ColdFusion supports, and information on setting the default display format of date, time, number, and currency values, see SetLocale.

Example

<h3>LSParseEuroCurrency Example</h3>
<p>Loop through all available locales. Create string representations of the value 123,456 in the three supported currency formats, and parse the results back to numbers.</p>
<cfloop list="&Server.Coldfusion.SupportedLocales&" index="locale" delimiters=".">
<cfset oldlocale = SetLocale(locale)>
...
Current Locale: <b><i>#locale#</i></b><br>

Value in local currency: #localCurrency#<br>
Parsed using LSParseEuroCurrency:
#LSParseEuroCurrency(localCurrency)#<br>

Value with International currency formatting: #IntlCurrency#<br>
Parsed using LSParseEuroCurrency:
#LSParseEuroCurrency(IntlCurrency)#<br>

Value with no currency formatting: #currency#<br>
Parsed using LSParseEuroCurrency:
#LSParseEuroCurrency(Currency)#<br>

<hr noshade>
LSParseNumber

Description

Converts a string that is a valid numeric representation in the current locale into a formatted number.

Returns

A formatted number that matches the value of the string.

Category

International functions, String functions

Function syntax

LSParseNumber(string)

See also

LSParseDateTime, SetLocale; “Locales” in Chapter 17, “Developing Globalized Applications” in ColdFusion MX Developer’s Guide

History

ColdFusion MX: Changed formatting behavior: this function might return different formatting than in earlier releases. This function uses Java standard locale formatting rules on all platforms.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>string</td>
<td>A string or a variable that contains one</td>
</tr>
</tbody>
</table>

Usage

This function uses Java standard locale formatting rules on all platforms.

To set the default display format of date, time, number, and currency values, use the SetLocale function.

Example

```cfml
<h3>LSParseNumber Example</h3>
<p>LSParseNumber converts a locale-specific string to a number. Returns the number matching the value of string.</p>
<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>
  </cfoutput>
</cfloop>
```
#LSNumberFormat(1234.5678, "+_________.")#<br>
#LSNumberFormat(1234.5678, "-_________.")#<br>
The actual number:<br>
#LSParseNumber(LSNumberFormat(1234.5678, "_________")#<br>
</cfoutput>
</cfloop>
LSTimeFormat

Description
Formats the time part of a date/time string into a string in a locale-specific format.

Returns
A string representing the time value.

Category
Date and time functions, Display and formatting functions, International functions

Function syntax
LSTimeFormat(time [, mask ])

See also
LSParseDateTime, LSDateFormat, TimeFormat; “Locales” in Chapter 17, “Developing Globalized Applications” in ColdFusion MX Developer’s Guide

History
ColdFusion MX 6.1: Added the mask character L or l to represent milliseconds.

ColdFusion MX:
• Changed formatting behavior: this function might return different formatting than in earlier releases. This function uses Java standard locale formatting rules on all platforms.
• Added support for the following mask parameter options: short, medium, long, and full.
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| string    | - A date/time value  
|           | - A string that is convertible to a time value  
|           | - A date/time object is in the range 100 AD–9999 AD. |
| mask      | Masking characters that determine the format:  
|           | - h: Hours; no leading zero for single-digit hours (12-hour clock)  
|           | - hh: Hours; leading zero for single-digit hours (12-hour clock)  
|           | - H: Hours; no leading zero for single-digit hours (24-hour clock)  
|           | - HH: Hours; leading zero for single-digit hours (24-hour clock)  
|           | - m: Minutes; no leading zero for single-digit minutes  
|           | - mm: Minutes; leading zero for single-digit minutes  
|           | - s: Seconds; no leading zero for single-digit seconds  
|           | - ss: Seconds; leading zero for single-digit seconds  
|           | - I: Milliseconds  
|           | - t: One-character time marker string, such as A or P.  
|           | - tt: Multiple-character time marker string, such as AM or PM |

Usage

This function uses Java standard locale formatting rules on all platforms.

When passing date/time value as a string, enclose it in quotation marks. Otherwise, it is interpreted as a number representation of a date/time object.

To calculate a difference between time zones, use the `GetTimeZoneInfo` function.

To set the default display format of date, time, number, and currency values, use the `SetLocale` function.

If no seconds value is passed to this function, and the mask value is s, the default output seconds format is one zero; for example, `lstimeformat(6:39, "h:m:s")` returns 6:39:0. If the mask value is ss, it returns 6:39:00.

Example

```
<h3>LSTimeFormat Example</h3>

<p>LSTimeFormat returns a time value using the locale convention.

<!--- loop through a list of locales and show time values --->
<cffloop LIST = "#Server.Coldfusion.SupportedLocales#" index = "locale" delimiters = ","&gt;
    <cfset oldlocale = SetLocale(locale)>
    <!--- Set locale to show time format in the locale --->
    &lt;cfset SetLocale(locale) = locale&gt;
    &lt;cfset newlocale = GetLocale(null) &gt;
    &lt;h5&gt;Locale: #locale# &lt;tt&gt;#newlocale#&lt;/tt&gt;&lt;/h5&gt;
    &lt;cfset oldlocale = newlocale&gt;
    &lt;cfset SetLocale(locale) = oldlocale&gt;
    &lt;cfset output = lstimeformat(6:39, "h:m:s")&gt;
    &lt;tt&gt;#output#&lt;/tt&gt;
</cfloop>
```
<cfoutput><p><b><i>#locale#</i></b><br>  #LSTimeFormat(Now())#<br>  #LSTimeFormat(Now(), 'hh:mm:ss')#<br>  #LSTimeFormat(Now(), 'hh:mm:ssst')#<br>  #LSTimeFormat(Now(), 'hh:mm:ssstt')#<br>  #LSTimeFormat(Now(), 'HH:mm:ss')#<br></cfoutput>  <hr noshade>  </cfoutput>
LTrim

Description
Removes leading spaces from a string.

Returns
A copy of the string, without leading spaces.

Category
Display and formatting functions, String functions

Function syntax
LTrim(string)

See also
RTrim, ToBase64

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>string</td>
<td>A string or a variable that contains one</td>
</tr>
</tbody>
</table>

Example

```cfml
<h3>LTrim Example</h3>

<cfif IsDefined("FORM.myText")>
<cfoutput>
<pre>
Your string: "#{FORM.myText}"
Your string: "#{LTrim(FORM.myText)}" (left trimmed)
</pre>
</cfoutput>
</cfif>

<form action = "ltrim.cfm">
<p>Type in some text, and it will be modified by LTrim to remove leading spaces from the left</p>
<p><input type = "Text" name = "myText" value = "   TEST" 
</p><p><input type = "Submit" name = ""></p>
</form>
```
Max

Description
Determine the greater of two numbers.

Returns
The greater of two numbers.

Category
Mathematical functions

Function syntax
Max(number1, number2)

See also
Min

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>number1, number2</td>
<td>Numbers</td>
</tr>
</tbody>
</table>

Example
<h3>Max Example</h3>
<cfif IsDefined("FORM.myNum1")>
  <cfif IsNumeric(FORM.myNum1) and IsNumeric(FORM.myNum2)>
    <p>The maximum of the two numbers is <cfoutput>#Max(FORM.myNum1, FORM.myNum2)#</cfoutput>
    <p>The minimum of the two numbers is <cfoutput>#Min(FORM.myNum1, FORM.myNum2)#</cfoutput>
  </cfif>
</cfif>

<form action = "max.cfm">
  <h3>Enter two numbers, see the maximum and minimum of them</h3>
  Number 1 <input type = "Text" name = "MyNum1">
  <br>Number 2 <input type = "Text" name = "MyNum2">
  <br><input type = "Submit" name = "See results">
</form>
Mid

Description

Extracts a substring from a string.

Returns

A string: the set of characters from string, beginning at start, of length count.

Category

String functions

Function syntax

Mid(string, start, count)

See also

Left, Len, Right

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>string</td>
<td>A string or a variable that contains one. Must be single-quotation mark or double-quotation mark delimited.</td>
</tr>
<tr>
<td>start</td>
<td>A positive integer or a variable that contains one. Position at which to start count. Positions start with 1, not 0.</td>
</tr>
<tr>
<td>count</td>
<td>A positive integer or a variable that contains one. Number of characters to return. (Zero is not valid, but it does not throw an error.)</td>
</tr>
</tbody>
</table>

Example

<h3>Mid Example</h3>

<cfif IsDefined("Form.myText")>
<!---- If len returns 0 (zero), then show error message. ---->
<cfif Len(Form.myText) LTE Form.RemoveChars>
<cfoutput><p style="color: red; font-weight: bold">Your string #Form.myText# only has #Len(Form.myText)# characters. You cannot output the #Form.removeChars# middle characters of this string because it is not long enough.</p></cfoutput>
<cfelseif Form.startPos GTE Len(Form.myText)>
<cfoutput><p style="color: red; font-weight: bold">Your string #Form.myText# only has #Len(Form.myText)# characters. You cannot start at position #Form.startPos#.</p></cfoutput>
<cfelse>
<cfoutput><p>Your original string: <strong>#Form.myText#</strong></p><p>Your changed string, showing only the <strong>#Mid(Form.myText, Form.startPos, Form.removeChars)#</strong> middle characters: <strong>#Mid(Form.myText, Form.startPos, Form.removeChars)#</strong></p></cfoutput>
</cfif>
</cfif>
<cfelse>
<p style="color: red; font-weight: bold">Please enter a string of more than 0 (zero) characters.</p>

776 Chapter 3: ColdFusion Functions
<form action="<cfoutput>#CGI.ScriptName#</cfoutput>" method="POST">
<p>Type in some text<br />
<input type="Text" name="myText"></p>
<p>Enter a starting position (from the beginning of the entered text)<br />
<input name="startPos" type="text" size="1"></p>
<p>How many characters do you want to show?<br />
<select name="RemoveChars">
<option value="1">1</option>
<option value="3" selected>3</option>
<option value="5">5</option>
<option value="7">7</option>
<option value="9">9</option>
</select>
<input type="Submit" name="Submit" value="Remove characters"></p>
</form>
**Min**

**Description**

Determines the lesser of two numbers.

**Returns**

The lesser of two numbers.

**Category**

Mathematical functions

**Function syntax**

Min(number1, number2)

**See also**

Max

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>number1, number2</td>
<td>Numbers</td>
</tr>
</tbody>
</table>

**Example**

```html
<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>
   <p>The minimum of the two numbers is <cfoutput>$Min(FORM.myNum1, FORM.myNum2)$</cfoutput></p>
   <cfelse>
   <p>Please enter two numbers</p>
   </cfif>
</cfif>

<form action = "min.cfm">
<h3>Enter two numbers, and see the maximum and minimum of the two numbers</h3>
Number 1 <input type = "Text" name = "MyNum1">
<br>Number 2 <input type = "Text" name = "MyNum2">
<br><input type = "Submit" name = "" value = "See results">
</form>
```
Minute

Description
Extracts the minute value from a date/time object.

Returns
The ordinal value of the minute, in the range 0–59.

Category
Date and time functions

Function syntax
Minute(date)

See also
DatePart, Hash, Second

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>date</td>
<td>• A date/time object, in the range 100 AD–9999 AD.</td>
</tr>
</tbody>
</table>

Usage
When passing a date/time value as a string, you must enclose it in quotation marks. Otherwise, it is interpreted as a number representation of a date/time object.

Example
<h3>Minute Example</h3>

<cfoutput>
The time is currently #TimeFormat(Now())#. We are in hour #Hour(Now())#, Minute #Minute(Now())# and Second #Second(Now())# of the day.
</cfoutput>
**Month**

**Description**

Extracts the month value from a date/time object.

**Returns**

The ordinal value of the month, in the range 1 (January) – 12 (December).

**Category**

*Date and time functions*

**Function syntax**

`Month(date)`

**See also**

`DatePart, MonthAsString, Quarter`

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>date</td>
<td>Date/time object, in the range 100 AD–9999 AD.</td>
</tr>
</tbody>
</table>

**Usage**

When passing a date/time value as a string, you must enclose it in quotation marks. Otherwise, it is interpreted as a number representation of a date/time object.

**Note:** You can pass the `CreateDate` function or the `Now` function as the `date` parameter of this function; for example: `#Month(CreateDate(2001, 3, 3))#`.

**Example**

```cfscript
<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>This is day #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)#
    <br>(day #DayOfWeek(yourDate)# of #DaysInYear(yourDate)#). <br>
    <cfif IsLeapYear(Year(yourDate))#This is a leap year
    <cfelse>This is not a leap year
  </cfoutput>
</cfif>
</cfoutput>
</cfif>
```
MonthAsString

Description
Determines the name of the month that corresponds to month_number.

Returns
A string; the name of the specified month, in the current locale.

Category
Date and time functions, String functions

Function syntax
MonthAsString(month_number)

See also
DatePart, Month, Quarter

Parameters
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>month_number</td>
<td>An integer in the range 1 - 12.</td>
</tr>
</tbody>
</table>

Example
<h3>MonthAsString Example</h3>
<cfif IsDefined("FORM.year")>
<p>More information about your date:
<cfset yourDate = CreateDate(FORM.year, FORM.month, FORM.day)>

<cfoutput>
<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(Years(yourDate))>This is a leap year
  <cfelse>This is not a leap year
</cfif>
</cfoutput>
</cfif>
**Now**

**Description**

Gets the current date and time of the computer running the ColdFusion server. The return value can be passed as a parameter to date functions such as `DaysInYear` or `FirstDayOfMonth`.

**Returns**

A date/time object; the current date and time of the computer running the ColdFusion server.

**Category**

Date and time functions

**Function syntax**

```coldfusion
Now()
```

**See also**

`CreateDateTime`, `DatePart`

**Example**

```coldfusion
<h3>Now Example</h3>
<p>Now returns the current date and time as a valid date/time object.</p>
<p>The current date/time value is <cfoutput>#Now()#</cfoutput></p>
<p>You can also represent this as <cfoutput>#DateFormat(Now())#, #TimeFormat(Now())#</cfoutput>
```
**NumberFormat**

**Description**

Creates a custom-formatted number value. Supports the numeric formatting used in the U.S. For international number formatting, see `LSNumberFormat`.

**Returns**

A formatted number value:

- If no mask is specified, returns the value as an integer with a thousands separator.
- If the parameter value is "" (an empty string), returns 0.

**Category**

Display and formatting functions

**Function syntax**

```
NumberFormat(number [, mask ])
```

**See also**

`DecimalFormat`, `DollarFormat`, `IsNumeric`, `LSNumberFormat`

**History**

ColdFusion MX: Changed behavior: if the mask format cannot correctly mask a number, this function returns the number unchanged. (It does not truncate the number nor throw an error.) (If no mask is selected, ColdFusion MX rounds the decimal part as ColdFusion 5 does. For example, it rounds 34.567 to 35.)

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>number</td>
<td>A number.</td>
</tr>
<tr>
<td>mask</td>
<td>A string or a variable that contains one. Set of characters that determine how ColdFusion displays the number</td>
</tr>
</tbody>
</table>

The following table explains mask characters:

<table>
<thead>
<tr>
<th>Mask character</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>_ (underscore)</td>
<td>Optional. Digit placeholder. (Shows decimal places more clearly than 9)</td>
</tr>
<tr>
<td>9</td>
<td>Optional. Digit placeholder.</td>
</tr>
<tr>
<td>.</td>
<td>Location of a mandatory decimal point.</td>
</tr>
<tr>
<td>0</td>
<td>Located to the left or right of a mandatory decimal point. Pads with zeros.</td>
</tr>
<tr>
<td>()</td>
<td>If number is less than zero, puts parentheses around the mask.</td>
</tr>
<tr>
<td>+</td>
<td>Puts plus sign before positive number; minus sign before negative number.</td>
</tr>
<tr>
<td>-</td>
<td>Puts a space before positive number; minus sign before negative number.</td>
</tr>
</tbody>
</table>
### Note:
If you do not specify a sign for the mask, positive and negative numbers do not align in columns. To put a plus sign or space before positive numbers and a minus sign before negative numbers, use the plus or minus sign, respectively.

### Usage
This function uses Java standard locale formatting rules on all platforms.

The position of symbols in format masks determines where the codes take effect. For example, if you put a dollar sign at the far left of a format mask, ColdFusion displays a dollar sign at the left edge of the formatted number. If you separate the dollar sign on the left edge of the format mask by at least one underscore, ColdFusion displays the dollar sign just to the left of the digits in the formatted number.

These examples show how symbols determine formats:

<table>
<thead>
<tr>
<th>Number</th>
<th>Mask</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.37</td>
<td>$______</td>
<td>&quot;$ 4.37&quot;</td>
</tr>
<tr>
<td>4.37</td>
<td><em>$</em>_____</td>
<td>&quot;$4.37&quot;</td>
</tr>
</tbody>
</table>

The positioning can also show where to place the minus sign for negative numbers:

<table>
<thead>
<tr>
<th>Number</th>
<th>Mask</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>-4.37</td>
<td>________</td>
<td>&quot; -4.37&quot;</td>
</tr>
<tr>
<td>-4.37</td>
<td>________</td>
<td>&quot;-4.37&quot;</td>
</tr>
</tbody>
</table>

The positions for a symbol are: far left, near left, near right, and far right. The left and right positions are determined by the side of the decimal point on which the code character is shown. For formats that do not have a fixed number of decimal places, you can use a caret (^) to separate the left fields from the right.

An underscore determines whether the code is placed in the far or near position. Most code characters’ effect is determined by the field in which they are located. This example shows how to specify where to put parentheses to display negative numbers:

<table>
<thead>
<tr>
<th>Number</th>
<th>Mask</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.21</td>
<td>C(<strong>^</strong>)</td>
<td>&quot;(3.21)&quot;</td>
</tr>
<tr>
<td>3.21</td>
<td>C__(^__)</td>
<td>&quot;(3.21)&quot;</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Mask character</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>.</td>
<td>Separates every third decimal place with a comma.</td>
</tr>
<tr>
<td>L,C</td>
<td>Left-justifies or center-justifies number within width of mask column. First character of mask must be L or C. The default value is right-justified.</td>
</tr>
<tr>
<td>$</td>
<td>Puts a dollar sign before formatted number. First character of mask must be the dollar sign ($).</td>
</tr>
<tr>
<td>^</td>
<td>Separates left and right formatting.</td>
</tr>
</tbody>
</table>

### Mask character Meaning

- **.**: Separates every third decimal place with a comma.
- **L,C**: Left-justifies or center-justifies number within width of mask column. First character of mask must be L or C. The default value is right-justified.
- **$**: Puts a dollar sign before formatted number. First character of mask must be the dollar sign ($).
- **^**: Separates left and right formatting.
When converting from string to double, to prevent rounding errors, this function adds a rounding factor of 1.5543122344752E-014 to the converted number. For example, without adding the rounding factor, converting the string value 1.275 to double with two digits of precision results in a value of 1.27499999999999999, which would be rounded up to 1.27. By adding the rounding factor, the conversion correctly results in a value of 1.28.

If you round off a double such as 1.99499999999999999999999999999, where the last decimal is 10E-14, the rounding factor can cause an incorrect result.

To set the default display format of date, time, number, and currency values, use the SetLocale function.

**Example**

```cftags
<cfloop FROM = 1000 TO = 1020 INDEX = "counter">
<cfset CounterRoot2 = counter * sqr(2)>

<!--- Show result in default format, adding comma for thousands place; and in custom format, displaying to two decimal places --->
<cfoutput>
<pre>#counter# * Square Root of 2: #NumberFormat(CounterRoot2, "_____.__")#</pre>
<pre>#counter# * Square Root of 2: #NumberFormat(CounterRoot2)#</pre>
</cfoutput>
</cfloop>
```

<table>
<thead>
<tr>
<th>Number</th>
<th>Mask</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.21</td>
<td>C(_)__</td>
<td>&quot;(3.21)&quot;</td>
</tr>
<tr>
<td>3.21</td>
<td>C__(^)__</td>
<td>&quot;(3.21)&quot;</td>
</tr>
</tbody>
</table>
ParagraphFormat

Description
Replaces characters in a string:

• Single newline characters (CR/LF sequences) with spaces
• Double newline characters with HTML paragraph tags (&lt;p&gt;)

Returns
A copy of the string, with characters converted.

Category
Display and formatting functions, String functions

Function syntax
ParagraphFormat(string)

See also
StripCR

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>string</td>
<td>A string or a variable that contains one</td>
</tr>
</tbody>
</table>

Usage
This function is useful for displaying data entered in textarea fields.

Example
<br>
<cfif IsDefined("FORM.myTextArea")>
<p>Your text area, formatted</p>
<p>&lt;cfoutput>#ParagraphFormat(FORM.myTextArea)#&lt;/cfoutput&gt;</p>
</cfif>

<!--- use #Chr(10)#&Chr(13)# to simulate a line feed/carriage return combination; i.e., a return --->
<form action = "paragraphformat.cfm"
<textArea name = "MyTextArea" cols = "35" ROWS = "8"
This is sample text and you see how it scrolls
&lt;cfoutput>#Chr(10)#&Chr(13)#&lt;/cfoutput&gt;
from one line
<cfoutput>#Chr(10)#&Chr(13)#&Chr(10)#&Chr(13)#&lt;/cfoutput&gt;
to the next
</textarea>
<input type = "Submit" name = "Show me the HTML version"
</form>
ParameterExists

Description

This function is deprecated. Use the IsDefined function.

Determines whether a parameter exists. ColdFusion does not evaluate the argument.

History

ColdFusion MX: Deprecated this function. It might not work, and might cause an error, in later releases.
ParseDateTime

Description

 Parses a date/time string according to the English (U.S.) locale conventions. (To format a date/time string for other locales, use the LSParseDateTime function.)

Returns

 A date/time object

Category

 Date and time functions, Display and formatting functions

Function syntax

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

See also

 IsDate, IsNumericDate, SetLocale

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>date/time string</td>
<td>A string containing a date/time value formatted according to U.S. locale conventions. Can represent a date/time in the range 100 AD–9999 AD. Years 0–29 are interpreted as 2000–2029; years 30–99 are interpreted as 1930–1999.</td>
</tr>
<tr>
<td>pop-conversion</td>
<td>• pop: specifies that the date/time string is in POP format, which includes the local time of the sender and a time-zone offset from UTC. ColdFusion applies the offset and returns a value with the UTC time. • standard: (the default) function does no conversion.</td>
</tr>
</tbody>
</table>

Usage

 This function is similar to CreateDateTime, but it takes a string instead of enumerated date/time values. These functions are provided primarily to increase the readability of code in compound expressions.

 To calculate a difference between time zones, use the GetTimeZoneInfo function.

 To set the default display format of date, time, number, and currency values, use the SetLocale function.

Example

 <h3>ParseDateTime Example</h3>
 <cfif IsDefined("form.theTestValue")>
   <cfif IsDate(form.theTestValue)>
     <h3>The expression <cfoutput>#DE(form.theTestValue)#</cfoutput> is a valid date</h3>
   </cfif>
   <p>The parsed date/time is:
     <cfoutput>#ParseDateTime(form.theTestValue)#</cfoutput>
   </cfelse>
<h3>The expression <cfoutput>#DE(form.theTestValue)#</cfoutput> is not a valid date</h3>
</cfif>
</cfif>
<form action="#CGI.ScriptName#" method="POST">
<p>Enter an expression, and discover if it can be evaluated to a date value.
<input type="Text" name="TheTestValue" value="<CFOUTPUT>#DateFormat(Now())#<br>/DateFormat(Now())#</CFOUTPUT>">
<input type="Submit" value="Parse the Date" name=""/>
</form>
Pi

Description
Gets the mathematical constant π, accurate to 15 digits.

Returns
The number 3.14159265358979.

Category
Mathematical functions

Function syntax
Pi()

See also
ASin, Cos, Sin, Tan

Example
<h3>Pi Example</h3>
<!--- By default, ColdFusion displays only 11 significant digits. Use NumberFormat to display all 15. --->
The Pi function Returns the number
<coutput>
#NumberFormat(Pi(), "_._______________")#
</coutput> the mathematical constant pi, accurate to 15 digits.
PreserveSingleQuotes

Description
Prevents ColdFusion from automatically escaping single-quotation mark characters that are contained in a variable. ColdFusion does not evaluate the argument.

Returns
(None)

Category
Other functions

Function syntax
PreserveSingleQuotes(variable)

History
ColdFusion MX: Changed behavior: ColdFusion automatically escapes simple-variable, array-variable, and structure-variable references within a cfquery tag body or block. (Earlier releases did not automatically escape array-variable references.)

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>variable</td>
<td>Variable that contains a string in which to preserve single-quotation marks.</td>
</tr>
</tbody>
</table>

Usage
This function is useful in SQL statements to defer evaluation of a variable reference until runtime. This prevents errors that result from the evaluation of a single-quote or apostrophe data character (for example, "Joe's Diner") as a delimiter.

Example A: Consider this code:
<cfset mystring = "'Newton's Law', 'Fermat's Theorem'">
PreserveSingleQuotes(#mystring#) is
<cfoutput>
    #PreserveSingleQuotes(mystring)#
</cfoutput>
The output is as follows:
PreserveSingleQuotes(#mystring#) is 'Newton's Law', 'Fermat's Theorem'

Example B: Consider this code:
<cfset list0 = "'1','2','3' ">
<cfquery sql = "select * from foo where bar in (#list0#)"
ColdFusion escapes the single-quote characters in the list as follows:
"*1**", "*2**", "*3**"
The cfquery tag throws an error.
You code this function correctly as follows:

```coldfusion
cfquery sql = "select * from foo where bar in (#preserveSingleQuotes(list0)#)"
```

This function ensures that ColdFusion evaluates the code as follows:

'1', '2', '3'

Example

```coldfusion
<h3>PreserveSingleQuotes Example</h3>
<p>This is a useful function for creating lists of information to return from a query. In this example, we pick the list of Centers in Suisun, San Francisco, and San Diego, using the SQL grammar IN to modify a WHERE clause, rather than looping through the result set after the query is run.
</p>
cfset List = "'Suisun', 'San Francisco', 'San Diego'"
<cfquery name = "GetCenters" datasource = "cfdocexamples">
  SELECT Name, Address1, Address2, City, Phone
  FROM Centers
  WHERE City IN (#PreserveSingleQuotes(List)#)
</cfquery>
<p>We found #GetCenters.RecordCount# records.
</p>
<cfoutput query = "GetCenters">
  #Name#
  #Address1#
  #Address2#
  #City#
  #Phone#
</cfoutput>
```
Quarter

Description
Calculates the quarter of the year in which a date falls.

Returns
An integer, 1–4.

Category
Date and time functions

Function syntax
Quarter(date)

See also
DatePart, Month

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>date</td>
<td>A date/time object in the range 100 AD–9999 AD.</td>
</tr>
</tbody>
</table>

Usage
When passing a date/time value as a string, you must enclose it in quotation marks. Otherwise, it is interpreted as a number representation of a date/time object.

Example
<h3>Quarter Example</h3>

Today, <cfoutput>#DateFormat(Now())#</cfoutput>, is in Quarter <cfoutput>#{Quarter(Now())}#</cfoutput>.
QueryAddColumn

Description
Adds a column to a query and populates its rows with the contents of a one-dimensional array. Pads query columns, if necessary, to ensure that all columns have the same number of rows.

Returns
The number of the column that was added.

Category
Query functions

Function syntax
QueryAddColumn(query, column-name[, datatype], array-name)

See also
QueryNew, QueryAddRow, QuerySetCell; “Managing data types for columns” in Chapter 22, “Using Query of Queries,” in ColdFusion MX Developer’s Guide

History
ColdFusion MX 7: Added the datatype parameter.
ColdFusion MX: Changed behavior: if a user attempts to add a column whose name is invalid, ColdFusion throws an error. (In earlier releases, ColdFusion permitted the add operation, but the user could not reference the column after adding it.)

Parameters
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>query</td>
<td>Name of a query object.</td>
</tr>
<tr>
<td>column-name</td>
<td>Name of the new column.</td>
</tr>
</tbody>
</table>
| datatype  | (Optional) Column data type. ColdFusion generates an error if data you add to the column is not of this type, or if it cannot convert the data to this type. The following data types are valid:  
  • Integer: 32-bit integer  
  • BigInt: 64-bit integer  
  • Double: 64-bit decimal number  
  • Decimal: Variable length decimal, as specified by java.math.BigDecimal  
  • VarChar: String  
  • Binary: Byte array  
  • Bit: Boolean (1=True, 0=False)  
  • Time: Time  
  • Data: Date (can include time information) |
| array-name | Name of an array whose elements populate the new column. |
Usage
You can add columns to query objects, such as queries retrieved with the `cfquery` tag or queries created with the `queryNew` function. You cannot use the `QueryAddColumn` function on a cached query. This function is useful for generating a query object from the arrays of output parameters that Oracle stored procedures can generate.

Macromedia recommends that you use the optional `datatype` parameter. Without this parameter, ColdFusion must try to determine the column’s data type when it uses the query object in a query of queries. Determining the data type requires additional processing, and can result in errors if ColdFusion does not guess the type correctly.

Example
The following example creates a new query object, uses the `QueryAddColumn` function to add three columns to the object, and displays the results. Because two of the arrays that provide the data are shorter than the third, `QueryAddColumn` adds padding to the corresponding columns in the query.

```coldfusion
<!--- Make a query. --->
<cfset myQuery = QueryNew('**')>

<!--- Create an array. --->
<cfset FastFoodArray = ArrayNew(1)>
<cfset FastFoodArray[1] = "French Fries">
<cfset FastFoodArray[2] = "Hot Dogs">
<cfset FastFoodArray[3] = "Fried Clams">
<cfset FastFoodArray[4] = "Thick Shakes">
<!--- Use the array to add a column to the query. --->
<cfset nColumnNumber = QueryAddColumn(myQuery, "FastFood", "VarChar", FastFoodArray)>

<!--- Create a second array. --->
<cfset FineCuisineArray = ArrayNew(1)>
<cfset FineCuisineArray[1] = "Lobster">
<cfset FineCuisineArray[2] = "Flambe">
<!--- Use the array to add a second column to the query. --->
<cfset nColumnNumber2 = QueryAddColumn(myQuery, "FineCuisine", "VarChar", FineCuisineArray)>

<!--- Create a third array. --->
<cfset HealthFoodArray = ArrayNew(1)>
<cfset HealthFoodArray[1] = "Bean Curd">
<cfset HealthFoodArray[2] = "Yogurt">
<cfset HealthFoodArray[3] = "Tofu">
<!--- Use the array to add a third column to the query. --->
<cfset nColumnNumber3 = QueryAddColumn(myQuery, "HealthFood", "VarChar", HealthFoodArray)>

<!--- Display the results. --->
<table cellspacing = "2" cellpadding = "2" border = "0">
<tr>
  <th align = "left">Fast Food</th>
  <th align = "left">Fine Cuisine</th>
</tr>
```


<th align="left">Health Food</th>
</tr>
<cfoutput query="myQuery">
<tr>
<td><%=FastFood%></td>
<td><%=FineCuisine%></td>
<td><%=HealthFood%></td>
</tr>
</cfoutput>
</table>
QueryAddRow

**Description**

Adds a specified number of empty rows to a query.

**Returns**

The number of rows in the query

**Category**

Query functions

**Function syntax**

```
QueryAddRow(query [, number ])
```

**See also**

QueryAddColumn, QueryAddRow, QuerySetCell, QueryNew; “Creating a record set with a function” in Chapter 22, “Using Query of Queries,” in *ColdFusion MX Developer’s Guide*

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>query</td>
<td>Name of an executed query.</td>
</tr>
<tr>
<td>number</td>
<td>Number of rows to add to the query. The default value is 1.</td>
</tr>
</tbody>
</table>

**Example**

```cfc
cfquery name = "GetCourses" datasource = "cfdocexamples"
  SELECT Course_ID, Descript
  FROM Courses
</cfquery>
<p>The Query "GetCourses" has <cfoutput>#GetCourses.RecordCount#</cfoutput> rows.</p>
<cfset CountVar = 0>
<cfloop CONDITION = "CountVar LT 15">
  <cfset temp = QueryAddRow(GetCourses)>
  <cfset CountVar = CountVar + 1>
  <cfset Temp = QuerySetCell(GetCourses, "Number", 100*CountVar)>
  <cfset CountVar = CountVar + 1>
  <cfset Temp = QuerySetCell(GetCourses, "Descript", "Description of variable #CountVar#")>
</cfloop>
<p>After the QueryAddRow action, the query has <cfoutput>#GetCourses.RecordCount#</cfoutput> records.</p>
```

---

QueryAddRow  797
QueryNew

Description

Creates an empty query (query object).

Returns

An empty query with a set of named columns, or an empty query.

Category

Query functions

Function syntax

QueryNew(columnlist [, columntypelist])

History

ColdFusion MX 7: Added columntypelist parameter.

See also

QueryAddColumn, QueryAddRow, QuerySetCell; “Managing data types for columns” in Chapter 22, “Using Query of Queries,” ColdFusion MX Developer’s Guide

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>columnlist</td>
<td>Comma-delimited list of column names, or an empty string.</td>
</tr>
<tr>
<td>columntypelist</td>
<td>(Optional) Comma-delimited list specifying column data types. ColdFusion generates an error if the data you add to the column is not of this type, or if it cannot convert the data to this type. The following data types are valid:</td>
</tr>
<tr>
<td></td>
<td>• Integer: 32-bit integer</td>
</tr>
<tr>
<td></td>
<td>• BigInt: 64-bit integer</td>
</tr>
<tr>
<td></td>
<td>• Double: 64-bit decimal number</td>
</tr>
<tr>
<td></td>
<td>• Decimal: Variable length decimal, as specified by java.math.BigDecimal</td>
</tr>
<tr>
<td></td>
<td>• VarChar: String</td>
</tr>
<tr>
<td></td>
<td>• Binary: Byte array</td>
</tr>
<tr>
<td></td>
<td>• Bit: Boolean (1=True, 0=False)</td>
</tr>
<tr>
<td></td>
<td>• Time: Time</td>
</tr>
<tr>
<td></td>
<td>• Date: Date (can include time information)</td>
</tr>
</tbody>
</table>

Usage

If you specify an empty string in the columnlist parameter, you must use the QueryAddColumn function to add columns to the query.

Macromedia recommends that you use the optional columntypelist parameter. Without this parameter, ColdFusion must try to determine data types when it uses the query object in a query of queries. Determining data types requires additional processing, and can result in errors if ColdFusion does not guess a type correctly.
Example

The following example uses the `QueryNew` function to create an empty query with three columns. It populates two rows of the query and displays the contents of the query object and its metadata.

```coldfusion
<!--- Create a new three-column query, specifying the column data types --->
<cfset myQuery = QueryNew(“Name, Time, Advanced”, “VarChar, Time, Bit”)>

<!--- Make two rows in the query --->
<cfset newRow = QueryAddRow(myQuery, 2)>

<!--- Set the values of the cells in the query --->
<cfset temp = QuerySetCell(myQuery, “Name”, “The Wonderful World of CMFL”, 1)>
<cfset temp = QuerySetCell(myQuery, “Time”, “9:15 AM”, 1)>
<cfset temp = QuerySetCell(myQuery, “Advanced”, False, 1)>
<cfset temp = QuerySetCell(myQuery, “Name”, ”CFCs for Enterprise Applications”, 2)>
<cfset temp = QuerySetCell(myQuery, “Time”, ”12:15 PM”, 2)>
<cfset temp = QuerySetCell(myQuery, “Advanced”, True, 2)>

<h4>The query object contents</h4>
<cfoutput query = “myQuery”>
  #Name# #Time# #Advanced#<br>
</cfoutput><br>

<h4>Using individual query data values</h4>
<cfoutput>
  #MyQuery.name[2]# is at #MyQuery.Time[2]#<br>
</cfoutput><br>

<h4>The query metadata</h4>
<cfset querymetadata=getMetaData(myQuery)>
<cfdump var=”#querymetadata#”>
```
QuerySetCell

Description
Sets a cell to a value. If no row number is specified, the cell on the last row is set.

Returns
True, if successful; False, otherwise.

Category
Query functions

Function syntax
QuerySetCell(query, column_name, value [, row_number ])

See also
QueryAddColumn, QueryAddRow, QueryNew; “Creating a record set with a function” in Chapter 22, “Using Query of Queries,” in ColdFusion MX Developer's Guide

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>query</td>
<td>Name of an executed query.</td>
</tr>
<tr>
<td>column_name</td>
<td>Name of a column in the query.</td>
</tr>
<tr>
<td>value</td>
<td>Value to set in the cell.</td>
</tr>
<tr>
<td>row_number</td>
<td>Row number. The default value is last row.</td>
</tr>
</tbody>
</table>

Example
<!---- This example shows the use of QueryAddRow and QuerySetCell --->

<!---- start by making a query --->
<cfquery name = "GetCourses" datasource = "cfdocexamples">
SELECT Course_ID, Descript
FROM Courses</cfquery>
<p>The Query "GetCourses" has <cfoutput>#GetCourses.RecordCount#</cfoutput> rows.</p>
<cfset CountVar = 0>
<cfloop CONDITION = "CountVar LT 15">
<cfset temp = QueryAddRow(GetCourses)>
<cfset CountVar = CountVar + 1>
<cfset Temp = QuerySetCell(GetCourses, "Number", 100*CountVar)>
<cfset CountVar = CountVar + 1>
<cfset Temp = QuerySetCell(GetCourses, "Descript", "Description of variable #CountVar#")>
</cfloop>
<p>After the QueryAddRow action, the query has <cfoutput>#GetCourses.RecordCount#</cfoutput> records.</p>
<cfoutput query="GetCourses">
<pre>#Course_ID# #Course_Number# #Descript#</pre></cfoutput>
QuotedValueList

Description

Gets the values of each record returned from an executed query. ColdFusion does not evaluate the arguments.

Returns

A delimited list of the values of each record returned from an executed query. Each value is enclosed in single-quotation marks.

Category

Query functions, List functions

Function syntax

QuotedValueList(query.column [, delimiter ])

See also

ValueList

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>query.column</td>
<td>Name of an executed query and column. Separate query name and column name with a period.</td>
</tr>
<tr>
<td>delimiter</td>
<td>A string or a variable that contains one character(s) that separate column data.</td>
</tr>
</tbody>
</table>

Example

<!--- use the contents of one query to create another dynamically --->
<cfset List = "'BIOL', 'CHEM'">
<!--- first, get the department IDs in our list --->
<cfquery name = "GetDepartments" datasource = "cfdocexamples">
SELECT Dept_ID FROM Departments
WHERE Dept_ID IN (#PreserveSingleQuotes(List)#)
</cfquery>

<!--- now, select the courses for that department based on the quotedValueList produced from our previous query --->
<cfquery name = "GetCourseList" datasource = "cfdocexamples">
SELECT *
FROM CourseList
WHERE Dept_ID IN ('#GetDepartments.Dept_ID#')
</cfquery>

<!--- now, output the results --->

List the course numbers that are in BIOL and CHEM (uses semicolon (;) as the delimiter):
<cfoutput>
#QuotedValueList(GetCourseList.CorNumber,";")#<br>
</cfoutput>
Rand

Description
Generates a pseudo-random number.

Returns
A pseudo-random decimal number, in the range 0 – 1.

Category
Mathematical functions, Security functions

Function syntax
Rand([algorithm])

History
ColdFusion MX 7: Added the algorithm parameter.

See also
Randomize, RandRange

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>algorithm</td>
<td>(Optional) The algorithm to use to generated the random number. ColdFusion MX installs a cryptography library with the following algorithms:</td>
</tr>
<tr>
<td></td>
<td>• CFMX_COMPAT: the algorithm used in ColdFusion MX (default).</td>
</tr>
<tr>
<td></td>
<td>• SHA1PRNG: generates a number using the Sun Java SHA1PRNG algorithm. This algorithm provides greater randomness than the default algorithm</td>
</tr>
<tr>
<td></td>
<td>• IBMSecureRandom: for IBM WebSphere (IBM JVM does not support the SHA1PRNG algorithm).</td>
</tr>
</tbody>
</table>

Usage
Call the Randomize function before calling this function to seed the random number generator. Seeding the generator ensures that the Rand function always generates the same sequence of pseudo-random numbers. This behavior is useful if you must reproduce a pattern consistently.

ColdFusion MX 7 uses the Java Cryptography Extension (JCE) and installs a Sun Java 1.4.2 runtime that includes the Sun JCE default security provider. This provider includes the algorithms listed in the Parameters section (except the default algorithm). The JCE framework includes facilities for using other provider implementations; however, Macromedia cannot provide technical support for third-party security providers.

Example
The following example uses the SHA1PRNG algorithm to generate a single random number:

```cfml
<h3>Rand Example</h3>
<cfoutput>
<p>Rand("SHA1PRNG") returned: #Rand("SHA1PRNG")#</p>
<p><A HREF = "#CGI.SCRIPT_NAME#">Try again</A></p>
</cfoutput>
```


**Randomize**

**Description**

Seeds the pseudo-random number generator with an integer number, ensuring repeatable number patterns.

**Returns**

A pseudo-random decimal number, in the range 0–1.

**Category**

Mathematical functions, Security functions

**Function syntax**

```
Randomize(number[, algorithm])
```

**History**

ColdFusion MX 7: Added the algorithm parameter.

**See also**

Rand, RandRange

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>number</td>
<td>Integer number. If the number is not in the range -2,147,483,648 – 2,147,483,647, ColdFusion generates an error.</td>
</tr>
</tbody>
</table>
| algorithm | (Optional) The algorithm to use to generate the seed number. ColdFusion MX installs a cryptography library with the following algorithms:  
  - CFMX_COMPAT: the algorithm used in ColdFusion MX (default).  
  - SHA1PRNG: generates a number using the Sun Java SHA1PRNG algorithm. This algorithm provides greater randomness than the default algorithm.  
  - IBMSecureRandom: for IBM WebSphere (IBM JVM does not support the SHA1PRNG algorithm). |

**Usage**

Call this function before calling Rand to seed the random number generator. Seeding the generator ensures that the Rand function always generates the same sequence of pseudo-random numbers. This behavior is useful if you must reproduce a pattern consistently.

ColdFusion MX 7 uses the Java Cryptography Extension (JCE) and installs a Sun Java 1.4.2 runtime that includes the Sun JCE default security provider. This provider includes the algorithms listed in the Parameters section (except the default algorithm). The JCE framework includes facilities for using other provider implementations; however, Macromedia cannot provide technical support for third-party security providers.
Example

The following example calls the Randomize function to seed the random number generator and generates 10 random numbers. To show the effect of the seed, submit the form with the same value multiple times.

<h3>Randomize Example</h3>

<!--- Do the following only if the form has been submitted. --->
<cfif IsDefined("Form.myRandomInt")>

<!--- Make sure submitted value is a number and display its value. --->
<cfif IsNumeric(FORM.myRandomInt)>
<cfoutput>
  <b>Seed value is #FORM.myRandomInt#</b><br>
</cfoutput>
</cfif>

<!--- Call Randomize to seed the random number generator. --->
<cfset r = Randomize(FORM.myRandomInt, "SHA1PRNG")>

<cfoutput>
  <b>Random number returned by Randomize(#For.myRandomInt#, "SHA1PRNG"): #r#</b><br>
  10 random numbers generated using the SHA1PRNG algorithm:<br>
  <cfloop index = "i" from = "1" to = "10" step = "1">
    #Rand("SHA1PRNG")#
  </cfloop>
</cfoutput>

<cfelse>
  <p>Please enter a number.</p>
</cfif>

<!--- Form to specify the seed value. --->
<form action="#CGI.SCRIPT_NAME#" method="post">
  Enter a number to seed the randomizer:<br>
  <input type = "Text" name = "MyRandomInt" value="12345">
  <p><input type = "Submit" name = ""></p>
</form>
RandRange

Description
Generates a pseudo-random integer in the range between two specified numbers.

Returns
A pseudo-random integer.

Category
Mathematical functions, Security functions

Function syntax
RandRange(number1, number2[, algorithm])

History
ColdFusion MX 7: Added the algorithm parameter.

See also
Rand, Randomize

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>number1, number2</td>
<td>Integer numbers. If the numbers are not in the range -2,147,483,648 – 2,147,483,647, ColdFusion generates an error.</td>
</tr>
</tbody>
</table>
| algorithm | (Optional) The algorithm to use to generated the random number. ColdFusion MX installs a cryptography library with the following algorithms:
• CFMX_COMPAT: the algorithm used in ColdFusion MX (default).
• SHA1PRNG: generates a number using the Sun Java SHA1PRNG algorithm. This algorithm provides greater randomness than the default algorithm
• IBMSecureRandom: for IBM WebSphere (IBM JVM does not support the SHA1PRNG algorithm.) |

Usage
Very large positive or negative values for the number1 and number2 parameters might result in poor randomness in the results. To prevent this problem, do not specify numbers outside the range -1,000,000,000 – 1,000,000,000.

ColdFusion MX 7 uses the Java Cryptography Extension (JCE) and installs a Sun Java 1.4.2 runtime that includes the Sun JCE default security provider. This provider includes the algorithms listed in the Parameters section (except the default algorithm). The JCE framework includes facilities for using other provider implementations; however, Macromedia cannot provide technical support for third-party security providers.
Example

The following example contains a form that requires random number range values, and lets you optionally specify a random number seed value. It uses cfform controls and attributes to specify a default range, ensure that the range fields have values, and validate that the field values are in a specified integer range. When you submit the form, it checks whether the seed field has an empty string; if the field has a value, the code uses the number to seed the random number generator. It then generates and displays the random number.

<h3>RandRange Example</h3>

<!--- Do the following only if the form has been submitted. --->
<cfif IsDefined("Form.mySeed")>

<!--- Do the following only if the seed field has a non-empty string. --->
<cfif Form.mySeed NEQ ">
  <cfoutput>
    <b>Seed value is #FORM.mySeed#</b>
  </cfoutput>
</cfif>

<!--- Call Randomize to seed the random number generator. --->
<cfset r = Randomize(FORM.mySeed, "SHA1PRNG")>

<cfelse>
  <b>No Seed value submitted</b>
</cfif>

<!--- Generate and display the random number. --->
<cfoutput>
  RandRange returned: #RandRange(FORM.myInt, FORM.myInt2, "SHA1PRNG")#
</cfoutput>
</cfif>

<!--- This form uses cfform input validation to check the input range. --->
<cfform action = "#CGI.SCRIPT_NAME#">
  <p>Enter the random number Range: From</p>
  <cfinput type = "Text" name = "MyInt" value = "1"
    RANGE = "-1000000000,1000000000"
    message = "Please enter a value between -1,000,000,000 and 1,000,000,000"
    validate = "integer" required = "Yes">
    To
  <cfinput type = "Text" name = "MyInt2" value = "9999"
    RANGE = "-1000000000,1000000000"
    message = "Please enter a value between -1,000,000,000 and 1,000,000,000"
    validate = "integer" required = "Yes">
  </p>
  <p>Enter a number to seed the randomizer:</p>
  <cfinput type = "Text" name = "mySeed" RANGE = "-1000000000,1000000000"
    message = "Please enter a value between -1,000,000,000 and 1,000,000,000"
    validate = "integer" required = "No"></p>
  <p><input type = "Submit" name = ""></p>
</cfform>
REFind

Description

Uses a regular expression (RE) to search a string for a pattern. The search is case sensitive.

For more information on regular expressions, including escape sequences, anchors, and modifiers, see Chapter 7, “Using Regular Expressions in Functions,” in ColdFusion MX Developer’s Guide.

Returns

Depends on the value of the returnsubexpressions parameter:

- If returnsubexpressions = “False”:
  - The position in the string where the match begins
  - 0, if the regular expression is not matched in the string
- If returnsubexpressions = “True”: a structure that contains two arrays, len and pos. The array elements are as follows:
  - If the regular expression is found in the string, the first element of the len and pos arrays contains the length and position, respectively, of the first match of the entire regular expression.
    - If the regular expression contains parentheses that group subexpressions, each subsequent array element contains the length and position, respectively, of the first occurrence of each group.
  - If the regular expression is not found in the string, the first element of the len and pos arrays contains 0.

Category

String functions

Function syntax

REFind(reg_expression, string [, start] [, returnsubexpressions ])

See also

Find, FindNoCase, REFindNoCase, REReplace, REReplaceNoCase

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>reg_expression</td>
<td>Regular expression for which to search. Case-sensitive.</td>
</tr>
<tr>
<td>string</td>
<td>A string, or a variable that contains one, in which to search.</td>
</tr>
</tbody>
</table>
Usage

This function finds the first occurrence of a regular expression in a string. To find the second and subsequent instances of the expression or of subexpressions in it, you call this function more than once, each time with a different start position. To determine the next start position, use the `returnsSubexpressions` parameter, and add the value returned in the first element of the length array to the value in the first element of the position array.

Example

This example shows the use of the `REFind` function with and without the `returnsSubexpressions` parameter set to True.

If you do not use the `returnsSubexpressions` 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.

```cfsql
<cfset teststring = "The cat in the hat hat came back!">
<p>Set startPos to one; returnMatchedSubexpressions = TRUE ---->
<hr size = "2" color = "#0000A0">
<p>If you use the `returnsSubexpressions` parameter, `REFind` returns the position and length of the first occurrence of a regular expression in a string starting from the specified position. The position and length variables are stored in a structure. To access position and length information, use the keys `<i>pos</i>` and `<i>len</i>`, respectively.</p>
<p>The string in which the function is to search is:<br>teststring</p>
```
The first call to REFind to search this string is:
\[\texttt{REFind("[A-Za-z]+",testString,1,"TRUE")}\]

This function returns a structure that contains two arrays: pos and len.

To create this structure you can use a CFSET statement, for example:
\[\texttt{CFSET st = REFind("[[[:alpha:]]]+",testString,1,"TRUE")}\]
\[\texttt{CFSET st = REFind("[[[:alpha:]]]+",testString,1,"TRUE")}\]

The number of elements in each array: \#ArrayLen(st.pos)#.

The number of elements in the pos and len arrays is always one if you do not use parentheses in the regular expression.

The value of st.pos[1] is: \texttt{#st.pos[1]#}.

The value of st.len[1] is: \texttt{#st.len[1]#}.

Substring is \texttt{[#Mid(testString,st.pos[1],st.len[1])#]}.

However, if you use parentheses in the regular expression, the first element contains the position and length of the first instance of the whole expression. The position and length of the first instance of each parenthesized subexpression within is included in additional array elements.

For example:
\[\texttt{CFSET st1 = REFind("([[:alpha:]]+)\([[:space:]]\1\)\),testString,1,"TRUE")}\]
\[\texttt{CFSET st1 = REFind("([[:alpha:]]+)\([[:space:]]\1\)\),testString,1,"TRUE")}\]

The number of elements in each array is \texttt{#ArrayLen(st1.pos)#}.

First whole expression match; position is \texttt{#st1.pos[1]#}; length is \texttt{#st1.len[1]#}; whole expression match is \texttt{[#Mid(testString,st1.pos[1],st1.len[1])#]}.

Subsequent elements of the arrays provide the position and length of the first instance of each parenthesized subexpression therein.

Subsequent elements of the arrays provide the position and length of the first instance of each parenthesized subexpression therein.

Position is \texttt{#st1.pos[1]#}; Length is \texttt{#st1.len[1]#}; Substring is \texttt{[#Mid(testString,st1.pos[1],st1.len[1])#]}.
**REFindNoCase**

**Description**

Uses a regular expression (RE) to search a string for a pattern, starting from a specified position. The search is case-insensitive.

For more information on regular expressions, including escape sequences, anchors, and modifiers, see Chapter 7, “Using Regular Expressions in Functions,” in *ColdFusion MX Developer’s Guide*.

**Returns**

Depends on the value of the `returnsubexpressions` parameter:

- If `returnsubexpressions` = "False":
  - The position in the string where the match begins
  - 0, if the regular expression is not matched in the string
- If `returnsubexpressions` = "True": a structure that contains two arrays, `len` and `pos`. The array elements are as follows:
  - If the regular expression is found in the string, the first element of the `len` and `pos` arrays contains the length and position, respectively, of the first match of the entire regular expression.
  - If the regular expression contains parentheses that group subexpressions, each subsequent array element contains the length and position, respectively, of the first occurrence of each group.
  - If the regular expression is not found in the string, the first element of the `len` and `pos` arrays contains 0.

**Category**

String functions

**Function syntax**

`REFindNoCase(reg_expression, string [, start] [, returnsubexpressions] )`

**See also**

*Find, FindNoCase, REFind, REReplace, REReplaceNoCase*

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| `reg_expression` | Regular expression for which to search. Case-insensitive. For more information, see Chapter 7, "Using Regular Expressions in Functions," in *ColdFusion MX Developer’s Guide*.
| `string`      | A string or a variable that contains one. String in which to search. |
This function finds the first occurrence of a regular expression in a string. To find the second and subsequent instances of the expression or of subexpressions in it, you call this function more than once, each time with a different start position. To determine the next start position, use the `returnsSubexpressions` parameter, and add the value returned in the first element of the length array to the value in the first element of the position array.

**Example**

```cftml
<cfset teststring = "The cat in the hat hat came back!">
<p>The string in which the function is to search is:<br>
<cfoutput><b>#teststring#</b></cfoutput></p>
```

### Usage

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>start</td>
<td>Optional. A positive integer or a variable that contains one. Position at which to start search. The default value is 1.</td>
</tr>
<tr>
<td>returnsSubexpressions</td>
<td>Optional. Boolean. Whether to return substrings of reg_expression, in arrays named <code>len</code> and <code>pos</code>:&lt;br&gt;• True: if the regular expression is found, the first array element contains the length and position, respectively, of the first match.&lt;br&gt;• False: the function returns the position in the string where the match begins. Default.</td>
</tr>
</tbody>
</table>

**Example**

```cftml
<cfset teststring = "The cat in the hat hat came back!">
<p>The string in which the function is to search is:<br>
<cfoutput><b>#teststring#</b></cfoutput></p>
```
The first call to REFindNoCase to search this string is:
<cfset st = REFindNoCase("[[:alpha:]]+",testString,1,"True")>
This function returns a structure that contains two arrays: pos and len. To create this structure you can use a CFSET statement. For example:
<cfset st = REFindNoCase("[[:alpha:]]+",testString,1,"True")>
<cfoutput>
The number of elements in each array: #ArrayLen(st.pos)#.
</cfoutput>
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. The value of st.pos[1] is: <cfoutput>#st.pos[1]#.</cfoutput>
The value of st.len[1] is: <cfoutput>#st.len[1]#.</cfoutput>
Substring is <cfoutput><#Mid(testString,st.pos[1],st.len[1])#></cfoutput>

However, if you use parentheses to denote subexpressions in the regular expression, the first element contains the position and length of the first instance of the whole expression. The position and length of the first instance of each subexpression within will be included in additional array elements. For example:
<cfset st1 = REFindNoCase("([[:alpha:]]+)[ ]+(\1)",testString,1,"True")>
The number of elements in each array is <cfoutput>#ArrayLen(st1.pos)#</cfoutput>
First whole expression match: position is <cfoutput>#st1.pos[1]#; length is #st1.len[1]#; whole expression match is <cfoutput><#Mid(testString,st1.pos[1],st1.len[1])#></cfoutput>
Subsequent elements of the arrays provide the position and length of the first instance of each parenthesized subexpression therein. For example: from = "2" to = "#ArrayLen(st1.pos)#" Style="block" align="left">
Position is #st1.pos[1]#; Length is #st1.len[1]#; Substring is <cfoutput><#Mid(testString,st1.pos[1],st1.len[1])#></cfoutput>
ReleaseComObject

Description
Releases a COM Object and frees up resources that it used.

Returns
Nothing.

Category
Extensibility functions

Function syntax
ReleaseComObject(objectName)

See also
CreateObject, cfobject

History
ColdFusion MX 6.1: Added this function.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>objectName</td>
<td>Variable name of a COM object that was created using the CreateObject function or cfobject tag.</td>
</tr>
</tbody>
</table>

Usage
This function forcefully terminates and releases the specified COM object and all COM objects that it created. Use this function when the object is no longer in use, to quickly free up resources. If the COM object has a method, such as a quit method, that terminates the program, call this method before you call the ReleaseComObject function.

This function can improve processing efficiency, but is not required for an application to work. If you do not use this function, the Java garbage collection mechanism eventually frees the resources. If you use this function on an object that is in use, the object is prematurely released and your application will get exceptions.

Example
<h3>ReleaseComObject Example</h3>
<cfscript>
obj = CreateObject("Com", "excel.application.9"); //code that uses the object goes here
obj.quit();
ReleaseComObject(obj);
</cfscript>
RemoveChars

Description

Removes characters from a string.

Returns

A copy of the string, with count characters removed from the specified start position. If no characters are found, returns zero.

Category

String functions

Function syntax

RemoveChars(string, start, count)

See also

Insert, Len

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>string</td>
<td>A string or a variable that contains one. String in which to search.</td>
</tr>
<tr>
<td>start</td>
<td>A positive integer or a variable that contains one. Position at which to start search.</td>
</tr>
<tr>
<td>count</td>
<td>Number of characters to remove.</td>
</tr>
</tbody>
</table>

Example

RemoveChars Example

Returns a string with count characters removed from the start position. Returns 0 if no characters are found.

```cfml
<cfif IsDefined("FORM.myString")>
  <cfif (FORM.numChars + FORM.start) GT Len(FORM.myString)>
    <p>Your string is only <cfoutput>#Len(FORM.myString)#+ characters long. Please enter a longer string, select fewer characters to remove or begin earlier in the string.<cfoutput></p>
  </cfif>
</cfif>

<cfelse>
  <cfoutput>
    <p>Your original string: #FORM.myString#</p>
    <p>Your modified string:#RemoveChars(FORM.myString, FORM.start, FORM.numChars)#</p>
  </cfoutput>
</cfif>
```
**RepeatString**

**Description**

Creates a string that contains a specified number of repetitions of the specified string.

**Returns**

A string.

**Category**

String functions

**Function syntax**

```
RepeatString(string, count)
```

**See also**

CJustify, LJustify, RJustify

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>string</td>
<td>A string or a variable that contains one.</td>
</tr>
<tr>
<td>count</td>
<td>Number of repeats.</td>
</tr>
</tbody>
</table>

**Example**

```
<h3>RepeatString Example</h3>
<p>RepeatString returns a string created from <i>string</i>, repeated a specified number of times.</p>
<ul>
  <li>RepeatString(".-", 10): <cfoutput>#RepeatString(".-", 10)#</cfoutput></li>
  <li>RepeatString("&lt;BR&gt;", 3): <cfoutput>#RepeatString("<br>", 3)#</cfoutput></li>
  <li>RepeatString("", 5): <cfoutput>#RepeatString("", 5)#</cfoutput></li>
  <li>RepeatString("abc", 0): <cfoutput>#RepeatString("abc", 0)#</cfoutput></li>
  <li>RepeatString("Lorem Ipsum", 2): <cfoutput>#RepeatString("Lorem Ipsum", 2)#</cfoutput></li>
</ul>
```
Replace

Description
Replaces occurrences of substring1 in a string with substring2, in a specified scope. The search is case-sensitive.

Returns
The string, after making replacements.

Category
String functions

Function syntax
Replace(string, substring1, substring2 [, scope ])

See also
Find, REFind, ReplaceNoCase, ReplaceList, REReplace

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>string</td>
<td>A string or a variable that contains one. String in which to search.</td>
</tr>
<tr>
<td>substring1</td>
<td>A string or a variable that contains one. String for which to search</td>
</tr>
<tr>
<td>substring2</td>
<td>String that replaces substring1</td>
</tr>
</tbody>
</table>
| scope     | • one: replaces the first occurrence (default)  
                         • all: replaces all occurrences |

Usage
To remove a string, specify the empty string (""") as substring2.
You do not need to escape comma characters in strings. For example, the following code deletes the commas from the sentence:
```coldfusion
replace("The quick brown fox jumped over the lazy cow, dog, and cat.", ",", ",", "All")
```

Example
```coldfusion
<h3>Replace Example</h3>
<p>The Replace function returns <i>string</i> with <i>substring1</i> replaced by <i>substring2</i> in the specified scope. This is a case-sensitive search.

<cfif IsDefined("FORM.MyString")>
<p>Your original string. <cfoutput>#FORM.MyString#</cfoutput>
<p>You wanted to replace the substring <cfoutput>#FORM.MySubstring1#</cfoutput>  
                                 with the substring <cfoutput>#FORM.MySubstring2#</cfoutput>.
<p>The result: <cfoutput>#Replace(FORM.myString,  
                                 FORM.MySubstring1, FORM.mySubString2)#</cfoutput>
</cfif>
```
ReplaceList

Description
Replaces occurrences of the elements from a delimited list in a string with corresponding elements from another delimited list. The search is case-sensitive.

Returns
A copy of the string, after making replacements.

Category
List functions, String functions

Function syntax
ReplaceList(string, list1, list2)

See also
Find, REFind, Replace, REReplace

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>string</td>
<td>A string, or a variable that contains one, within which to replace substring</td>
</tr>
<tr>
<td>list1</td>
<td>Comma-delimited list of substrings for which to search</td>
</tr>
<tr>
<td>list2</td>
<td>Comma-delimited list of replacement substrings</td>
</tr>
</tbody>
</table>

Usage
The list of substrings to replace is processed sequentially. If a list1 element is contained in list2 elements, recursive replacement might occur. The second example shows this.

Example
<p>The ReplaceList function returns <i>string</i> with <i>substringlist1</i> (e.g. "a,b") replaced by <i>substringlist2</i> (e.g. "c,d") in the specified scope.</p><cfif IsDefined("FORM.MyString")>
<p>Your original string, <cfoutput>#FORM.MyString#</cfoutput></p>
<p>You wanted to replace the substring <cfoutput>#FORM.MySubstring1#</cfoutput></p>
<p>with the substring <cfoutput>#FORM.MySubstring2#</cfoutput>.</p>
<p>The result: <cfoutput>#ReplaceList(FORM.myString, FORM.MySubstring1, FORM.mySubString2)#</cfoutput></p></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 = ""></p>
</form>
<h3>ReplaceList Example Two</h3>
<cfset stringtoreplace = "The quick brown fox jumped over the lazy dog.""
<cfoutput>
  #ReplaceList(stringtoreplace,"dog,brown,fox,black", "cow,black,ferret,white")#
</cfoutput>
ReplaceNoCase

Description
Replaces occurrences of substring1 with substring2, in the specified scope. The search is case-insensitive.

Returns
A copy of the string, after making replacements.

Category
String functions

Function syntax
ReplaceNoCase(string, substring1, substring2 [, scope ])

See also
Find, REFind, Replace, ReplaceList, REReplace

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>string</td>
<td>A string (or variable that contains one) within which to replace substring.</td>
</tr>
<tr>
<td>substring1</td>
<td>String (or variable that contains one) to replace, if found.</td>
</tr>
<tr>
<td>substring2</td>
<td>String (or variable that contains one) that replaces substring1.</td>
</tr>
</tbody>
</table>
| scope     | • one: replaces the first occurrence (default).  
           | • all: replaces all occurrences. |

Example
<h3>ReplaceNoCase Example</h3>
<p>The ReplaceNoCase function returns <i>string</i> with <i>substring1</i> replaced by <i>substring2</i> in the specified scope. The search/replace is case-insensitive.

<cfif IsDefined("FORM.MyString")>
<p>Your original string. <cfoutput>#FORM.MyString#</cfoutput>
<p>You wanted to replace the substring <cfoutput>#FORM.MySubstring1#</cfoutput> with the substring <cfoutput>#FORM.MySubstring2#</cfoutput>. The result: <cfoutput>#ReplaceNoCase(FORM.myString, FORM.MySubstring1, FORM.mySubString2)#</cfoutput>
</cfif>
REReplace

Description
Uses a regular expression (RE) to search a string for a string pattern and replace it with another. The search is case-sensitive.

Returns
If the scope parameter is set to one, returns a string with the first occurrence of the regular expression replaced by the value of substring.
If the scope parameter is set to all, returns a string with all occurrences of the regular expression replaced by the value of substring.
If the function finds no matches, it returns a copy of the string unchanged.

Category
String functions

Function syntax
REReplace(string, reg_expression, substring [, scope ])

See also
REFind, Replace, ReplaceList, REReplaceNoCase

History
ColdFusion MX: Added supports for the following special codes in a replacement substring, to control case conversion:
• \u - uppercase the next character
• \l - lowercase the next character
• \U - uppercase until \E
• \L - lowercase until \E
• \E - end \U or \L
For more information on new features, see REFind.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>string</td>
<td>A string or a variable that contains one. String within which to search.</td>
</tr>
<tr>
<td>reg_expression</td>
<td>Regular expression to replace. The search is case-sensitive.</td>
</tr>
<tr>
<td>substring</td>
<td>A string or a variable that contains one. Replaces reg_expression.</td>
</tr>
<tr>
<td>scope</td>
<td>• one: replaces the first occurrence (default).</td>
</tr>
<tr>
<td></td>
<td>• all: replaces all occurrences.</td>
</tr>
</tbody>
</table>
Usage

For details on using regular expressions, see Chapter 7, “Using Regular Expressions in Functions,” in ColdFusion MX Developer’s Guide.

Example

The `REReplace` function returns `<i>string</i>` with a regular expression replaced with `<i>substring</i>` in the specified scope. Case-sensitive search.

```cfoutput`
REReplace("CABARET","C|B","G","ALL");
REReplace("CABARET","[A-Z]","G","ALL");
REReplace("I love jellies","jelly","cookies");
REReplace("I love jellies","jell(y|ies)","cookies");
REReplace("I love jelly","jell(y|ies)","cookies");
```

```
REReplaceNoCase

Description
Uses a regular expression to search a string for a string pattern and replace it with another. The search is case-insensitive.

Returns
- If `scope = "one"`: returns a string with the first occurrence of the regular expression replaced by the value of `substring`.
- If `scope = "all"`: returns a string with all occurrences of the regular expression replaced by the value of `substring`.
- If the function finds no matches: returns a copy of the string, unchanged.

Category
String functions

Function syntax
```
REReplaceNoCase(string, reg_expression, substring [, scope ])
```

See also
RFind, RFindNoCase, Replace, ReplaceList

History
ColdFusion MX: Changed behavior: this function inserts the following special characters in regular expression replacement strings, to control case conversion: \u, \U, \l, \L, and \E. If any of these strings is present in a ColdFusion 5 application, you must insert a backslash before it (for example, change \u to \\u).

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>string</td>
<td>A string or a variable that contains one.</td>
</tr>
<tr>
<td>reg_expression</td>
<td>Regular expression to replace. For more information, see Chapter 7, &quot;Using Regular Expressions in Functions,&quot; in ColdFusion MX Developer's Guide.</td>
</tr>
<tr>
<td>substring</td>
<td>A string or a variable that contains one. Replaces <code>reg_expression</code>.</td>
</tr>
<tr>
<td>scope</td>
<td>• one: replaces the first occurrence of the regular expression. Default.</td>
</tr>
<tr>
<td></td>
<td>• all: replaces all occurrences of the regular expression.</td>
</tr>
</tbody>
</table>

Usage
For details on using regular expressions, see Chapter 7, “Using Regular Expressions in Functions,” in ColdFusion MX Developer’s Guide.

Example
```
<p>The REReplaceNoCase function returns `<i>string</i>` with a regular expression replaced with `<i>substring</i>` in the specified scope. This is a case-insensitive search.</p>
<p>REReplaceNoCase("cabaret","C|B","G","ALL"): 
```
<cfoutput>@REReplaceNoCase("cabaret","C|B","G","ALL")#</cfoutput>
<p>REReplaceNoCase("cabaret","[A-Z]","G","ALL"): </p>
<cfoutput>@REReplaceNoCase("cabaret","[A-Z]","G","ALL")#</cfoutput>
<p>REReplaceNoCase("I LOVE JELLIES","jell(y|ies)","cookies"): </p>
<cfoutput>@REReplaceNoCase("I LOVE JELLIES","jell(y|ies)","cookies")#</cfoutput>
<p>REReplaceNoCase("I LOVE JELLY","jell(y|ies)","cookies"): </p>
<cfoutput>@REReplaceNoCase("I LOVE JELLY","jell(y|ies)","cookies")#</cfoutput>
Reverse

Description
Reverses the order of items, such as the characters in a string, the digits in a number, or the elements in an array.

Returns
A copy of string, with the characters in reverse order.

Category
String functions

Function syntax
Reverse(string)

See also
Left, Mid, Right

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>string</td>
<td>A string or a variable that contains one</td>
</tr>
</tbody>
</table>

Usage
You can call this function on a number with code such as the following:

```cfoutput```
reverse(6*2) equals #reverse(6*2)#
```

This code outputs the following:
reverse(6*2) equals 21

Example

```
Reverse Example

Reverse returns your string with the positions of the characters reversed.
<cfif IsDefined("FORM.myString")>
  <cfif FORM.myString is not ">
    <p>Reverse returned: #Reverse(FORM.myString)#</p>
  </cfif>
<cfelse>
  <p>Please enter a string to be reversed.</p>
</cfif>
```

```
<cfif IsDefined("FORM.myString")>
  <cfif FORM.myString is not ">
    <p>Reverse returned: #Reverse(FORM.myString)#</p>
  </cfif>
</cfif>
```

Example

```
Reverse Example

Reverse returns your string with the positions of the characters reversed.
<cfif IsDefined("FORM.myString")>
  <cfif FORM.myString is not ">
    <p>Reverse returned: #Reverse(FORM.myString)#</p>
  </cfif>
<cfelse>
  <p>Please enter a string to be reversed.</p>
</cfif>
```

```
<cfif IsDefined("FORM.myString")>
  <cfif FORM.myString is not ">
    <p>Reverse returned: #Reverse(FORM.myString)#</p>
  </cfif>
</cfif>
```

```
<form action = "reverse.cfm">
  <p>Enter a string to be reversed:</p>
  <input type = "Text" name = "MyString">
  <p><input type = "Submit" name = ""></p>
</form>
```

824  Chapter 3: ColdFusion Functions
Right

Description

Gets a specified number of characters from a string, beginning at the right.

Returns the specified number of characters from the end (or right side) of the specified string.

Returns

• If the length of the string is greater than or equal to count, the rightmost count characters of the string
• If count is greater than the length of the string, the whole string
• If count is greater than 1, and the string is empty, an empty string

Category

String functions

Function syntax

Right(string, count)

See also

Left, Mid, Reverse

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>string</td>
<td>A string or a variable that contains one.</td>
</tr>
<tr>
<td>count</td>
<td>A positive integer that specifies the number of characters to return.</td>
</tr>
</tbody>
</table>

Example

<!--- Simple Right Example--->
<cfoutput>
#Right("See the quick red fox jump over the fence", 9)#
<br>
#Right("ColdFusion", 6)#
</cfoutput>

<!--- Right Example using form input --->
<h3>Right Example</h3>
<cfif IsDefined("Form.MyText")>
<!--- If len returns 0 (zero), then show error message. --->
<cfif Len(FORM.myText)>
<cfif Len(FORM.myText) LTE FORM.RemoveChars>
<p style="color: red; font-weight: bold">Your string #FORM.myText# only has #Len(FORM.myText)# characters. You cannot output the #FORM.removeChars# rightmost characters of this string because it is not long enough.</p>
</cfoutput>
<cfelse>
<p>Your original string: <strong>#FORM.myText#</strong></p>
<p>Your changed string, showing only the <strong>#FORM.removeChars#</strong> rightmost characters:
</p><strong>#Right(Form.myText, FORM.removeChars)#</strong></p>
</cfoutput>

Right 825
<cfoutput>
</cfoutput>  
<cfif>
  <cfelse>
    <p style="color: red; font-weight: bold">Please enter a string of more than 0 (zero) characters.</p>
  </cfif>
</cfif>

<form action="<cfoutput>#CGI.ScriptName#</cfoutput>" method="POST">
<p>Type in some text<br />
<input type="Text" name="myText"></p>
<p>How many characters from the right do you want to show?
<select name="RemoveChars">
  <option value="1">1</option>
  <option value="3" selected>3</option>
  <option value="5">5</option>
  <option value="7">7</option>
  <option value="9">9</option>
</select>
<input type="Submit" name="Submit" value="Remove characters"></p>
</form>
RJustify

Description
Right justifies characters of a string.

Returns
A copy of a string, right-justified in the specified field length.

Category
Display and formatting functions, String functions

Function syntax
RJustify(string, length)

See also
CJustify, LJustify

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>string</td>
<td>A string enclosed in quotation marks, or a variable that contains one.</td>
</tr>
<tr>
<td>length</td>
<td>A positive integer or a variable that contains one. Length of field in which to justify string.</td>
</tr>
</tbody>
</table>

Example
<!--- This example shows how to use RJustify --->
<cfparam name = "jstring" default = "">
<cfif IsDefined("FORM.justifyString")>
  <cfset jstring = rjustify(FORM.justifyString, 35)>
</cfif>
<html>
<head>
<title>RJustify Example</title>
</head>
<body>
<h3>RJustify Function</h3>
<p>Enter a string. It will be right justified within the sample field</p>
<form action = "rjustify.cfm">
  <p><input type = "Text" value = "#{jString}" size = 35 name = "justifyString"> </p>
  <p><input type = "Submit" name = "> <input type = "reset"> </p>
</form>
**Round**

**Description**

Rounds a number to the closest integer that is larger than the input parameter.

**Returns**

An integer.

**Category**

Mathematical functions

**Function syntax**

`Round(number)`

**See also**

`Ceiling, Fix, Int`

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>number</td>
<td>Number to round</td>
</tr>
</tbody>
</table>

**Usage**

Use this function to round a number. This function rounds numbers that end with .5 up to the nearest integer. It rounds 3.5 to 4 and -3.5 to -3.

**Example**

```cfml
<h3>Round Example</h3>
<p>This function rounds a number to the closest integer.</p>
<ul>
  <li>Round(7.49) : <cfoutput>#Round(7.49)#</cfoutput></li>
  <li>Round(7.5) : <cfoutput>#Round(7.5)#</cfoutput></li>
  <li>Round(-10.775) : <cfoutput>#Round(-10.775)#</cfoutput></li>
  <li>Round(-35.5) : <cfoutput>#Round(-35.5)#</cfoutput></li>
  <li>Round(35.5) : <cfoutput>#Round(35.5)#</cfoutput></li>
  <li>Round(1.2345*100)/100 : <cfoutput>#Round(1.2345*100)/100#</cfoutput></li>
</ul>
```
RTrim

**Description**

Removes spaces from the end of a string.

**Returns**

A copy of `string`, after removing trailing spaces.

**Category**

String functions

**Function syntax**

`RTrim(string)`

**See also**

`LTrim, Trim`

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>string</code></td>
<td>A string or a variable that contains one</td>
</tr>
</tbody>
</table>

**Example**

```cftml
<h3>RTrim Example</h3>

<cfif IsDefined("FORM.myText")>
  <cfoutput>
    <pre>
      Your string:
      "#FORM.myText#"
      Your string:
      "#RTrim(FORM.myText)#"
      (right trimmed)
    </pre>
  </cfoutput>
</cfif>

<form action="Rtrim.cfm" method="post">
  <p>Enter some text. It will be modified by Rtrim to remove spaces from the right.</p>
  <input type="Text" name="myText" value="TEST   ">
  <input type="Submit" name="">
</form>
```
Second

Description

Extracts the ordinal for the second from a date/time object.

Returns

An integer in the range 0–59.

Category

Date and time functions

Function syntax

Second(date)

See also

DatePart, Hash, Minute

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>date</td>
<td>A date/time object</td>
</tr>
</tbody>
</table>

Usage

When passing a date/time object as a string, you must enclose it in quotation marks. Otherwise, it is interpreted as a number representation of a date/time object.

Example

<!--- This example shows the use of Hour, Minute, and Second --->
<h3>Second Example</h3>
<cfoutput>
The time is currently #TimeFormat(Now())#.
We are in hour #Hour(Now())#, Minute #Minute(Now())# and Second #Second(Now())# of the day.
</cfoutput>
SendGatewayMessage

Description
Sends an outgoing message through a ColdFusion MX event gateway.

Returns
String. The value returned depends on the gateway type.

Category
Extensibility functions

Function syntax
SendGatewayMessage(gatewayID, data)

See also

History
ColdFusion MX 7: Added this function.

Parameters
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>gatewayID</td>
<td>Identifier of the gateway to send the message. Must be the Gateway ID of one of the ColdFusion event gateway instances configured on the ColdFusion MX Administrator Event Gateways section’s Gateways page.</td>
</tr>
<tr>
<td>data</td>
<td>A ColdFusion structure. The contents of the structure depend on the event gateway type, but typically include a MESSAGE field that contains the message to send and a field that contains the destination address.</td>
</tr>
</tbody>
</table>

Usage
The SendGatewayMessage function calls the specified gateway’s outgoingMessage method. The value returned by the function depends on the gateway type. The following table describes the return values for standard ColdFusion MX gateway types:

<table>
<thead>
<tr>
<th>Gateway type</th>
<th>Return values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Asynchronous CFML</td>
<td>If the message was queued for delivery to the CFC, returns True; False, otherwise.</td>
</tr>
<tr>
<td>Lotus SameTime</td>
<td>If the message or command was successful, returns OK. If an error occurred, returns a string indicating the cause.</td>
</tr>
</tbody>
</table>
Example

The following example uses an instance of the CFML gateway to log messages asynchronously to a file. To use this example, you must configure an instance of the CFML gateway with the name “Asynch Logger” in the ColdFusion MX Administrator. This gateway instance must use a CFC that takes the message and logs it. For sample CFC code, see “Using the CFML event gateway for asynchronous CFCs” in Chapter 42, “Using Event Gateways,” in ColdFusion MX Developer’s Guide.

Sending an event to the CFML event gateway that is registered in the ColdFusion MX Administrator as Asynch Logger.<br>
<cfscript>
status = false;
props = structNew();
props.message = "Replace me with a variable with data to log";
status = SendGatewayMessage("Asynch Logger", props);
if (status IS True) {WriteOutput("Event Message \\
props.message has been sent.");
</cfscript>
SetEncoding

Description
Sets the character encoding (character set) of Form and URL scope variable values; used when the character encoding of the input to a form, or the character encoding of a URL, is not in UTF-8 encoding.

Returns
None

Category
International functions, System functions

Function syntax
SetEncoding(scope_name,charset)

See also
GetEncoding, cfcontent, cfprocessingdirective, URLEncode, URLEncodedFormat;

History
ColdFusion MX: Added this function.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>scope_name</td>
<td>• url  • form</td>
</tr>
<tr>
<td>charset</td>
<td>The character encoding in which text in the scope variables is encoded. The following list includes commonly used values: • utf-8 • iso-8859-1 • windows-1252 • us-ascii • shift_jis • iso-2022-jp • euc-jp • euc-kr • big5 • euc-cn • utf-16</td>
</tr>
</tbody>
</table>
Usage

Use this function when the character encoding of the input to a form or the character encoding of a URL is not in UTF-8 encoding. For example, Traditional Chinese characters are often in Big5 encoding. This function resets URL and Form variables, so you should call it before using these variables (typically, in the Application.cfm page or Application.cfc file). Calling this function first also avoids interpreting the characters of the variables incorrectly.

For more information on character encoding, see the following web pages:

- [www.w3.org/International/O-charset.html](http://www.w3.org/International/O-charset.html) provides general information on character encoding and the web, and has several useful links.
- [www.iana.org/assignments/character-sets](http://www.iana.org/assignments/character-sets) is a complete list of character sets names used on the Internet, maintained by the Internet Assigned Numbers Authority.
- [java.sun.com/j2se/1.4.1/docs/guide/intl/encoding.doc.html](http://java.sun.com/j2se/1.4.1/docs/guide/intl/encoding.doc.html) lists the character encoding that Java 1.4.1, and therefore the default ColdFusion configuration, can interpret. If you use a JVM that does not conform to the Sun Java 2 Platform, Standard Edition, v 1.4.1, the supported locales may differ. The list uses Java internal names, not the IANA character encoding names that you normally use in the SetEncoding charset parameter and other ColdFusion attributes and parameters. Java automatically converts standard IANA names to its internal names as needed.

Example

<!--- This example sends and interprets the contents of two fields as big5 encoded text. Note that the form fields are received as URL variables because the form uses the GET method. --->
<cfcontent type="text/html; charset=big5">
<form action='#cgi.script_name#' method='get'>
<input name='xxx' type='text'>
<input name='yyy' type='text'>
<input type="Submit" value="Submit">
</form>

<cfif IsDefined("URL.xxx")>
<cfscript>
   SetEncoding("url", "big5");
   WriteOutput("URL.XXX is " & URL.xxx & "<br>");
   WriteOutput("URL.YYY is " & URL.yyy & "<br>");
   theEncoding = GetEncoding("URL");
   WriteOutput("The URL variables were decoded using '" & theEncoding & '" encoding.");
</cfscript>
</cfif>
</cfcontent>
SetLocale

Description
Sets the country/language locale for ColdFusion processing and the page returned to the client. The locale value determines the default format of date, time, number, and currency values, according to language and regional conventions.

Returns
The locale value prior to setting the new locale, as a string.

Category
International functions, System functions

Function syntax
SetLocale(new_locale)

See also
GetHttpTimeString, GetLocale, GetLocaleDisplayName; “Locales” in Chapter 17, “Developing Globalized Applications” in ColdFusion MX Developer’s Guide

History
ColdFusion MX 7: Added support for all locales supported by the ColdFusion MX Java runtime.

ColdFusion MX:
• Changed formatting behavior: this function might return a different value than in earlier releases. This function uses Java standard locale determination and formatting rules on all platforms.
• Deprecated the Spanish (Mexican) locale option. It might not work, and it might cause an error, in later releases.
• Changed the Spanish (Modern) option: it now sets the locale to Spanish (Standard).

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>new_locale</td>
<td>The name of a locale; for example, &quot;English (US)&quot;</td>
</tr>
</tbody>
</table>

Usage
You can specify any locale name that is listed in the Server.Coldfusion.SupportedLocales variable. This variable is a comma-delimited list of all locale names supported by the JVM, plus the locale names that were required by ColdFusion MX prior to ColdFusion MX 7.

The following locale names were used in ColdFusion releases through ColdFusion MX 6.1, and continue to be supported. If you use any of these values in the SetLocale function, the GetLocale function returns the name you set, not the corresponding Java locale name.

<table>
<thead>
<tr>
<th>Chinese (China)</th>
<th>French (Belgian)</th>
<th>Korean</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chinese (Hong Kong)</td>
<td>French (Canadian)</td>
<td>Norwegian (Bokmal)</td>
</tr>
</tbody>
</table>
ColdFusion determines the locale value as follows:

- By default, ColdFusion uses the JVM locale, and the default JVM locale is the operating system locale. You can set JVM locale value explicitly in ColdFusion MX in the ColdFusion Administrator Java and JVM Settings page JVM Arguments field; for example: `-Duser.language=de -Duser.region=DE`.

- A locale set using the `SetLocale` function persists for the current request or until it is reset by another `SetLocale` function in the request.

- If a request has multiple `SetLocale` functions, the current locale setting affects how locale-sensitive ColdFusion tags and functions, such as the functions that start with LS format data. The last `SetLocale` function that ColdFusion processes before sending a response to the requestor (typically the client browser) determines the value of the response `Content-Language` HTTP header. The browser that requested the page displays the response according to the rules for the language specified by the `Content-Language` header.

- ColdFusion ignores any `SetLocale` functions that follow a `cfflush` tag.

Because this function returns the previous locale setting, you can save the original locale value. You can restore the original locale by calling `SetLocale` again with the saved variable. For example, the following line saves the original locale ins a Session variable:

```
<cfset Session.oldlocale = SetLocale(newLocale)>
```

The variable `server.ColdFusion.SupportedLocales` is initialized at startup with a comma-delimited list of the locales that ColdFusion and the operating system support. If you call `SetLocale` with a locale that is not in the list, the call generates an error.

**Note:** ColdFusion uses the Spanish (Standard) formats for Spanish (Modern) and Spanish (Standard).

### Example

```html
<h3>SetLocale Example</h3>
<p>SetLocale sets the locale to the specified new locale for the current session. A locale encapsulates the set of attributes that govern the display and formatting of date, time, number, and currency values. The locale for this system is `@GetLocale()@`.

```
SetProfileString

Description
Sets the value of a profile entry in an initialization file.

Returns
An empty string, upon successful execution; otherwise, an error message.

Category
System functions

Function syntax
SetProfileString(iniPath, section, entry, value)

See also
GetProfileSections, GetProfileString, SetProfileString

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>iniPath</td>
<td>Absolute path of initialization file</td>
</tr>
<tr>
<td>section</td>
<td>Section of the initialization file in which the entry is to be set</td>
</tr>
<tr>
<td>entry</td>
<td>Name of the entry to set</td>
</tr>
<tr>
<td>value</td>
<td>Value to which to set the entry</td>
</tr>
</tbody>
</table>

Example
<h3>SetProfileString Example</h3>
This example uses SetProfileString to set the timeout value in an initialization file. Enter the full path of your initialization file, specify the timeout value, and submit the form.

<!--- This section checks whether the form was submitted. If so, this section sets the initialization path and timeout value to the path and timeout value specified in the form --->
<cfif Isdefined("Form.Submit")>

<cfset IniPath = FORM.iniPath>
<cfset Section = "boot loader">
<cfset MyTimeout = FORM.MyTimeout>
<cfset timeout = GetProfileString(IniPath, Section, "timeout")>
<cfif timeout Is Not MyTimeout>
<cfif MyTimeout Greater Than 0>
<hr size = "2" color = "#0000A0">
<p>Setting the timeout value to <cfoutput>#MyTimeout#</cfoutput></p>
<cfset code = SetProfileString(IniPath, Section, "timeout", MyTimeout)>
<p>Value returned from SetProfileString:
<cfoutput>#code#</cfoutput></p>
<cfelse>

</cfelse>
Timeout value should be greater than zero in order to provide time for user response.

The timeout value in your initialization file is already set. Timeout is set to: 

Default directory is: 

```
<cfif timeout lt 0>
<p>Timeout value should be greater than zero in order to provide time for user response.</p>
</cfif>
<cfelse>
<p>The timeout value in your initialization file is already set: #MyTimeout#</p>
</cfif>

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

```
<h4>Boot Loader</h4>
<p>Timeout is set to: #timeout#</p>
<p>Default directory is: #default#</p>
```

```
<form action = "setprofilestring.cfm">
<table cellspacing = "2" cellpadding = "2" border = "0">
<tr>
<td>Full Path of Init File</td>
<td><input type = "Text" name = "IniPath" value = "C:\myboot.ini"></td>
</tr>
<tr>
<td>Timeout</td>
<td><input type = "Text" name = "MyTimeout" value = "30"></td>
</tr>
<tr>
<td><input type = "Submit" name = "Submit" value = "Submit"></td>
</tr>
</table>
</form>
SetVariable

Description
This function is no longer required in well-formed ColdFusion pages.
Sets a variable in the name parameter to the value of the value parameter.

Returns
The new value of the variable.

Category
Dynamic evaluation functions

Function syntax
SetVariable(name, value)

See also
DE, Evaluate, IIf

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Variable name</td>
</tr>
<tr>
<td>value</td>
<td>A string, the name of a string, or a number</td>
</tr>
</tbody>
</table>

Usage
You can use direct assignment statements in place of this function to set values of dynamically named variables. To do so, put the dynamically named variable in quotation marks and number signs (#); for example:

```<cfset DynamicVar2 = "ABD"> <cfset "#DynamicVar2#" = "Test Value2">```

Also, the following lines are equivalent:

```<cfset "myVar#i#" = myVal> SetVariable("myVar" & i, myVal)```

For more information, see Chapter 4, “Using Expressions and Number Signs,” in ColdFusion MX Developer's Guide.

Example

```<h3>SetVariable Example</h3> <cfif IsDefined("FORM.myVariable")> <!--- strip out url, client., cgi., session., caller. ---> <!--- This example only lets you set form variables ---> <cfset myName = ReplaceList(FORM.myVariable, "url.client.cgi.session.caller", "FORM,FORM,FORM,FORM,FORM")> <cfset temp = SetVariable(myName, FORM.myValue)> <cfset varName = myName>```
<cfset varNameValue = Evaluate(myName)>
<cfoutput>
    <p>Your variable, #varName#</p>
    <p>The value of #varName# is #varNameValue#</p>
</cfoutput>
</cfif>
**Sgn**

**Description**
Determines the sign of a number.

**Returns**
- 1, if `number` is positive.
- 0, if `number` is 0.
- -1, if `number` is negative.

**Category**
Mathematical functions

**Function syntax**
`Sgn(number)`

**See also**
Abs

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>number</td>
<td>A number</td>
</tr>
</tbody>
</table>

**Example**

```html
<h3>Sgn Example</h3>
<p>Sgn determines the sign of a number. Returns 1 if number is positive; 0 if number is 0; -1 if number is negative.</p>
<p>Sgn(14): <cfoutput>Sgn(14)</cfoutput></p>
<p>Sgn(21-21): <cfoutput>Sgn(21-21)</cfoutput></p>
<p>Sgn(-0.007): <cfoutput>Sgn(-0.007)</cfoutput></p>
```
Sin

Description
Calculates the sine of an angle that is entered in radians.

Returns
A number; the sine of the angle.

Category
Mathematical functions

Function syntax
\texttt{Sin(number)}

See also
\texttt{ASin, Cos, ACos, Tan, Atn, Pi}

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>number</td>
<td>Angle, in radians for which to calculate the sine.</td>
</tr>
</tbody>
</table>

Usage
The range of the result is -1 to 1.

To convert degrees to radians, multiply degrees by \( \pi/180 \). To convert radians to degrees, multiply radians by \( 180/\pi \).

Note: Because the function uses floating point arithmetic, it returns a very small number (such as 6.12323399574E-017) for angles that should produce 0. To test for a 0 value, check whether the value is less than 0.0000000000001.

Example

```coldfusion
<h3>Sin Example</h3>
<!--- Calculate sine if form has been submitted --->
<cfif IsDefined("FORM.sinNum")>
<!--- Make sure input is a number --->
<cfif IsNumeric(#FORM.sinNum#)>
<!--- Convert degrees to radians, call the Sin function. --->
<cfset sinValue=#Sin((Form.sinNum * PI()) / 180)#>
<!--- 0.0000000000001 is the function's precision limit. If absolute value of returned sine value is less, set result to 0 --->
<cfif Abs(sinValue) LT 0.0000000000001>
<cfset sinValue=0>
</cfif>
<cfoutput>
Sin(#FORM.sinNum#) = #sinValue#<br><br>
</cfoutput>
<cfelse>
<!--- If input is not a number, show an error message --->
<h4>You must enter a numeric angle in degrees.</h4>
```

842 Chapter 3: ColdFusion Functions
<form action="#CGI.script_name#" method="post">
Enter an angle in degrees to get its sine:<br><input type="Text" name="sinNum" size="15"><br><br>
<input type="Submit" name=""&nbsp;&nbsp;
<input type="RESET"
</form>
SpanExcluding

Description

Gets characters from a string, from the beginning to a character that is in a specified set of characters. The search is case-sensitive.

Returns

A string; characters from string, from the beginning to a character that is in set.

Category

String functions

Function syntax

SpanExcluding(string, set)

See also

GetToken, SpanIncluding; “Caching parts of ColdFusion pages” in Chapter 13, “Designing and Optimizing a ColdFusion Application,” in ColdFusion MX Developer’s Guide

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>string</td>
<td>A string or a variable that contains one</td>
</tr>
<tr>
<td>set</td>
<td>A string or a variable that contains one. Must contain one or more characters</td>
</tr>
</tbody>
</table>

Example

<h3>SpanExcluding Example</h3>

<cfif IsDefined("FORM.myString")>
<p>Your string was #FORM.myString#</p>
<p>Your set of characters was #FORM.mySet#</p>
<p>Your string up until one of the characters in the set is: #SpanExcluding(FORM.myString, FORM.mySet)#</p>
</cfif>

<p>Returns all characters from string from beginning to a character from the set of characters. The search is case-sensitive.</p>

<form method = post action = "spanexcluding.cfm">
<p>Enter a string:<br><input type = "Text" name = "myString" value = "Hey, you!">
And a set of characters:<br><input type = "Text" name = "mySet" value = "Ey"> <input type = "Submit" name = ">
</form>
SpanIncluding

Description

Gets characters from a string, from the beginning to a character that is not in a specified set of characters. The search is case-sensitive.

Returns

A string: characters from string, from the beginning to a character that is not in set.

Category

String functions

Function syntax

SpanIncluding(string, set)

See also

GetToken, SpanExcluding; “Caching parts of ColdFusion pages" in Chapter 13, “Designing and Optimizing a ColdFusion Application,” in ColdFusion MX Developer’s Guide

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>string</td>
<td>A string or a variable that contains the search string.</td>
</tr>
<tr>
<td>set</td>
<td>A string or a variable that contains a set of characters. Must contain one or more characters.</td>
</tr>
</tbody>
</table>

Example

<h3>SpanIncluding Example</h3>
<cfif IsDefined("FORM.myString")>
<p>Your string was <cfoutput>#FORM.myString#</cfoutput>
<p>Your set of characters was <cfoutput>#FORM.mySet#</cfoutput>
<p>Your string, until the characters in the set have been found, is: <cfoutput>#SpanIncluding(FORM.myString, FORM.mySet)#</cfoutput>
</cfif>

<p>Returns characters of a string, from beginning to a character that is not in set. The search is case-sensitive.

<form action = "spanincluding.cfm" method="post">
<p>Enter a string:
<br><input type = "Text" name = "myString" value = "Hey, you!">
<p>And a set of characters:
<br><input type = "Text" name = "mySet" value = "ey,H">
<br><input type = "Submit" name = "">
</form>
Sqr

Description
Calculates the square root of a number.

Returns
Number; square root of number.

Category
Mathematical functions

Function syntax
Sqr(number)

See also
Abs

Parameters
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>number</td>
<td>A positive integer or a variable that contains one. Number whose square root to get.</td>
</tr>
</tbody>
</table>

Usage
The value in number must be greater than or equal to 0.

Example
<h3>Sqr Example</h3>
<p>Returns the square root of a positive number.</p>
<p>Sqr(2): <cfoutput>#{Sqr(2)}#</cfoutput></p>
<p>Sqr(Abs(-144)): <cfoutput>#{Sqr(Abs(-144))}#</cfoutput></p>
<p>Sqr(25^2): <cfoutput>#{Sqr(25^2)}#</cfoutput></p>
StripCR

Description

Deletes return characters from a string.

Returns

A copy of string, after removing return characters.

Category

Display and formatting functions, String functions

Function syntax

StripCR(string)

See also

ParagraphFormat

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>string</td>
<td>A string or a variable that contains one</td>
</tr>
</tbody>
</table>

Usage

Useful for preformatted (between <pre> and </pre> tags) HTML display of data entered in textarea fields.

Example

<h3>StripCR Example</h3>

<p>Function StripCR is useful for preformatted HTML display of data (PRE) entered in textarea fields.</p>

<cfif isdefined("Form.myTextArea")>
<pre>
<cfoutput>#StripCR(Form.myTextArea)#</cfoutput>
</pre>
</cfif>

<!--- use #Chr(10)##Chr(13)# to simulate line feed/carriage return combination --->

<form action = "stripcr.cfm">
<textarea name = "MyTextArea" cols = "35" rows = 8>
This is sample text and you see how it scrolls
</textarea>
<input type = "Submit" name = "Show me the HTML version">
</form>
StructAppend

Description
Appends one structure to another.

Returns
True, upon successful completion; False, otherwise.

Category
Structure functions

Function syntax
StructAppend(struct1, struct2, overwriteFlag)

See also

History
ColdFusion MX: Changed behavior: this function can be used on XML objects.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>struct1</td>
<td>Structure to which struct2 is appended.</td>
</tr>
<tr>
<td>struct2</td>
<td>Structure that contains the data to append to struct1.</td>
</tr>
</tbody>
</table>
| overwriteFlag | • True or "Yes": values in struct2 overwrite corresponding values in struct1. Default.  
                     • False or "No": values in struct2 do not overwrite corresponding values in struct1. |

Usage
This function appends the fields and values of struct2 to struct1; struct2 is not modified. If struct1 already contains a field of struct2, overwriteFlag determines whether the value in struct2 overwrites it.

A structure’s keys are unordered.

Example
<html>
<body>
<!---- Create a Name structure --->
<cfset nameCLK=StructNew()>
<cfset nameCLK.first="Chris”>
<cfset nameCLK.middle="Lloyd”>
<cfset nameCLK.last="Gilson”>
<!---- Create an address struct --->
<cfset addrCLK=StructNew()>
<cfset addrCLK.street="17 Gigantic Rd”>
</body>
</html>
<cfset addrCLK.city="Watertown">
<cfset addrCLK.state="MA">
<cfset addrCLK.zip="02472">
<!---- Create a Person structure ---->
<cfset personCLK=StructNew()>
<cfset personCLK.name=#nameCLK#>
<cfset personCLK.addr=#addrCLK#>
<!---- Display the contents of the person struct before the Append ---->
<p>
The person struct <b>before</b> the Append call:<br>
<cfloop collection=#personCLK# item="myItem">
<cfoutput>
#myItem#<br>
</cfoutput>
</cfloop>
<!---- Merge the address struct into the top-level person struct ---->
<cfset bSuccess = StructAppend( personCLK, addrCLK )>
<!---- Display the contents of the person struct, after the Append ---->
<p>
The person struct <b>after</b> the Append call:
<cfloop collection=#personCLK# item="myItem">
<cfoutput>
#myItem#<br>
</cfoutput>
</cfloop>
StructClear

Description
Removes all data from a structure.

Returns
True, on successful execution; False, otherwise.

Category
Structure functions

Function syntax
StructClear(structure)

See also

History
ColdFusion MX: Changed behavior: this function can be used on XML objects.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>structure</td>
<td>Structure to clear</td>
</tr>
</tbody>
</table>

Usage
Do not call this function on a session variable. For more information, see TechNote 14143, “ColdFusion 4.5 and the StructClear(Session) function,” at www.coldfusion.com/Support/KnowledgeBase/SearchForm.cfm. (The article applies to ColdFusion 4.5, 5.x, and ColdFusion MX.)

Example
<!--- Shows StructClear function. Calls cf_addemployee custom tag which uses the addemployee.cfm file. --->
<body>
<!--- Establish params for first time through --->
<cfparam name = "Form.firstname" default = ">
<cfparam name = "Form.lastname" default = ">
<cfparam name = "Form.email" default = ">
<cfparam name = "Form.phone" default = ">
<cfparam name = "Form.department" default = ">
<cfif form.firstname eq ">
<p>Please fill out the form.
<cfelse>
<cfoutput>
<cfscript>
employee = StructNew();
StructInsert(employee, "firstname", Form.firstname);
StructInsert(employee, "lastname", Form.lastname);
StructInsert(employee, "email", Form.email);
StructInsert(employee, "phone", Form.phone);
StructInsert(employee, "department", Form.department);
</cfscript>
</cfoutput>

<!--- Call the custom tag that adds employees --->
<cf_addemployee empinfo = "#employee#">
<cfscript>StructClear(employee);</cfscript>
</cfif>
**StructCopy**

**Description**
Copies a structure. Copies top-level keys, values, and arrays in the structure by value; copies nested structures by reference.

**Returns**
A copy of a structure, with the same keys and values; if `structure` does not exist, throws an exception.

**Category**
Structure functions

**Function syntax**

```
StructCopy(structure)
```

**See also**

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>structure</td>
<td>Structure to copy</td>
</tr>
</tbody>
</table>

**Usage**

The following code shows how this function copies a structure that contains a string field, a number field, and a two-dimensional array at the top level:

```
<cfoutput>
  <cfset assignedCopy = StructNew()>
  <cfset assignedCopy.string = #struct.string#>
  <cfset assignedCopy.number = #struct.number#>
  <cfset assignedCopy.array = ArrayNew(2)>
  <cfset assignedCopy.array[1][1] = #struct.array[1][1]#>
  <cfset assignedCopy.array[1][2] = #struct.array[1][2]#>
</cfoutput>
```

The following code shows how `StructCopy` copies a nested structure:

```
<cfoutput>
  <cfset assignedCopy.nestedStruct = struct.nestedStruct>
</cfoutput>
```

To copy a structure entirely by value, use `Duplicate` on page 577.
The following table shows how variables are assigned:

<table>
<thead>
<tr>
<th>Variable type</th>
<th>Assigned by</th>
</tr>
</thead>
<tbody>
<tr>
<td>structure.any_simple_value</td>
<td>Value</td>
</tr>
<tr>
<td>Boolean</td>
<td>Value</td>
</tr>
<tr>
<td>Binary</td>
<td>Value</td>
</tr>
<tr>
<td>Base64</td>
<td>Value</td>
</tr>
<tr>
<td>structure.array</td>
<td>Value</td>
</tr>
<tr>
<td>structure.nested_structure</td>
<td>Reference</td>
</tr>
<tr>
<td>structure.object</td>
<td>Reference</td>
</tr>
<tr>
<td>structure.query</td>
<td>Reference</td>
</tr>
</tbody>
</table>

Example

```cfs
// This script creates a structure that StructCopy copies by value. <br>
<cfscript>
// Create elements.
s = StructNew();
s.array = ArrayNew(2):

// Assign simple values to original top-level structure fields.
s.number = 99;
s.string = "hello tommy":

// Assign values to original top-level array.
s.array[1][1] = "one one";
s.array[1][2] = "one two";
</cfscript>

// Copy this structure to a new structure. <br>
<cfset copied = StructCopy(s)>
```

```cfs
// Change the values of the original structure. <br>
s.number = 100;
s.string = "hello tommy (modified)";
s.array[1][1] = "one one (modified)";
s.array[1][2] = "one two (modified)";
</cfscript>```
Modified Original Values

```coldfusion
// Simple values
s.number = #s.number#

s.string = #s.string#

// Array value
s.array[1][1] = #s.array[1][1]#
s.array[1][2] = #s.array[1][2]#
```

Copied structure values should be the same as the original.

```coldfusion
// Simple values
copied.number = #copied.number#

copied.string = #copied.string#

// Array value
copied.array[1][1] = #copied.array[1][1]#
copied.array[1][2] = #copied.array[1][2]#
```

This script creates a structure that StructCopy copies by reference.

```coldfusion
// Create elements.
s = StructNew();
s.nested = StructNew();
s.nested.array = ArrayNew(2);

// Assign simple values to nested structure fields.
s.nested.number = 99;
s.nested.string = "hello tommy";

// Assign values to nested array.
s.nested.array[1][1] = "one one";
s.nested.array[1][2] = "one two";
```

Original Values

```coldfusion
// Simple values
s.nested.number = #s.nested.number#

s.nested.string = #s.nested.string#

// Array values
s.nested.array[1][1] = #s.nested.array[1][1]#
s.nested.array[1][2] = #s.nested.array[1][2]#
```

Use StructCopy to copy this structure to a new structure.

```coldfusion
<cfset copied = StructCopy(s)>
```

Use Duplicate to clone this structure to a new structure.

```coldfusion
<cfset duplicated = Duplicate(s)>
```

Change the values of the original structure.
s.nested.number = 100;
s.nested.string = "hello tommy (modified)";
s.nested.array[1][1] = "one one (modified)";
s.nested.array[1][2] = "one two (modified)";
</cfscript>
<hr>
<b>Modified Original Values</b><br>
<cfoutput>
// Simple values <br>
s.nested.number = #s.nested.number#<br>
s.nested.string = #s.nested.string#<br>
// Array value <br>
s.nested.array[1][1] = #s.nested.array[1][1]#<br>
s.nested.array[1][2] = #s.nested.array[1][2]#<br>
</cfoutput>
<hr>
<b>Copied structure values should reflect changes to original.</b><br>
<cfoutput>
// Simple values <br>
copied.nested.number = #copied.nested.number#<br>
copied.nested.string = #copied.nested.string#<br>
// Array values <br>
copied.nested.array[1][1] = #copied.nested.array[1][1]#<br>
copied.nested.array[1][2] = #copied.nested.array[1][2]#<br>
</cfoutput>
<hr>
<b>Duplicated structure values should remain unchanged.</b><br>
<cfoutput>
// Simple values <br>
duplicated.nested.number = #duplicated.nested.number#<br>
duplicated.nested.string = #duplicated.nested.string#<br>
// Array values <br>
duplicated.nested.array[1][1] = #duplicated.nested.array[1][1]#<br>
duplicated.nested.array[1][2] = #duplicated.nested.array[1][2]#<br>
</cfoutput>
StructCount

Description
Counts the keys in a structure.

Returns
A number; if structure does not exist, throws an exception.

Category
Structure functions

Function syntax
StructCount(structure)

See also

History
ColdFusion MX: Changed behavior: this function can be used on XML objects.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>structure</td>
<td>Structure to access</td>
</tr>
</tbody>
</table>

Example
<!--- This view-only example shows use of StructCount. --->
<p>This file is similar to addemployee.cfm, which is called by StructNew, StructClear, and StructDelete. To test, copy StructCount function to appropriate place in addemployee.cfm.

<!---
<cfswitch expression="#ThisTag.ExecutionMode#">
<cfcase value="start">
<cfif StructIsEmpty(attributes.EMPINFO)>
<cfoutput>Error. No employee data was passed.</cfoutput>
<cfexit method="ExitTag">
<cfelse>
<cfquery name="AddEmployee" datasource="cfdocexamples"> INSERT INTO Employees (FirstName, LastName, Email, Phone, Department) VALUES 
<cfoutput> ( 
'"#StructFind(attributes.EMPINFO, "firstname")#', 
'"#StructFind(attributes.EMPINFO, "lastname")#', 
'"#StructFind(attributes.EMPINFO, "email")#', 
'"#StructFind(attributes.EMPINFO, "phone")#', 
'"#StructFind(attributes.EMPINFO, "department")#' ) 
</cfoutput>
</cfquery>
<cfswitch> --->

</cfquery>
</cfif>
<cfoutput><hr>Employee Add Complete
<p>#StructCount(attributes.EMPINFO)# columns added.</cfoutput>
</cfcase>
</cfswitch> --->
**StructDelete**

**Description**

Removes an element from a structure.

**Returns**

Boolean value. The value depends on the `indicatenotexisting` parameter value.

**Category**

Structure functions

**Function syntax**

```plaintext
StructDelete(structure, key [, indicatenotexisting ])
```

**See also**


**History**

ColdFusion MX: Changed behavior: this function can be used on XML objects.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>structure</td>
<td>Structure or a variable that contains one. Contains element to remove.</td>
</tr>
<tr>
<td>key</td>
<td>Element to remove.</td>
</tr>
<tr>
<td>indicatenotexisting</td>
<td>• True: returns Yes if key exists; No if it does not.</td>
</tr>
<tr>
<td></td>
<td>• False: returns Yes regardless of whether key exists. Default.</td>
</tr>
</tbody>
</table>

**Example**

```plaintext
<cfoutput>
Field to be deleted: #form.field#
</cfoutput>
<cfscript>
employee = StructNew();
StructInsert(employee, "firstname", firstname);
StructInsert(employee, "lastname", lastname);
StructInsert(employee, "email", email);
</cfscript>
```

858  Chapter 3: ColdFusion Functions
StructInsert(employee, "phone", phone);
StructInsert(employee, "department", department);
</CFScript>

Before deletion, employee structure looks like this:
<cfdump var="#employee#">
<br>
<cfset rc = StructDelete(employee, "#form.field#", "True")>
<cfoutput>
Did I delete the field "#form.field#"? The code indicates: #rc#<br>
The structure now looks like this:<br>
<cfdump var="#employee#"
<br>
</cfoutput>
</cfif>
</cff>
<br><br>
<form method="post" action="#CGI.Script_Name#">
<p>Select the field to be deleted:<br>
<select name = "field">
<option value = "firstname">first name<br>
<option value = "lastname">last name<br>
<option value = "email">email<br>
<option value = "phone">phone<br>
<option value = "department">department<br></select>
<br>
<input type = "submit" name = "Delete" value = "Delete">
</form>
Delete this comment to make this page work --->
**StructFind**

**Description**

Determines the value associated with a key in a structure.

**Returns**

The value associated with a key in a structure; if *structure* does not exist, throws an exception.

**Category**

Structure functions

**Function syntax**

`StructFind(structure, key)`

**See also**


**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>structure</td>
<td>Structure that contains the value to return</td>
</tr>
<tr>
<td>key</td>
<td>Key whose value to return</td>
</tr>
</tbody>
</table>

**Usage**

A structure’s keys are unordered.

**Example**

```coldfusion
<!--- This view-only example shows the use of StructFind. --->
<p>This file is identical to addemployee.cfm, which is called by StructNew, StructClear, and StructDelete. It adds employees. Employee information is passed through the employee structure (EMPINFO attribute). In UNIX, you must also add the Emp_ID.

<!---
<cfswitch expression = "#ThisTag.ExecutionMode#">  
<cfcase value = "start">  
<cfif StructIsEmpty(attributes.EMPINFO)>  
<cfoutput>Error. No employee data was passed.</cfoutput>  
<cfexit method = "ExitTag">  
<cfelse>  
<cfquery name = "AddEmployee" datasource = "cfdocexamples">  
INSERT INTO Employees (FirstName, LastName, Email, Phone, Department) VALUES  
<cfoutput>  
('#StructFind(attributes.EMPINFO, "firstname")#',  
'#StructFind(attributes.EMPINFO, "lastname")#',  
'#StructFind(attributes.EMPINFO, "email")#',  
'#StructFind(attributes.EMPINFO, "phone")#',  
'#StructFind(attributes.EMPINFO, "department")#')  
</cfoutput>
<!---
</cfelse>
</cfswitch>
```
**StructFindKey**

**Description**

Searches recursively through a substructure of nested arrays, structures, and other elements, for structures whose values match the search key in the `value` parameter.

**Returns**

An array that contains structures with values that match `value`.

**Category**

Structure functions

**Function syntax**

```
StructFindKey(top, value, scope)
```

**See also**


**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>top</td>
<td>ColdFusion object (structure or array) from which to start search. This parameter requires an object, not a name of an object.</td>
</tr>
<tr>
<td>value</td>
<td>String or a variable that contains one for which to search.</td>
</tr>
</tbody>
</table>
| scope     | • one: returns one matching key. Default.  
            • all: returns all matching keys. |

**Usage**

Returns an array that includes one structure for each of the specified values it finds. The fields of each of these structures are:

- **Value**: value held in the found key
- **Path**: string that can be used to reach the found key
- **Owner**: parent object that contains the found key

A structure’s keys are unordered.

**Example**

```
<cfset aResults = StructFindKey( #request#, "bass" )>
```
StructFindValue

Description
Searches recursively through a substructure of nested arrays, structures, and other elements for structures with values that match the search key in the value parameter.

Returns
An array that contains structures with values that match the search key value. If none are found, returns an array of size 0.

Category
Structure functions

Function syntax
StructFindValue( top, value [, scope])

See also

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>top</td>
<td>ColdFusion object (a structure or an array) from which to start search. This parameter requires an object, not a name of an object.</td>
</tr>
<tr>
<td>value</td>
<td>String or a variable that contains one for which to search. The type must be a simple object. Arrays and structures are not supported.</td>
</tr>
</tbody>
</table>
| scope     | • one: function returns one matching key (default).  
• all: function returns all matching keys. |

Usage
The fields of each structure in the returned array are:
• Key: name of the key in which the value was found
• Path: string which could be used to reach the found key
• Owner: parent object that contains the found key
A structure’s keys are unordered.

Example
<cfset aResults = StructFindValue( #request#, "235" )>
**StructGet**

**Description**

Gets a structure(s) from a specified path.

**Returns**

An alias to the variable in the `pathDesired` parameter. If necessary, `StructGet` creates structures or arrays to make `pathDesired` a valid variable "path."

**Category**

Structure functions

**Function syntax**

`StructGet(pathDesired)`

**See also**


**History**

ColdFusion MX:

- Changed behavior: this function can be used on XML objects.
- Changed behavior: if there is no structure or array present in `pathDesired`, this function creates structures or arrays to make `pathDesired` a valid variable "path."

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>pathDesired</code></td>
<td>Pathname of variable that contains structure or array from which ColdFusion retrieves structure.</td>
</tr>
</tbody>
</table>

**Usage**

You can inadvertently create invalid structures using this function. For example, if array notation is used to expand an existing array, the specified new element is created, regardless of the type currently held in the array.

**Example**

```coldfusion
test = StructGet("dog.myscope.test")
<cfif NOT IsDefined("dog")>
    Dog is not defined<br>
</cfif>
<cfif NOT IsDefined("dog.myscope")>
    Dog.Myscope is not defined<br>
</cfif>
<cfif NOT IsDefined("dog.myscope.test")>
    Dog.Myscope.Test is not defined<br>
</cfif>
```
<cfif NOT Isdefined("dog.myscope.test.foo")>
  Dog.Myscope.Test.Foo is not defined<br>
</cfif>
<cfoutput>
  #dog.myscope.test.foo#<br>
</cfoutput>
<cfset test = StructGet( "request.myscope[1].test" )>
<cfset test.foo = 2>
<cfoutput>
  #request.myscope[1].test.foo#<br>
</cfoutput>
<cfset test = StructGet( "request.myscope[1].test[2]" )>
<cfset test.foo = 3>
<cfoutput>
  #request.myscope[1].test[2].foo#<br>
</cfoutput>
StructInsert

Description

Inserts a key-value pair into a structure.

Returns

True, upon successful completion. If `structure` does not exist, or if `key` exists and `allowoverwrite = "False"`, ColdFusion throws an exception.

Category

Structure functions

Function syntax

`StructInsert(structure, key, value [, allowoverwrite ])`

See also


History

ColdFusion MX: Changed behavior: this function can be used on XML objects.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>structure</td>
<td>Structure to contain the new key-value pair.</td>
</tr>
<tr>
<td>key</td>
<td>Key that contains the inserted value.</td>
</tr>
<tr>
<td>value</td>
<td>Value to add.</td>
</tr>
<tr>
<td>allowoverwrite</td>
<td>Optional. Whether to allow overwriting a key. The default value is False.</td>
</tr>
</tbody>
</table>

Usage

A structure’s keys are unordered.

Example

```xml
<h1>Add New Employees</h1>
<!--- Establish params for first time through --->
<cfparam name = "FORM.firstname" default = "">
<cfparam name = "FORM.lastname" default = "">
<cfparam name = "FORM.email" default = "">
<cfparam name = "FORM.phone" default = "">
<cfparam name = "FORM.department" default = "">

<cfif FORM.firstname EQ "">
  <p>Please fill out the form.</p>
<cfelse>
  <cfoutput>
    <CFScript>
      employee = StructNew();
      StructInsert(employee, "firstname", FORM.firstname);
    </CFScript>
  </cfoutput>
</cfelse>
```

866 Chapter 3: ColdFusion Functions
StructInsert(employee, "lastname", FORM.lastname);
StructInsert(employee, "email", FORM.email);
StructInsert(employee, "phone", FORM.phone);
StructInsert(employee, "department", FORM.department);
</CFScript>

<p>First name is #StructFind(employee, "firstname")#</p>
<p>Last name is #StructFind(employee, "lastname")#</p>
<p>EMail is #StructFind(employee, "email")#</p>
<p>Phone is #StructFind(employee, "phone")#</p>
<p>Department is #StructFind(employee, "department")#</p>
</cfoutput>

<!--- Call the custom tag that adds employees --->
<CF_ADDEMPLOYEE EMPINFO = "#employee#"/>
</cfif>

<!--- Form for adding employees --->
<form action = "structinsert.cfm">
    <p>First Name:&nbsp; <input name = "firstname" type = "text" hspace = "30" maxlength = "30"></p>
    <p>Last Name:&nbsp; <input name = "lastname" type = "text" hspace = "30" maxlength = "30"></p>
    <p>EMail:&nbsp; <input name = "email" type = "text" hspace = "30" maxlength = "30"></p>
    <p>Phone:&nbsp; <input name = "phone" type = "text" hspace = "20" maxlength = "20"></p>
    <p>Department:&nbsp; <input name = "department" type = "text" hspace = "30" maxlength = "30"></p>
    <p><input type = "submit" value = "OK"></p>
</form>
**StructIsEmpty**

**Description**

Determines whether a structure contains data.

**Returns**

True, if `structure` is empty; if `structure` does not exist, ColdFusion throws an exception.

**Category**

Decision functions, Structure functions

**Function syntax**

```
StructIsEmpty(structure)
```

**See also**


**History**

ColdFusion MX: Changed behavior: this function can be used on XML objects.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>structure</td>
<td>Structure to test</td>
</tr>
</tbody>
</table>

**Example**

```html
<!--- This example illustrates use of StructIsEmpty. --->
<p>This file is identical to addemployee.cfm, which is called by StructNew, StructClear, and StructDelete. It adds employees. Employee information is passed through employee structure (EMPINFO attribute). In UNIX, you must also add the Emp_ID.

```cfswitch expression = "#ThisTag.ExecutionMode#"
```cfcase value = "start"
```cfif StructIsEmpty(attributes.EMPINFO)
```cfoutput>Error. No employee data was passed.</cfoutput>
```cfexit method = "ExitTag"
```cfelse
```<!--- Add the employee; In UNIX, you must also add the Emp_ID --->
```cfquery name = "AddEmployee" datasource = "cfdocexamples"
```INSERT INTO Employees
  (FirstName, LastName, Email, Phone, Department)
VALUES
  ('#StructFind(attributes.EMPINFO, "firstname")#',
  '#StructFind(attributes.EMPINFO, "lastname")#',
  '#StructFind(attributes.EMPINFO, "email")#',
  '#StructFind(attributes.EMPINFO, "phone")#',
  '#StructFind(attributes.EMPINFO, "department")#')
```"
</cfoutput>
</cfquery>
</cfif>
</cfoutput><hr>Employee Add Complete</cfoutput>
</cfcase>
</cfswitch>
StructKeyArray

Description
Finds the keys in a ColdFusion structure.

Returns
An array of keys; if structure does not exist, ColdFusion throws an exception.

Category
Structure functions

Function syntax
StructKeyArray(structure)

See also

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>structure</td>
<td>Structure from which to extract a list of keys</td>
</tr>
</tbody>
</table>

Usage
A structure's keys are unordered.

Example

```coldfusion
<!--- Shows StructKeyArray function to copy keys from a structure to an array. Uses StructNew to create structure and fills its fields with the information the user enters in the form fields. --->
<h3>StructKeyArray Example</h3>

<!--- Extracting the Keys from the Employee Structure--->

<!--- Create structure. Check whether Submit was pressed. If so, define fields in employee structure with user entries on form. ----->
<cfset employee = StructNew()>
<cfif Isdefined("Form.Submit")>
    <cfif Form.Submit is "OK">
        <cfset employee.firstname = FORM.firstname>
        <cfset employee.lastname = FORM.lastname>
        <cfset employee.email = FORM.email>
        <cfset employee.phone = FORM.phone>
        <cfset employee.company = FORM.company>
    <cfelseIf Form.Submit is "Clear">
        <cfset rc = StructClear(employee)>
    </cfif>
</cfif>
<p>This example uses the StructNew function to create a structure called "employee" that supplies employee info. Its fields are filled by the form. After you enter employee information in structure, the example uses StructKeyArray function to copy all of the keys from the structure into an array. </p>
```
<hr size = "2" color = "#0000A0">
<form action = "structkeyarray.cfm">
<table cellspacing = "2" cellpadding = "2" border = "0">
<tr><td>First Name:</td><td><input name = "firstname" type = "text" value = "" hspace = "30" maxlength = "30"></td></tr>
<tr><td>Last Name:</td><td><input name = "lastname" type = "text" value = "" hspace = "30" maxlength = "30"></td></tr>
<tr><td>EMail</td><td><input name = "email" type = "text" value = "" hspace = "30" maxlength = "30"></td></tr>
<tr><td>Phone:</td><td><input name = "phone" type = "text" value = "" hspace = "20" maxlength = "20"></td></tr>
<tr><td>Company:</td><td><input name = "company" type = "text" value = "" hspace = "30" maxlength = "30"></td></tr>
<tr><td><input type = "submit" name = "submit" value = "OK"></td></tr>
</tbody></table>
</form>
</body>
</html>
**StructKeyExists**

**Description**
Determines whether a specific key is present in a structure.

**Returns**
True, if `key` is in `structure`; if `structure` does not exist, ColdFusion throws an exception.

**Category**
Decision functions, Structure functions

**Function syntax**
```coldfusion
StructKeyExists(structure, "key")
```

**See also**

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>structure</td>
<td>Name of structure to test</td>
</tr>
<tr>
<td>key</td>
<td>Key to test</td>
</tr>
</tbody>
</table>

**Usage**
This function can sometimes be used in place of the `IsDefined` function, when working with the URL and Form scopes, which are structures. The following pieces of code are equivalent:

```coldfusion
cfif IsDefined("Form.JediMaster")>
  <cfif StructKeyExists(Form."JediMaster")>
A structure's keys are unordered.
```

**Example**
```coldfusion
<!--- This example shows the use of StructKeyExists. --->
<p>This file is similar to addemployee.cfm, which is called by StructNew, StructClear, and StructDelete. To test, copy the &LT;CFELSEIf&GT; statement to the appropriate place in addemployee.cfm. It is a custom tag to add employees. Employee information is passed through the employee structure (the EMPINFO attribute). In UNIX, you must also add the Emp_ID.</p>
<cfswitch expression = "#ThisTag.ExecutionMode#">
  <cfcase value = "start">
    <cfif StructIsEmpty(attributes.EMPINFO)>
      <cfoutput>Error. No employee data was passed.</cfoutput>
      <cfexit method = "ExitTag">
    </cfif>
    <cfelseIf NOT StructKeyExists(attributes.EMPINFO, "department")>
      <cfscript>StructUpdate(attributes.EMPINFO, "department", "Unassigned");
    </cfscript>
    <cfelse>
      <cfexit method = "ExitTag">
    </cfelse>
</cfswitch>
```

872 Chapter 3: ColdFusion Functions
**StructKeyList**

**Description**

Extracts keys from a ColdFusion structure.

**Returns**

A list of keys; if *structure* does not exist, ColdFusion throws an exception.

**Category**

Structure functions

**Function syntax**

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

**See also**


**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>structure</td>
<td>Structure from which to extract a list of keys.</td>
</tr>
<tr>
<td>delimiter</td>
<td>Optional. Character that separates keys in list. The default value is comma.</td>
</tr>
</tbody>
</table>

**Usage**

A structure’s keys are unordered.

**Example**

```
<!--- This example shows how to use StructKeyList to list the keys in a structure. It uses StructNew function to create structure and fills it with information user enters in form fields. --->
<!--- This section creates structure and checks whether Submit has been pressed. If so, code defines fields in the employee structure with what the user entered in the form. --->
<cfset employee = StructNew()>
<cfif Isdefined("Form.Submit")>
  <cfif Form.Submit is "OK">
    <cfset employee.firstname = FORM.firstname>
    <cfset employee.lastname = FORM.lastname>
    <cfset employee.email = FORM.email>
    <cfset employee.phone = FORM.phone>
    <cfset employee.company = FORM.company>
  </cfelseIf Form.Submit is "Clear">
  <cfset rc = StructClear(employee)>
  </cfif>
</cfif>
<html>
<title>StructKeyList Function</title>
<head>
<title>StructKeyList Function</title>
</head>
```
<h3>StructKeyList Function</h3>
<h3>Listing the Keys in the Employees Structure</h3>

This example uses StructNew function to create structure "employee" that supplies employee information. The fields are filled with the contents of the following form.

After you enter employee information into structure, example uses StructKeyList function to list keys in structure.

This code does not show how to insert information into a database. See cfquery for more information about database insertion.

```
<form action = "structkeylist.cfm">
<table cellspacing = "2" cellpadding = "2" border = "0">
<tr>
<td>First Name:</td>
<td><input name = "firstname" type = "text" value = "" hspace = "30" maxlength = "30"></td>
</tr>
<tr>
<td>Last Name:</td>
<td><input name = "lastname" type = "text" value = "" hspace = "30" maxlength = "30"></td>
</tr>
<tr>
<td>Email:</td>
<td><input name = "email" type = "text" value = "" hspace = "30" maxlength = "30"></td>
</tr>
<tr>
<td>Phone:</td>
<td><input name = "phone" type = "text" value = "" hspace = "20" maxlength = "20"></td>
</tr>
<tr>
<td>Company:</td>
<td><input name = "company" type = "text" value = "" hspace = "30" maxlength = "30"></td>
</tr>
<tr>
<td><input type = "submit" name = "submit" value = "OK"></td>
</tr>
</table>
</form>
```

If fields are correct, we can process new employee information. If they are not correct, consider rewriting application.

```
StructNew

Description
Creates a structure.

Returns
A structure.

Category
Structure functions

Function syntax
StructNew()

See also

Parameters
None

Example
<!--- Shows StructNew. Calls CF_ADDEMPLOYEE, which uses the | addemployee.cfm file to add employee record to database. --->
<h1>Add New Employees</h1>
<cfparam name = "FORM.firstname" default = "">
<cfparam name = "FORM.lastname" default = "">
<cfparam name = "FORM.email" default = "">
<cfparam name = "FORM.phone" default = "">
<cfparam name = "FORM.department" default = "">
<cfif FORM.firstname EQ "">
<p>Please fill out the form.
<cfelse>
<cfoutput>
<cfscript>
employee = StructNew();
StructInsert(employee, "firstname", FORM.firstname);
StructInsert(employee, "lastname", FORM.lastname);
StructInsert(employee, "email", FORM.email);
StructInsert(employee, "phone", FORM.phone);
StructInsert(employee, "department", FORM.department);
</cfscript>
<p>First name is #StructFind(employee, "firstname")#
<p>Last name is #StructFind(employee, "lastname")#
<p>Email is #StructFind(employee, "email")#
<p>Phone is #StructFind(employee, "phone")#
<p>Department is #StructFind(employee, "department")#
</cfoutput>
</cfif>
<!--- Call the custom tag that adds employees --->
<CF_ADDEMPLOYEE EMPINFO = "#employee#">
</cfif>
**StructSort**

**Description**

Returns a sorted array of the top level keys in a structure. Sorts using alphabetic or numeric sorting, and can sort based on the values of any structure element.

**Returns**

An array of top-level key names (strings), sorted by the value of the specified subelement.

**Category**

*Structure functions*

**Function syntax**

```coldfusion
StructSort(base, sortType, sortOrder, pathToSubElement)
```

**See also**

*Structure functions*; “Structure functions” in Chapter 5, “Using Arrays and Structures,” in *ColdFusion MX Developer’s Guide*

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>base</td>
<td>A ColdFusion struct with one field (an associative array).</td>
</tr>
<tr>
<td>sortType</td>
<td>• numeric</td>
</tr>
<tr>
<td></td>
<td>• text: case-sensitive (all lowercase letters precede the first uppercase letter). Default.</td>
</tr>
<tr>
<td></td>
<td>• textnocase</td>
</tr>
<tr>
<td>sortOrder</td>
<td>• asc: ascending (a to z) sort order. Default.</td>
</tr>
<tr>
<td></td>
<td>• desc: descending (z to a) sort order</td>
</tr>
<tr>
<td>pathToSubElement</td>
<td>String or a variable that contains one. Path to apply to each top-level key, to reach element value by which to sort. The default value is nothing (top-level entries sorted by their own values).</td>
</tr>
</tbody>
</table>

**Usage**

The `pathToSubElement` string does not support array notation, and only supports substructures of structures.

This function does not sort or change the structure.

**Example**

```coldfusion
<cfscript>
    salaries = StructNew() ;
    employees = StructNew() ;
    departments = StructNew() ;
    for ( i=1; i lt 6; i=i+1 )
    |
        salary = 120000 - i*10000 ;
        salaries["employee$i"] = salary ;
</cfscript>
```
employee = StructNew();
employee["salary"] = salary;
// employee.salary = salary;
employees["employee#i#"] = employee;

departments["department#i#"] = StructNew();
departments["department#i#"].boss = employee;
</cfscript>

<cfoutput>
<p>list of employees based on the salary (text search): <br>
1) #ArrayToList( StructSort( salaries ) )#<br>
2) #ArrayToList( StructSort( salaries, "text", "ASC" ) )#<br>
3) #ArrayToList( StructSort( salaries, "textnocase", "ASC" ) )#<br>
4) #ArrayToList( StructSort( salaries, "text", "DESC" ) )#<br>
<br></p>

<p>list of employees based on the salary (numeric search): <br>
5) #ArrayToList( StructSort( salaries, "numeric", "ASC" ) )#<br>
6) #ArrayToList( StructSort( salaries, "numeric", "DESC" ) )#<br>
<br></p>

<p>list of employees based on the salary (subfield search): <br>
7) #ArrayToList( StructSort( employees, "numeric", "DESC", "salary" ) )#<br>
8) #ArrayToList( StructSort( employees, "text", "ASC", "salary" ) )#<br>
<br></p>

<p>list of departments based on the salary (sub-sub-field search): <br>
9) #ArrayToList( StructSort( departments, "text", "ASC", "boss.salary" ) )#<br></p>
</cfoutput>

<!--- add an invalid element and test that it throws an error --->
<p>
<cfset employees[ "employee4" ] = StructNew()>
<cftry>
  <cfset temp = StructSort( employees, "text", "ASC", "salary" )>
  <cfoutput>We have a problem - this was supposed to throw an exception!</cfoutput>
</cfoutput>
<cfcatch type="any">
  <cfoutput>
  ERROR: <b>This error was expected!</b><br>
  #cfcatch.message# - #cfcatch.detail#<br>
  </cfoutput>
</cfcatch>
<br>
</cftry>
</p>
StructUpdate

Description
Updates a key with a value.

Returns
True, on successful execution; if the structure does not exist, ColdFusion throws an error.

Category
Structure functions

Function syntax
StructUpdate(structure, key, value)

See also

History
ColdFusion MX: Changed behavior: this function can be used on XML objects.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>structure</td>
<td>Structure to update</td>
</tr>
<tr>
<td>key</td>
<td>Key, the value of which to update</td>
</tr>
<tr>
<td>value</td>
<td>New value</td>
</tr>
</tbody>
</table>

Example
<!--- This example shows the use of StructUpdate. --->
This file is similar to addemployee.cfm, which is called by StructNew, StructClear, and StructDelete. To test this file, copy the &LT;CFFELSEIF&GT; statement to the appropriate place in addemployee.cfm. It is an example of a custom tag used to add employees. Employee information is passed through the employee structure (the EMPINFO attribute). In UNIX, you must also add the Emp_ID.

<cfswitch expression = "#ThisTag.ExecutionMode#">
<cfcase value = "start">
  <cfif StructIsEmpty(attributes.EMPINFO)>
    <cfoutput>Error. No employee data was passed.</cfoutput>
    <cfexit method = "ExitTag">
  <cfelseif StructFind(attributes.EMPINFO, "department") EQ "">
    <cfscript>
      StructUpdate(attributes.EMPINFO, "department", "Unassigned");
    </cfscript>
    <cfexit method = "ExitTag">
  <cfelse>
    Parameter Description
    structure Structure to update
    key Key, the value of which to update
    value New value

Example
<!--- This example shows the use of StructUpdate. --->
This file is similar to addemployee.cfm, which is called by StructNew, StructClear, and StructDelete. To test this file, copy the &LT;CFFELSEIF&GT; statement to the appropriate place in addemployee.cfm. It is an example of a custom tag used to add employees. Employee information is passed through the employee structure (the EMPINFO attribute). In UNIX, you must also add the Emp_ID.

<cfswitch expression = "#ThisTag.ExecutionMode#">
<cfcase value = "start">
  <cfif StructIsEmpty(attributes.EMPINFO)>
    <cfoutput>Error. No employee data was passed.</cfoutput>
    <cfexit method = "ExitTag">
  <cfelseif StructFind(attributes.EMPINFO, "department") EQ "">
    <cfscript>
      StructUpdate(attributes.EMPINFO, "department", "Unassigned");
    </cfscript>
    <cfexit method = "ExitTag">
  <cfelse>
    Parameter Description
    structure Structure to update
    key Key, the value of which to update
    value New value

Example
<!--- This example shows the use of StructUpdate. --->
This file is similar to addemployee.cfm, which is called by StructNew, StructClear, and StructDelete. To test this file, copy the &LT;CFFELSEIF&GT; statement to the appropriate place in addemployee.cfm. It is an example of a custom tag used to add employees. Employee information is passed through the employee structure (the EMPINFO attribute). In UNIX, you must also add the Emp_ID.
**Tan**

**Description**
Calculates the tangent of an angle that is entered in radians.

**Returns**
A number; the tangent of an angle.

**Category**
Mathematical functions

**Function syntax**
```
Tan(number)
```

**See also**
Atn, Cos, Atn, Sin, ASin, Pi

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>number</td>
<td>Angle, in radians, for which to calculate the tangent.</td>
</tr>
</tbody>
</table>

**Usage**
To convert degrees to radians, multiply degrees by $\pi/180$. To convert radians to degrees, multiply radians by $180/\pi$.

**Note:** Because the function uses floating point arithmetic, it can return a very small number (such as $6.12323399574E-017$) for angles that should produce 0 and can return a very large number (such as $1.63312393532E+016$) for infinity or not a number. To test for a 0 value, check whether the value is less than $0.0000000000000001$. To test for an infinite value, check whether the value is more than $1E15$.

**Example**
```
<h3>Tan Example</h3>
<!---- Calculate tangent if form has been submitted --->
<cfif IsDefined("FORM.tanNum")>
<!---- Make sure input is a number --->
<cfif IsNumeric(#FORM.tanNum#)>
<!---- Convert degrees to radians, call the Tan function. --->
<cfset tanValue=#Tan((Form.tanNum * PI()) / 180)#>
<!---- 0.0000000000000001 is the function's precision limit.
If absolute value of returned value is less, set result to 0 --->
<cfif Abs(tanValue) LT 0.0000000000000001>
<cfset tanValue=0>
</cfif>
</cfif>
</cfoutput>
Tan(#FORM.tanNum#) = #tanValue#<br><br>
<cfelse>
<!---- If input is not a number, show an error message --->
<h4>You must enter a numeric angle in degrees.</h4>
```

---

**Tan** 879
Enter an angle in degrees to get its tangent:
<input type="Text" name="tanNum" size="15">
<input type="Submit" name=""></form>
**TimeFormat**

**Description**

Formats a time value using U.S. English time formatting conventions.

**Returns**

A custom-formatted time value. If no mask is specified, returns a time value using the `hh:mm tt` format. For international time formatting, see `LSTimeFormat`.

**Category**

Date and time functions, Display and formatting functions

**Function syntax**

`TimeFormat(time [, mask ])`

**See also**

`CreateTime, Now, ParseDateTime, LSTimeFormat, DateFormat`

**History**

ColdFusion MX 6.1: Added the mask character L or l to represent milliseconds.

ColdFusion MX:

- Changed the way extra characters are processed: this function processes extra characters within the `mask` value differently than in earlier releases, as follows:
  - ColdFusion 5 and earlier: the function returns the time format and an apostrophe-delimited list of the extra characters. For example, `TimeFormat(Now(), "hh:mm:ss dog")` returns `8:17:23 d'o'g`.
  - ColdFusion MX: the function returns the time format and the extra characters. For example, for the call above, it returns `8:17:23 dog`.

- If the extra characters are single-quoted (for example, `hh:mm:ss 'dog'`), ColdFusion 5 and ColdFusion MX return the time format and the extra characters: `8:17:23 dog`.

- Added support for the following `mask` parameter options: `short, medium, long, and full`.
# Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>time</td>
<td>A date/time value or string to convert</td>
</tr>
</tbody>
</table>
| mask      | Masking characters that determine the format:  
  • h: hours; no leading zero for single-digit hours (12-hour clock)  
  • hh: hours; leading zero for single-digit hours (12-hour clock)  
  • H: hours; no leading zero for single-digit hours (24-hour clock)  
  • HH: hours; leading zero for single-digit hours (24-hour clock)  
  • m: minutes; no leading zero for single-digit minutes  
  • mm: minutes; a leading zero for single-digit minutes  
  • s: seconds; no leading zero for single-digit seconds  
  • ss: seconds; leading zero for single-digit seconds  
  • I or L: milliseconds. I gives 3 digits. L gives 2 digits.  
  • t: one-character time marker string, such as A or P  
  • tt: multiple-character time marker string, such as AM or PM  
  • short: equivalent to h:mm tt  
  • medium: equivalent to h:mm:ss tt  
  • long: medium followed by three-letter time zone; as in, 2:34:55 PM EST  
  • full: same as long |

## Usage

When passing a date/time value as a string, you must enclose it in quotation marks. Otherwise, it is interpreted as a number representation of a date/time object.

Database query results for date and time values can vary in sequence and formatting unless you use functions to format the results. To ensure that dates and times display with appropriate formatting, and that users of your ColdFusion application are not confused by dates and times displayed, Macromedia recommends that you use the `DateFormat` and `TimeFormat` functions to format date and time values from queries. For more information and examples, see TechNote 22183, “ColdFusion Server (5 and 4.5.x) with Oracle: Formatting Date and Time Query Results,” at www.coldfusion.com/Support/KnowledgeBase/SearchForm.cfm.

## Example

```cfml
<cfset todayDate = #Now()#>
<body>
<h3>TimeFormat Example</h3>
<p>Today's date is <cfoutput>#todayDate#</cfoutput>.</p>
<p>Using Timeformat, we can display the value in different ways:</cfoutput>
</body>
<ul>
  <li>#TimeFormat(todayDate)#
  <li>#TimeFormat(todayDate, "hh:mm:ss")#
  <li>#TimeFormat(todayDate, "hh:mm:ssst")#
  <li>#TimeFormat(todayDate, "hh:mm:ssstt")#
  <li>#TimeFormat(todayDate, "HH:mm:ss")#
</ul>
<p>To generate a standard ISO 8601 W3C Date and Time string like 1997-07-16T19:20, concatenate a DateFormat function, the character T, and a</p>
TimeFormat function.
For example: dateformat(now(), "yyyy-mm-dd")#T#TimeFormat(now(), "HH:mm:ss")
produces:
<cfoutput>#dateformat(now(), "yyyy-mm-dd")#T#TimeFormat(now(), "HH:mm:ss")#</cfoutput>
</body>
ToBase64

Description

Calculates the Base64 representation of a string or binary object. The Base64 format uses printable characters, allowing binary data to be sent in forms and e-mail, and stored in a database or file.

Returns

The Base64 representation of a string or binary object.

Category

Conversion functions, String functions

Function syntax

ToBase64(string or binary_object[, encoding])

See also

• BinaryEncode for conversion of binary data to base64
• cffile for information about loading and reading binary data
• cfwddx for information about serializing and deserializing binary data
• IsBinary and ToBinary for checking for binary data and converting a Base64 object to binary format

History

ColdFusion MX: Added the encoding parameter.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>string or binary_object</td>
<td>A string, the name of a string, or a binary object.</td>
</tr>
</tbody>
</table>
| encoding           | For a string, defines how characters are represented in a byte array. The following list includes commonly used values:  
  • utf-8  
  • iso-8859-1  
  • windows-1252  
  • us-ascii  
  • shiftjis  
  • iso-2022-jp  
  • euc-jp  
  • euc-kr  
  • big5  
  • euc-cn  
  • utf-16  
  For more information on character encoding, see: www.w3.org/International/O-charset.html.  
  The default value is the encoding of the page on which the function is called. See cfcontent. For a binary object, this parameter is ignored. |
Usage

Macromedia recommends that you use the `BinaryEncode` function to convert binary data to Base64-encoded data in all new applications.

```cfml
<h3>ToBase64 Example</h3>
<!--- Initialize data. ---->
<cfset charData = ">
<!--- Create string of ASCII characters (32-255); concatenate them --->
<cfloop index = "data" from = "32" to = "255">
    <cfset ch = chr(data)>
    <cfset charData = charData & ch>
</cfloop>
<p>The following string is the concatenation of all characters (32 to 255) from the ASCII table.<br>
<cfoutput>#charData#</cfoutput></p>
<!--- Create a Base64 representation of this string. ---->
<cfset data64 = toBase64(charData)>

<!--- Convert string to binary. ------->
<cfset binaryData = toBinary(data64)>
<!--- Convert binary back to Base64. ---->
<cfset another64 = toBase64(binaryData)>
<!--- Compare another64 with data64 to ensure that they are equal. ---->
<cfif another64 eq data64>
    <h3>Base64 representations are identical.</h3>
<cfelse>
    <h3>Conversion error.</h3>
</cfif>
```

ToBase64 885
ToBinary

**Description**
Calculates the binary representation of Base64-encoded data.

**Returns**
The binary representation of Base64-encoded data.

**Category**
Conversion functions, String functions

**Function syntax**
```
ToBinary(string_in_Base64 or binary_value)
```

**See also**
- `BinaryDecode` for conversion of binary-encoded data, including Base64, to binary data
- `cffile` for information about loading and reading binary data
- `cfwddx` for information about serializing and deserializing binary data
- `IsBinary` and `ToBase64` for checking format and converting to Base64
- `Len` for determining the length of a binary object

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>string_in_Base64</td>
<td>A string in Base64 format to convert to binary.</td>
</tr>
</tbody>
</table>

**Usage**

Macromedia recommends that you use the `BinaryDecode` function to convert Base64 encoded data to binary data in all new applications.

If you pass a binary value to this function, it returns the input value.

**Example**

```coldfusion
toBinary Example</h3>
<!---- Initialize data. ---->
cfset charData = ">
<!---- Create a string of ASCII characters (32-255); concatenate them. ---->
cfloop index = "data" from = "32" to = "255">
    <cfset ch = chr(data)>
    <cfset charData = charData & ch>
</cfloop>
<p>The following string is the concatenation of all characters (32 to 255) from the ASCII table.<br>
cfoutput>
</p>
<!---- Create a Base64 representation of this string. ---->
cfset data64 = toBase64(charData)
```
<!--- Convert string to binary. ---->
<cfset binaryData = toBinary(data64)>
<!--- Convert binary back to Base64. ---->
<cfset another64 = toBase64(binaryData)>
<!--- Compare another64 with data64 to ensure that they are equal. ---->
<cfif another64 eq data64>
  <h3>Base64 representation of binary data is identical to the Base64
  representation of string data.</h3>
<cfelse>
  <h3>Conversion error.</h3>
</cfif>
**ToScript**

**Description**

Creates a JavaScript or ActionScript expression that assigns the value of a ColdFusion variable to a JavaScript or ActionScript variable. This function can convert ColdFusion strings, numbers, arrays, structures, and queries to JavaScript or ActionScript syntax that defines equivalent variables and values.

**Returns**

A string that contains a JavaScript or ActionScript variable definition corresponding to the specified ColdFusion variable value.

**Category**

Conversion functions, Extensibility functions

**Function syntax**

```
ToScript(cfvar, javascriptvar, outputformat, ASFormat)
```

**See also**

`cfwddx`: Chapter 9, “WDDX JavaScript Objects,” in *ColdFusion MX Developer's Guide*

**History**

ColdFusion MX 7: Added this function.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| cfvar          | A ColdFusion variable. This can contain one of the following:  
|                | • String  
|                | • Number  
|                | • Array  
|                | • Structure  
|                | • Query  
| javascriptvar  | A string that specifies the name of the JavaScript variable that the `ToScript` function creates.  
| outputformat   | Optional. A Boolean value that determines whether to create WDDX (JavaScript) or ActionScript style output for structures and queries:  
|                | • True: creates WDDX-style output (default).  
|                | • False: creates ActionScript-style output.  
| ASFormat       | Optional. A Boolean value that specifies whether to use ActionScript shortcuts in the script:  
|                | • True: creates new Objects and Arrays with ActionScript shortcuts: [] for New Array(), and () for New Object. Using ActionScript shortcuts allows you to pass ActionScript into cfform attributes without triggering ActionScript validation.  
|                | • False: does not use ActionScript shortcuts to create new Objects and new Arrays when generating the script. Instead, generates New Object() and New Array() in the script (default). |
Usage

To use a ColdFusion variable in JavaScript or ActionScript, the ToScript function must be in a cfoutput region and be surrounded by number signs (#). For example, the following code uses the ToScript function to convert a ColdFusion variable to a JavaScript variable:

```cfdsl
<cfset thisString="hello world">
<cfscript type="text/javascript" language="JavaScript">
  <cfoutput>
    var #toScript(thisString, "jsVar")#;
  </cfoutput>
</cfscript>
```

When ColdFusion runs this code, it sends the following to the client:

```javascript
var jsVar = "hello world";
```

An HTML script tag must enclose the JavaScript code. The cfoutput tag does not need to be inside the script block; it can also surround the block.

WDDX-style output generates JavaScript code that creates a WDDXRecordset object, where the key of each record set entry is a column name, and the value of the recordlist entry is an array of the corresponding query column entries, as follows:

```javascript
WDDXQuery = new WddxRecordset();
col0 = new Array();
col0[0] = "John";
col0[1] = "John";
WDDXQuery["firstname"] = col0;
col0 = null;
col1 = new Array();
col1[0] = "Lund";
col1[1] = "Allen";
WDDXQuery["lastname"] = col1;
col1 = null;
```

To use WDDX-style output, you must first load the cf_webroot/CFIDE/scripts/wddx.js script, which defines JavaScript WDDX objects, as in the following line:

```javascript
<script type="text/javascript" src="/CFIDE/scripts/wddx.js script"></script>
```

For more information on WDDX in JavaScript, see Chapter 9, “WDDX JavaScript Objects,” on page 1091.

ActionScript-style output generates code that creates an array of objects, where the array is indexed by row number, and the objects consist of column name - column value pairs, as follows:

```javascript
ActionScriptQuery = new Array();
ActionScriptQuery[0] = new Object();
ActionScriptQuery[0]["firstname"] = "John";
ActionScriptQuery[0]["lastname"] = "Lund";
ActionScriptQuery[1] = new Object();
ActionScriptQuery[1]["firstname"] = "John";
ActionScriptQuery[1]["lastname"] = "Allen";
```
An ActionScript-style array does not require you to include the wddx.js file, and creates a variable that you can use in ActionScript on a Flash format form, for example, in an onChange attribute.

If the outputformat parameter is False, setting ASFormat to True causes ToScript to use the ActionScript shortcut [] in place of New Array() and the shortcut {} in place of New Object(). Using these shortcuts allows you to pass ActionScript into cfform attributes without triggering ActionScript validation. If ASFormat is False, ToScript generates New Array() and New Object() in the script.

Example

The following example shows the results of converting a ColdFusion string, array, and query object to JavaScript variables. It also uses the string and array in JavaScript code.

```html
<h2>ToScript</h2>

<h3>Converting a string variable</h3>
<cfset thisString = "This is a string">
<cfoutput>
  <b>The thisString variable in ColdFusion</b><br>
  #thisString#
</cfoutput>

**The output of ToScript(thisString, "jsVar")**
#ToScript(thisString, "jsVar")#

**In a JavaScript script, convert thisString Variable to JavaScript and output the resulting variable:**

```javascript
var #ToScript(thisString, "jsVar")#;
document.write("jsVar in JavaScript is: " + jsVar);
```

<h3>Converting an array</h3>

<!--- Create and populate a one-dimensional array --->
<cfset myArray=ArrayNew(1)>
<cfloop index="i" from="1" to="4">
  <cfset myArray[i]="This is array element" & i>
</cfloop>

**The ColdFusion myArray Array**

<!--- Write the contents of the myArray variable in ColdFusion --->
<cfloop index="i" from="1" to="#arrayLen(myArray)#">
  myArray[#i#]: #myArray[i]#
</cfloop>

**The output of ToScript(myArray, "jsArray")**
#toScript(myArray, "jsArray")#

**In JavaScript, convert myArray to a JavaScript variable and write it's contents**

```javascript
var #ToScript(myArray, "jsArray")#;
for (i in jsArray)
```

---

890 Chapter 3: ColdFusion Functions
Converting a query

This section converts the following query object to both WDDX format and ActionScript type Javascript objects.

```coldfusion
<cfquery name="thisQuery" datasource="cfdocexamples">
    SELECT FirstName, LastName
    FROM employee
    WHERE FirstName = 'John'
</cfquery>
```

The Query in ColdFusion:
```
<cfoutput>
<cftable query="thisQuery" headerlines="1" colheaders="">
    <cfcol align="left" width="9" header="<b>FirstName</b>" text="#FirstName#">
    <cfcol align="left" width="9" header="<b>LastName</b>" text="#LastName#">
</cftable>
</cfoutput>
```

**JavaScript generated by ToScript(thisQuery, "WDDXQuery"):**
```
#toScript(thisQuery, "WDDXQuery")#
```

**JavaScript generated by ToScript(thisQuery, "ActionScriptQuery", False):**
```
#toScript(thisQuery, "ActionScriptQuery", False)#
```

--- Convert to both WDDX format and ActionScript format ---
```
<script type="text/javascript" language="JavaScript">
    #ToScript(thisQuery, "WDDXQuery")#:;
    #ToScript(thisQuery, "ActionScriptQuery", False)#;
</script>
```

--- For brevity, this example does not use JavaScript query variables ---
```
</cfoutput>
ToString

Description
Converts a value to a string.

Returns
A string.

Category
Conversion functions, String functions

Function syntax
ToString(value[, encoding])

See also
ToBase64, ToBinary, CharsetEncode; Chapter 35, “Using XML and WDDX,” in ColdFusion MX Developer’s Guide

History
ColdFusion MX:
• Changed Unicode support: ColdFusion supports the Java UCS-2 representation of Unicode character values 0–65535. (ColdFusion 5 and earlier releases supported ASCII values 1–255.)
• Added the encoding parameter.
• Added ability to convert an XML document object to a string.
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>Value to convert to a string; can be a simple value such as an integer, a binary object, or an XML document object.</td>
</tr>
</tbody>
</table>
| encoding | The character encoding (character set) of the string. Optional for binary data. Generates an error if used for a simple value or XML document object. The following list includes commonly used values:  
  • utf-8  
  • iso-8859-1  
  • windows-1252  
  • us-ascii  
  • shift_jis  
  • iso-2022-jp  
  • euc-jp  
  • euc-kr  
  • big5  
  • euc-cn  
  • utf-16  
For more information on character encoding, see: www.w3.org/International/O-charset.html. The default value is the encoding of the page on which the function is called. See cfcontent. |

Usage

This function can convert simple values and binary values that do not contain Byte zero. If this function cannot convert a value, it throws an exception. This function can also convert an XML document object to a string XML representation.

Macromedia recommends that you use the CharsetEncode function to convert binary data to a string.

Example

```html
<h3>ToString Example</h3>
<!---- Initialize data. ----->
<cfset charData = "">
<!----- Create string of ASCII characters (32-255) and concatenate them. ----->
<cfloop index = "data" from = "32" to = "255">
  <cfset ch = chr(data)>
  <cfset charData = charData & ch>
</cfloop>
<p>The following string is the concatenation of characters (32 to 255) from the ASCII table.<br>
<cfoutput>#charData#</cfoutput></p>
<!------ Create a Base64 representation of this string. ------>
<cfset data64 = toBase64(#charData#)>
<p>The following string is the Base64 representation of the string.<br>
<cfoutput>#data64#</cfoutput></p>
<!---- Create a binary representation of Base64 data. ----->
```
<cfset dataBinary = toBinary(data64)>

<!-- Create the string representation of the binary data. ------>
<cfset dataString = ToString(dataBinary)>
<p>The following is the string representation of the binary data.<br>
<cfoutput>#dataString#</cfoutput></p>
Trim

Description
Removes leading and trailing spaces from a string.

Returns
A copy of string, after removing leading and trailing spaces.

Category
String functions

Function syntax
Trim(string)

See also
LTrim, RTrim

Parameters
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>string</td>
<td>A string or a variable that contains one</td>
</tr>
</tbody>
</table>

Example

<h3>Trim Example</h3>
<cfif IsDefined("FORM.myText")>
  <cfoutput>
    <pre>
      Your string: "#FORM.myText#"
      Your string: "#Trim(FORM.myText)#"
      (trimmed on both sides)
    </pre>
  </cfoutput>
</cfif>
<form method = "post" action = "trim.cfm">
  <p>Type in some text, and it will be modified by trim to remove leading spaces from the left and right</p>
  <p><input type = "Text" name = "myText" value = " TEST ">
  <input type = "Submit" name = ""></form>
**UCase**

**Description**

Converts the alphabetic characters in a string to uppercase.

**Returns**

A copy of a string, converted to uppercase.

**Category**

String functions

**Function syntax**

UCase(string)

**See also**

LCase

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>string</td>
<td>A string or a variable that contains one</td>
</tr>
</tbody>
</table>

**Example**

```cfc
cif IsDefined("FORM.sampleText")
  
cif FORM.sampleText is not ""
    
    p>Your text. <cfoutput>#FORM.sampleText#</cfoutput>,
    returned in uppercase is <cfoutput>UCase(FORM.sampleText)</cfoutput>.
  
cifelse
    
    p>Please enter some text.
  
  </cfif>
cif

<form action = "ucase.cfm">
  
  p>Enter your sample text, and press "submit" to see the text returned in uppercase:
  <p><input type = "Text" name = "SampleText" value = "sample">

  <input type = "Submit" name = "" value = "submit">

</form>
```
**URLDecode**

**Description**
Decodes a URL-encoded string.

**Returns**
A copy of a string, decoded.

**Category**
Conversion functions, Other functions, String functions

**Function syntax**

```
URLDecode(urlEncodedString[, charset])
```

**See also**

`URLEncodedFormat`; “Tags and functions for globalizing” in Chapter 17, “Developing Globalized Applications,” in *ColdFusion MX Developer’s Guide*

ColdFusion MX 6.1: Changed the default charset: the default charset is the character encoding of the URL scope.

ColdFusion MX:

- Changed Unicode support: ColdFusion supports the Java UCS-2 representation of Unicode character values 0–65535. (Earlier releases supported ASCII values.)
- Added the `charset` parameter.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>urlEncodedString</code></td>
<td>URL-encoded string or a variable that contains one.</td>
</tr>
</tbody>
</table>
| `charset`     | The character encoding in which the URL is encoded. Optional. The following list includes commonly used values:  
  - utf-8  
  - iso-8859-1  
  - windows-1252  
  - us-ascii  
  - shift_jis  
  - iso-2022-jp  
  - euc-jp  
  - euc-kr  
  - big5  
  - euc-cn  
  - utf-16  
  For more information on character encoding, see: [www.w3.org/International/O-charset.html](http://www.w3.org/International/O-charset.html). The default value is the character encoding of the URL scope. See [SetEncoding](SetEncoding). |
Usage

URL encoding formats some characters with a percent sign and the two-character hexadecimal representation of the character. For example, a character whose code is 129 is encoded as %81. A space is encoded with a plus sign.

Query strings in HTTP are always URL-encoded.

Example

This example creates, encodes, and decodes a string that contains ASCII character codes:

```cfc
<cfscript>
    // Build string
    s = "";
    for (c = 1; c le 256; c = c + 1) {
        s = s & chr(c);
    }
    // Encode string and display result
    enc = URLEncodedFormat(s);
    WriteOutput("Encoded string is: '#enc#'.\n    // Decode and compare result with original
    dec = URLDecode(enc);
    if (dec neq s) {
        WriteOutput("Decoded is not the same as encoded.\n    } else {
        WriteOutput("All's quiet on the Western front.\n    }
</cfscript>
```
URLEncodedFormat

Description
Generates a URL-encoded string. For example, it replaces spaces with %20, and non-alphanumeric characters with equivalent hexadecimal escape sequences. Passes arbitrary strings within a URL (ColdFusion automatically decodes URL parameters that are passed to a page).

Returns
A copy of a string, URL-encoded.

Category
Conversion functions, Other functions, String functions

Function syntax
URLEncodedFormat(string [, charset ])

See also
URLDecode; “Tags and functions for globalizing” in Chapter 17, “Developing Globalized Applications,” in ColdFusion MX Developer’s Guide

History
ColdFusion MX 6.1: Changed the default encoding to be the response character encoding.
ColdFusion MX: Added the charset parameter.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>string</td>
<td>A string or a variable that contains one</td>
</tr>
<tr>
<td>charset</td>
<td>The character encoding in which the string is encoded. Optional. The following list includes commonly used values: • utf-8 • iso-8859-1 • windows-1252 • us-ascii • shift_jis • iso-2022-jp • euc-jp • euc-kr • big5 • euc-cn • utf-16 For more information on character encoding, see: <a href="http://www.w3.org/International/O-charset.html">www.w3.org/International/O-charset.html</a>. The default value is the character encoding of the response. See cfcontent.</td>
</tr>
</tbody>
</table>
Usage
URL encoding formats some characters with a percent sign and the two-character hexadecimal representation of the character. For example, a character whose code is 129 is encoded as %81. A space is encoded with a plus sign.

Query strings in HTTP are always URL-encoded.

Example
<h3>URLEncodedFormat Example</h3>
<cfif IsDefined("url.myExample")>
  <p>The url variable url.myExample was passed from the previous link ...
  its value is:
  <br><b>"<cfoutput>#url.myExample#</cfoutput>"</b></p>
</cfif>
<p>This function returns a URL encoded string.  
<cfset s = "My url-encoded string has special characters & other stuff"
<p><A HREF = "urleencodedformat.cfm?myExample=<cfoutput>#URLEncodedFormat(s)#
</cfoutput>">Click me</A>
**URLSessionFormat**

**Description**

Depending on whether a client computer accepts cookies, this function does the following:

- If the client does not accept cookies: automatically appends all required client identification information to a URL
- If the client accepts cookies: does not append information

This function automatically determines which identifiers are required, and sends only the required information. It provides a more secure and robust method for supporting client identification than manually encoding the information in each URL, because it sends only required information, when it is required, and it is easier to code.

**Returns**

A URL; if cookies are disabled for the browser, client and session data are appended.

**Category**

Other functions; Chapter 15, “Maintaining client identity,” in Chapter 15, “Using Persistent Data and Locking,” in *ColdFusion MX Developer's Guide*

**Function syntax**

```
URLSessionFormat(request_URL)
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>request_URL</td>
<td>URL of a ColdFusion page</td>
</tr>
</tbody>
</table>

**Usage**

In the following example, the `cfform` tag posts a request to another page and sends the client identification, if required. If cookie support is detected, the function returns the following:

```
myactionpage.cfm
```

If the detected cookie is not turned on, or cookie support cannot be reliably detected, the function return value is as follows:

```
myactionpage.cfm?jsessionid=xxxx;cfid=xxxx&cftoken=xxxxxxxx
```

**Example**

```
<cfform method="Post"
  action="#URLSessionFormat("MyActionPage.cfm")#"
/>
</cfform>
```
Val

Description

Converts numeric characters that occur at the beginning of a string to a number.

Returns

A number. If conversion fails, returns zero.

Category

Conversion functions, String functions

Function syntax

Val(string)

See also

IsNumeric

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>string</td>
<td>A string or a variable that contains one</td>
</tr>
</tbody>
</table>

Usage

This function works as follows:

- If TestValue = "234A56?7"", Val(TestValue) returns 234.
- If TestValue = "234'5678'9"", Val(TestValue) returns 234.
- If TestValue = "BG234", Val(TestValue) returns the value 0, (not an error).
- If TestValue = "0", Val(TestValue) returns the value 0, (not an error).

Example

<h3>Val Example</h3>
<cfif IsDefined("FORM.theTestValue")>
  <cfif Val(FORM.theTestValue) is not 0>
    <h3>The string <cfoutput>#DE(FORM.theTestValue)#</cfoutput>
    can be converted to a number:<br>
    <cfoutput>#Val(FORM.theTestValue)#</cfoutput></h3>
  </cfif>
  <cfelse>
    <h3>The beginning of the string <cfoutput>#DE(FORM.theTestValue)#</cfoutput>
    cannot be converted to a number</h3>
  </cfelse>
</cfif>

<form action="val.cfm">
  Enter a string, and determine whether its beginning can be evaluated
to a numeric value.

  <p>
    <input type="Text"
      name="TheTestValue"
      value="123Boy">
    <input type="Submit"
value = "Is the beginning numeric?"
name = "">
</form>
ValueList

Description
Inserts a delimiter between each value in an executed query. ColdFusion does not evaluate the arguments.

Returns
A delimited list of the values of each record returned from an executed query.

Category
List functions, Query functions

Function syntax
ValueList(query.column [, delimiter ])

See also
QuotedValueList

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>query.column</td>
<td>Name of an executed query and column. Separate query name and column name with a period.</td>
</tr>
<tr>
<td>delimiter</td>
<td>A delimiter character to separate column data items. The default value is comma (,).</td>
</tr>
</tbody>
</table>

Example
<h3>ValueList Example</h3>

<!--- use the contents of a query to create another dynamically --->
<cfquery name = "GetDepartments" datasource = "cfdocexamples">
SELECT Dept_ID FROM Departments
WHERE Dept_ID IN ('BIOL')
</cfquery>

<cfquery name = "GetCourseList" datasource = "cfdocexamples">
SELECT *
FROM CourseList
WHERE Dept_ID IN ('#GetDepartments.Dept_ID#')
</cfquery>

Value list of all BIOL Course ID's using (--) as the delimiter:<br>
<cfoutput>
#ValueList(GetCourseList.Course_ID,"--")#<br>
</cfoutput>

Value list of all BIOL Course Numbers using (;) as the delimiter:<br>
<cfoutput>
#ValueList(GetCourseList.CorNumber,;";")#<br>
</cfoutput>
Week

Description

From a date/time object, determines the week number within the year.

Returns

An integer in the range 1–53; the ordinal of the week, within the year.

Category

Date and time functions

Function syntax

Week(date)

See also

DatePart

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>date</td>
<td>A date/time object in the range 100 AD–9999 AD.</td>
</tr>
</tbody>
</table>

Usage

When passing date as a string, enclose it in quotation marks. Otherwise, it is interpreted as a number representation of a date.

Example

```xml
<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>
</cfoutput>
</cfif>
```
Wrap

Description
Wraps text so that each line has a specified maximum number of characters.

Returns
String containing the wrapped text.

Category
String functions

Function syntax
Wrap(string, limit[, strip])

See also
cfmail

History
ColdFusion MX 6.1: Added this function

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>string</td>
<td>String or variable that contains one. The text to wrap.</td>
</tr>
<tr>
<td>limit</td>
<td>Positive integer maximum number of characters to allow on a line.</td>
</tr>
<tr>
<td>strip</td>
<td>Boolean specifying whether to remove all existing newline and carriage return characters in the input string with spaces before wrapping the text. The default value is False.</td>
</tr>
</tbody>
</table>

Usage
Inserts line break at the location of the first white space character (such as a space, tab, or new line) before the specified limit on a line. If a line has no whitespace characters before the limit, inserts a line break at the limit. Uses the operating-system specific line break: newline for UNIX, carriage return and newline on Windows.

If you specify the strip parameter, all existing line breaks are removed, so any paragraph formatting is lost.

Use this function to limit the length of text lines, such as text to be included in a mail message. The cfmail and cfmailpart tag wraptex attributes use this function

Example
<h3>Wrap Example</h3>
<cfset inputText="This is an example of a text message that we want to wrap. It is rather long and needs to be broken into shorter lines."
<cfoutput>#Wrap(inputText, 59)#</cfoutput>
WriteOutput

Description

Appends text to the page-output stream.

This function writes to the page-output stream regardless of conditions established by the cfsetting tag.

Category

Other functions, System functions

Function syntax

WriteOutput(string)

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>string</td>
<td>A string or a variable that contains one</td>
</tr>
</tbody>
</table>

Usage

Within the cfquery and cfmail tags, this function does not output to the current page; it writes to the current SQL statement or mail text. Do not use WriteOutput within cfquery and cfmail.

Although you can call this function anywhere within a page, it is most useful inside a cfscript block.

Example

```cfc
<cfscript>
    employee = StructNew();
    StructInsert(employee, "firstname", FORM.firstname);
    StructInsert(employee, "lastname", FORM.lastname);
    StructInsert(employee, "email", FORM.email);
    StructInsert(employee, "phone", FORM.phone);
    StructInsert(employee, "department", FORM.department);
    WriteOutput("About to add " & FORM.firstname & " " & FORM.lastname);
</cfscript>
```
XmlChildPos

Description

Gets the position of a child element within an XML document object.

Returns

The position, in an XmlChildren array, of the Nth child that has the specified name.

Category

XML functions

Function syntax

XmlChildPos(elem, childName, N)

See also

IsXmlElem, XmlElemNew, XmlSearch, XmlTransform; Chapter 35, “Using XML and WDDX” in ColdFusion MX Developer's Guide

History

ColdFusion MX: Added this function.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>elem</td>
<td>XML element within which to search.</td>
</tr>
<tr>
<td>childName</td>
<td>XML child element for which to search. Must be an immediate child of the elem parameter.</td>
</tr>
<tr>
<td>N</td>
<td>Index of XML child element for which to search.</td>
</tr>
</tbody>
</table>

Usage

You can use the returned index in the ArrayInsertAt and ArrayDeleteAt functions to change XML document objects. If the specified child is not found, the function returns -1.

Example

The following example searches XML document element xmlobject.employee.name[1], for its second Status element child and uses the position in an ArrayDeleteAt function to remove the element:

```xml
<!--- Create an XML document object --->
<cfxml variable="xmlobject">
<employee>
  <!-- A list of employees -->
  <name EmpType="Regular">
    <first>Almanzo</first>
    <last>Wilder</last>
    <Status>Medical Absence</Status>
    <Status>Extended Leave</Status>
  </name>
</employee>
```

908 Chapter 3: ColdFusion Functions
<first>Laura</first>
<last>Ingalls</last>
</name>
</employee>
</cfxml>

<!--- Find the second Status child of the first employee.name element --->
<cfscript>
   elempos=XMLChildPos(xmlobject.employee.name[1]."Status", 2);
   ArrayDeleteAt(xmlobject.employee.name[1].XmlChildren, elempos);
</cfscript>

<!--- Dump the resulting document object to confirm the deletion --->
<cfdump var="#xmlobject#"
XmlElemNew

Description
Creates an XML document object element.

Returns
An XML document object element.

Category
XML functions

Function syntax
XmlElemNew(xmlObj[, namespace], childName)

See also
cfxml, IsXmlElem, XmlChildPos, XmlFormat, XmlNew, XmlParse; Chapter 35, “Using XML and WDDX” in ColdFusion MX Developer’s Guide

History
ColdFusion MX 7: Added the namespace parameter.
ColdFusion MX: Added this function.

Parameters
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>xmlObj</td>
<td>Name of the XML document object in which you are creating the element</td>
</tr>
<tr>
<td>namespace</td>
<td>(Optional) URI of the namespace to which this element belongs</td>
</tr>
<tr>
<td>childName</td>
<td>Name of the element to create</td>
</tr>
</tbody>
</table>

Usage
The function’s return variable specifies the location of the new element in the document object. It must specify a valid location in the document object identified by the xmlObj parameter. The following statements show this use:

```cfc
ArrayAppend(MyDoc.MyRoot.XmlChildren, XmlElemNew(MyDoc,"childNode"));
```

If you do not specify a namespace URI and use a namespace prefix in the childName parameter, ColdFusion checks to see if a namespace URI has already been specified for the prefix, and if so, uses that namespace.

Example
The following example creates and displays a ColdFusion document object:

```cfc
<cfscript>
MyDoc = XmlNew();
MyDoc.xmlRoot = XmlElemNew(MyDoc,"MyRoot");
if (testVar IS TRUE)
```
else
    MyDoc.MyRoot.XmlText = "The value of testVar is False.";
for (i = 1; i LTE 4; i = i + 1)
  MyDoc.MyRoot.XmlChildren[i].XmlText = "This is Child node " & i & ":
</cfscript>
<cfdump var=#MyDoc#>
XmlFormat

Description
Espaces special XML characters in a string so that the string can be used as text in XML.

Returns
A copy of the string parameter that is safe to use as text in XML.

Category
String functions, XML functions

Function syntax
XmlFormat(string)

See also
cfxml, XmlNew, XmlParse, XmlValidate; Chapter 35, “Using XML and WDDX” in ColdFusion MX Developer’s Guide

History
ColdFusion MX: Added this function.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>string</td>
<td>A string or a variable that contains one</td>
</tr>
</tbody>
</table>

Usage
This function escapes characters as follows:

<table>
<thead>
<tr>
<th>Text character</th>
<th>Escaped representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Greater than symbol (&gt;)</td>
<td>&gt;</td>
</tr>
<tr>
<td>Less than symbol (&lt;)</td>
<td>&lt;</td>
</tr>
<tr>
<td>Single-quotation mark (’)</td>
<td>'</td>
</tr>
<tr>
<td>Double-quotation mark (“)</td>
<td>&quot;</td>
</tr>
<tr>
<td>Ampersand symbol (&amp;)</td>
<td>&amp;</td>
</tr>
<tr>
<td>Carriage return (but not line feed)</td>
<td>Removed from the text.</td>
</tr>
</tbody>
</table>
| High ASCII characters in the range 128-255. | Replaced by unicode escape sequence; for example, É (capital E with an Acute symbol) is replaced by &xc9;.

Example
The following example shows how XmlFormat escapes special XML characters. Use the View Source command in the browser to see the results. ColdFusion interprets the “” in the second text string as representing a single-quotation mark in text before it applies the XmlFormat function.

```xml
<?xml version = "1.0"?>
<cfoutput>
<tr>
```

912  Chapter 3: ColdFusion Functions
<someXML>
    <someElement someAttribute="#XmlFormat("'a quoted value'")#">
        #XmlFormat("Body of element with <, >, '', " and & goes here.")#
    </someElement>
</someXML>
XmlGetNodeType

**Description**
Determines the type of an XML document object node.

**Returns**
A string identifying the XML node type. The following values are valid:

<table>
<thead>
<tr>
<th>Node Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATTRIBUTE_NODE</td>
<td>CDATA_SECTION_NODE</td>
</tr>
<tr>
<td>COMMENT_NODE</td>
<td>DOCUMENT_FRAGMENT_NODE</td>
</tr>
<tr>
<td>DOCUMENT_NODE</td>
<td>DOCUMENT_TYPE_NODE</td>
</tr>
<tr>
<td>ELEMENT_NODE</td>
<td>ENTITY_NODE</td>
</tr>
<tr>
<td>ENTITY_REFERENCE_NODE</td>
<td>NOTATION_NODE</td>
</tr>
<tr>
<td>PROCESSING_INSTRUCTION_NODE</td>
<td>TEXT_NODE</td>
</tr>
</tbody>
</table>

If the argument is not a document object node, the function generates an error.

**Category**
XML functions

**Function syntax**
`XmlGetNodeType(xmlNode)`

**See also**
`IsXmlAttribute, IsXmlDoc, IsXmlElem, IsXmlNode, IsXmlRoot, XmlChildPos, XmlValidate`;
Chapter 35, "Using XML and WDDX" in ColdFusion MX Developer's Guide

**History**
ColdFusion MX 7: Added this function.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>xmlNode</td>
<td>An XML DOM object node</td>
</tr>
</tbody>
</table>

**Usage**
The `XmlGetNodeType` function can determine the types of the nodes returned by the `XmlSearch` function, or the types of the entries in an element's XmlNodes array.

**Example**
The following example checks the node types of various parts of an XML document object:

```xml
<!---- Create an XML document object --->
cfxml variable="xmlobject">
<?xml version="1.0" encoding="UTF-8"?>
<order id="4323251">
  <customer firstname="Philip" lastname="Cramer" accountNum="21"/>
</order>
```
<items>
  <item id="43">
    <!-- This item is coded to show several node types -->
    <![CDATA["Our Best" hammer & chisel set!!!]]> Imported from France
    <quantity>1</quantity>
    <unitprice>15.95</unitprice>
  </item>
</items>
</order>
</cfxml>

<!--- Display the node types --->
<cfoutput>
  <h3>Node Types</h3>
  xmlobject: #XMLGetNodeType(xmlobject)#<br>
  xmlobject.order: #XMLGetNodeType(xmlobject.order)#
  <br>
  Now check the types of all the nodes in the xmlobject.order.items.item
  element's XmNodes array.<br>
  Note the many apparently empty Text nodes generated by whitespace characters
  in the XML text source.<br>
  <cfset descnodes=xmlobject.order.items.item.XrNodes>
  <cfloop from="1" to="#ArrayLen(descnodes)#" index="i">
    #i# Node type is: #XMLGetNodeType(descnodes[i])#<br>
    #i# Node name is: #descnodes[i].XmlName#<br>
    <cfif (descnodes[i].XmlValue NEQ ")"><br>
    #i# Node value is: #descnodes[i].XmlValue#<br>
  </cfif>
  </cfloop>
</cfoutput>
**XmlNew**

**Description**

Creates an XML document object.

**Returns**

An empty XML document object.

**Category**

XML functions

**Function syntax**

```xml
XmlNew([caseSensitive])
```

**See also**

cfxml, IsXmlDoc, ToString, XmlFormat, XmlParse, XmlValidate; Chapter 35, “Using XML and WDDX” in *ColdFusion MX Developer’s Guide*

**History**

ColdFusion MX: Added this function.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>caseSensitive</td>
<td>Determines how ColdFusion processes the case of XML document object component identifiers:</td>
</tr>
<tr>
<td></td>
<td>• True: maintains case</td>
</tr>
<tr>
<td></td>
<td>• False: ColdFusion ignores case. Default.</td>
</tr>
</tbody>
</table>

**Usage**

An XML document object is represented in ColdFusion as a structure.

The `caseSensitive` parameter value determines whether identifiers whose characters are of varying case, but are otherwise the same, refer to different components; for example:

- If True, the element or attribute names "name" and "NAME" refer to different elements or attributes.
- If False, these names refer to the same elements or attributes.

If your XML object is case sensitive, you cannot use dot notation to reference an element or attribute name. Use the name in associative array (bracket) notation, or a reference that does not use the case-sensitive name (such as xmlChildren[1]) instead. In the following code, the first line will work with a case-sensitive XML object. The second and third lines cause errors:

```plaintext
MyDoc.xmlRoot.XmlAttributes["Version"] = "12b";
MyDoc.xmlRoot.XmlAttributes.Version = "12b";
MyDoc.MyRoot.XmlAttributes["Version"] = "12b";
```

To convert an XML document object into a string, use the `ToString` function.
Example

The following example creates and displays a ColdFusion document object:

```coldfusion
<cfset testVar = True>
<cfscript>
    MyDoc = XmlNew();
    MyDoc.xmlRoot = XmlElemNew(MyDoc,"MyRoot");
    if (testVar IS TRUE)
        MyDoc.MyRoot.XmlText = "The value of testVar is True.";
    else
        MyDoc.MyRoot.XmlText = "The value of testVar is False.";
    for (i = 1; i LTE 4; i = i + 1)
        MyDoc.MyRoot.XmlChildren[i].XmlText = "This is Child node " & i & ".";
    </cfscript>
<cfdump var=#MyDoc#>
```
XmlParse

Description
Converts XML text into an XML document object.

Returns
An XML document object.

Category
Conversion functions, XML functions

Function syntax
XmlParse(xmlText [[, caseSensitive [, validator]]])

See also
cfxml, IsXML, ToString, XmlFormat, XmlNew, XmlSearch, XmlTransform, XmlValidate;
Chapter 35, “Using XML and WDDX” in ColdFusion MX Developer’s Guide

History
ColdFusion MX 7:
• Added the validator parameter.
• Added support for filenames and URLs in the xmlText parameter.
• Added support for relative URLs and path names.
ColdFusion MX: Added this function.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>xmlText</td>
<td>Any of the following:</td>
</tr>
<tr>
<td></td>
<td>• A string containing XML text.</td>
</tr>
<tr>
<td></td>
<td>• The name of an XML file.</td>
</tr>
<tr>
<td></td>
<td>• The URL of an XML file; valid protocol identifiers include http, https, ftp, and file.</td>
</tr>
<tr>
<td>caseSensitive</td>
<td>• Yes: maintains the case of document elements and attributes.</td>
</tr>
<tr>
<td></td>
<td>• No: Default</td>
</tr>
<tr>
<td>validator</td>
<td>Any of the following:</td>
</tr>
<tr>
<td></td>
<td>• The name of a Document Type Definition (DTD) or XML Schema file.</td>
</tr>
<tr>
<td></td>
<td>• The URL of a DTD or Schema file; valid protocol identifiers include http, https, ftp, and file.</td>
</tr>
<tr>
<td></td>
<td>• A string representation of a DTD or Schema.</td>
</tr>
<tr>
<td></td>
<td>• An empty string; in this case, the XML file must contain an embedded DTD or Schema identifier, which is used to validate the document.</td>
</tr>
</tbody>
</table>

Usage
If you specify a relative URL or path name in a parameter, ColdFusion uses the directory (or, for URLs, the logical directory) that contains the current ColdFusion page as the path root.
The `caseSensitive` parameter value determines whether identifiers whose characters are of varying case, but are otherwise the same, refer to different components; for example:

- If true, the element or attribute names “name” and “NAME” refer to different elements or attributes.
- If false, these names refer to the same elements or attributes.

If your XML object is case sensitive, you cannot use dot notation to reference an element or attribute name. Use the name in associative array (bracket) notation, or a reference that does not use the case-sensitive name (such as `xmlChildren[1]`) instead. In the following code, the first line will work with a case-sensitive XML object. The second and third lines cause errors:

```java
MyDoc.xmlRoot.XmlAttributes["Version"] = "12b";
MyDoc.xmlRoot.XmlAttributes.Version = "12b";
MyDoc.MyRoot.XmlAttributes["Version"] = "12b";
```

The optional `validator` parameter specifies a DTD or Schema to use to validate the document. If the parser encounters a validation error, ColdFusion generates an error and stops parsing the document. You must specify a `validator` parameter to make the `XmlParse` function validate your document. If you do not specify a `validator` parameter, and the XML file specifies a DTD or Schema, ColdFusion ignores the DTD or Schema. If you specify a `validator` parameter, you must also specify a `caseSensitive` parameter.

If you do not specify a `validator` parameter, the `xmlText` parameter can specify a well-formed XML fragment, and does not have to specify a complete document.

**Note:** To convert an XML document object back into a string, use the `ToString` function.

**Example**

The following example has three parts: an XML file, a DTD file, and a CFML page that parses the XML file and uses the DTD for validation. The CFML file displays the returned XML document object. To show the results of invalid XML, modify the `bmenuD.xml`.

**Note:** The DTD used in the following example represents the same XML structure as the Schema used in the `XmlValidate` example.

The `custorder.xml` file is as follows:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE order SYSTEM "C:\CFusionMX7\wwwroot\examples\custorder.dtd">
<order id="4323251">
  <customer firstname="Philip" lastname="Cramer" accountNum="21"/>
  <items>
    <item id="43">
      <name>Deluxe Carpenter's Hammer</name>
      <quantity>1</quantity>
      <unitprice>15.95</unitprice>
    </item>
    <item id="54">
      <name>36" Plastic Rake</name>
      <quantity>2</quantity>
      <unitprice>6.95</unitprice>
    </item>
    <item id="68">
```
The custorder.dtd file is as follows:

```xml
<!ELEMENT order (customer, items)>
<!ATTLIST order
  id CDATA #REQUIRED>
<!ELEMENT customer EMPTY>
<!ATTLIST customer
  firstname CDATA #REQUIRED
  lastname CDATA #REQUIRED
  accountNum CDATA #REQUIRED>
<!ELEMENT items (item+)>   
<!ELEMENT item (name, quantity, unitprice)>
<!ATTLIST item
  id CDATA #REQUIRED>
<!ELEMENT name (#PCDATA)>
<!ELEMENT quantity (#PCDATA)>
<!ELEMENT unitprice (#PCDATA)>
```

The CFML file is as follows. It uses a filename for the XML file and a URL for the DTD. Note that the XML and URL paths must be absolute.

```cfml
<cfset
  myDoc=XMLParse("C:\CFusionMX7\wwwroot\examples\custorder.xml",
                  false, "http://localhost:8500/examples/custorder.dtd")>
Dump of myDoc XML document object<br>
<cfdump var="#myDoc#">
```
XmlSearch

Description
Uses an XPath language expression to search an XML document object.

Returns
An array of XML object nodes that match the search criteria.

Category
XML functions

Function syntax
XmlSearch(xmlDoc, xPathString)

See also
cfxml, IsXML, XmlChildPos, XmlParse, XmlTransform; Chapter 35, “Using XML and WDDX” in ColdFusion MX Developer’s Guide

History
ColdFusion MX 7: Added support for attribute searches.
ColdFusion MX: Added this function.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>xmlDoc</td>
<td>XML document object</td>
</tr>
<tr>
<td>xPathString</td>
<td>XPath expression</td>
</tr>
</tbody>
</table>

Usage
The XmlSearch function does not support complete XPath syntax. It only supports XPath expressions that return one or more XML nodes and attribute searches. XPath expressions that return any other type of value, such as a string, number, or Boolean, generate errors.

XPath is specified by the World Wide Web Consortium (W3C). For detailed information on XPath, including XPath expression syntax, see the W3C website at www.w3.org/TR/xpath.

Example
The following example extracts the elements named last, which contain employee last names, from an XML file, and displays the names.

```
<?xml version="1.0" encoding="UTF-8"?>
<employee>
<!-- A list of employees -->
<name EmpType="Regular">
  <first>Almanzo</first>
  <last>Wilder</last>
</name>
```

Example: employeesimple.xml file contains the following XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<employee>
  <!-- A list of employees -->
  <name EmpType="Regular">
    <first>Almanzo</first>
    <last>Wilder</last>
  </name>
```
The CFML file contains the following lines:

```cfscript
myxmlDoc = XmlParse("C:\CFusionMX7\wwwroot\examples\employeesimple.xml");
selectedElements = XmlSearch(myxmlDoc, "/employee/name/last");
for (i = 1; i LTE ArrayLen(selectedElements); i = i + 1)
    writeoutput(selectedElements[i].XmlText & "<br>");
</cfscript>
```
**XmlTransform**

**Description**
Applies an Extensible Stylesheet Language Transformation (XSLT) to XML. The XML can be in string format or an XML document object.

**Returns**
A string containing the results of applying the XSLT to the XML.

**Category**
Conversion functions, XML functions

**Function syntax**
```
XmlTransform(xml, xsl[, parameters])
```

**See also**
cfxml, XmlFormat, XmlNew, XmIParse, XmlSearch, XmlValidate; Chapter 35, “Using XML and WDDX” in *ColdFusion MX Developer’s Guide*

**History**
ColdFusion MX 7: Added the parameters parameter and the ability to use a file for the XSL.
ColdFusion MX: Added this function.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>xml</td>
<td>An XML document in string format, or an XML document object</td>
</tr>
</tbody>
</table>
| xsl       | XSLT transformation to apply; can be any of the following: Any of the following:  
  • A string containing XSL text.  
  • The name of an XSLT file. Relative paths start at the directory containing the current CFML page.  
  • The URL of an XSLT file; valid protocol identifiers include http, https, ftp, and file. Relative paths start at the directory containing the current CFML page. |
| parameters | A structure containing XSL template parameter name-value pairs to use in transforming the document. The XSL transform defined in the xslString parameter uses these parameter values in processing the XML. |

**Usage**
An XSLT converts an XML document to another format or representation by applying an Extensible Stylesheet Language (XSL) stylesheet to it. XSL, including XSLT syntax is specified by the World Wide Web Consortium (W3C). For detailed information on XSL and XSLT, see the W3C website at [www.w3.org/Style/XSL/](http://www.w3.org/Style/XSL/).
If the XSLT code contains include statements with relative paths, ColdFusion resolves them relative to the location of the XSLT file, or for an XSL string, the location of the current ColdFusion page.

**Example**

The following example converts an XML document that represents a customer order into an HTML document with the customer name and a table with the order items and quantities:

The custorder.xml file that represents a customer order has the following lines:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<order id="4323251">
  <customer firstname="Philip" lastname="Cramer" accountNum="21"/>
  <items>
    <item id="43">
      <name>Deluxe Carpenter's Hammer</name>
      <quantity>1</quantity>
      <unitprice>15.95</unitprice>
    </item>
    <item id="54">
      <name>36" Plastic Rake</name>
      <quantity>2</quantity>
      <unitprice>6.95</unitprice>
    </item>
    <item id="68">
      <name>Standard paint thinner</name>
      <quantity>3</quantity>
      <unitprice>8.95</unitprice>
    </item>
  </items>
</order>
```

The custorder.xsd XSLT file that transforms the XML to HTML that displays the customer's name, and the items and quantities ordered has the following lines:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="html" doctype-public="-//W3C//DTD HTML 4.0 Transitional//EN"/>
  <xsl:template match="/">
    <html>
      <body>
        <table border="2" bgcolor="yellow">
          <tr>
            <th>Name</th>
            <th>Price</th>
          </tr>
          <xsl:for-each select="breakfast_menu/food">
            <tr>
              <td>
                <xsl:value-of select="name"/>
              </td>
              <td>
                <xsl:value-of select="price"/>
              </td>
            </tr>
          </xsl:for-each>
        </table>
      </body>
    </html>
  </xsl:template>
</xsl:stylesheet>
```
The CFML file has the following lines:
<cffile action="read" file="C:\CFusionMX7\wwwroot\examples\custorder.xsl" variable="xmltrans">
<cfset xmldoc = XmlParse("C:\CFusionMX7\wwwroot\examples\custorder.xml")>
<cfoutput>$XmlTransform(xmldoc, xmltrans)<!-- $output --></cfoutput>
XmlValidate

Description

Uses a Document Type Definition (DTD) or XML Schema to validate an XML text document or an XML document object.

Returns

The following validation structure:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Errors</td>
<td>An array containing any validator error messages. These messages indicate that the document does not conform to the DTD or Schema (is not valid).</td>
</tr>
<tr>
<td>FatalErrors</td>
<td>An array containing any validator fatal error messages. Fatal errors indicate that the document contains XML formatting errors (is not well-formed XML).</td>
</tr>
</tbody>
</table>
| Status  | A Boolean value:  
  • True if the document is valid.  
  • False if the validation check failed. |
| Warning | An array containing any validator warnings. A well-formed and valid document can produce warning messages. |

Category

XML functions

Function syntax

XmlValidate(xmlDoc[, validator])

See also

cfxml, IsXmlDoc, IsXML, XmlFormat, XmlNew, XmlParse, XmlSearch, XmlTransform;  
Chapter 35, “Using XML and WDDX” in ColdFusion MX Developer’s Guide

History

ColdFusion MX 7: Added this function.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| xmlDoc    | Any of the following:  
  • A string containing an XML document.  
  • The name of an XML file.  
  • The URL of an XML file; valid protocol identifiers include http, https, ftp, and file.  
  • An XML document object, such as one generated by the XmlParse function. |
| validator | Any of the following:  
  • A string containing a DTD or Schema.  
  • The name of a DTD or Schema file.  
  • The URL of a DTD or Schema file; valid protocol identifiers include http, https, ftp, and file. |
Usage

If you specify a relative URL or filename in a parameter, ColdFusion uses the directory (or, for URLs, the virtual directory) that contains the current ColdFusion page as the path root.

The validator parameter specifies a DTD or Schema to use to validate the document. If you omit the parameter, the XML document must contain one of the following:

- A !DOCTYPE tag to specify the DTD or its location
- An xsi:schemaLocation or xsi:noNamespaceSchemaLocation tag to specify the Schema location

If you use a validator parameter and the XML document specifies a DTD or Schema, the XmlValidate function uses the validator parameter, and ignores the specification in the XML document.

If you do not use a validator parameter, and the XML document does not specify a DTD or Schema, the function returns a structure with an error message in the Errors field.

This function attempts to process the complete XML document, and reports all errors found during the processing. As a result, the returned structure can have a combination of Warning, Error, and FatalError fields, and each field can contain multiple error messages.

Example

The following example has three parts: an XML file, an XSD Schema file, and a CFML page that parses the XML file and uses the Schema for validation. The CFML file displays the value of the returned structure's Status field and displays the returned structure. To show the results of invalid XML, modify the custorder.xml file.

Note: The Schema used in the following example represents the same XML structure as the DTD used in the XmlParse example.

The custorder.xml file is as follows:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<order xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="http://localhost:8500/something.xsd"
    id="4323251">
  <customer firstname="Philip" lastname="Cramer" accountNum="21"/>
  <items>
    <item id="43">
      <name>Deluxe Carpenter's Hammer</name>
      <quantity>1</quantity>
      <unitprice>15.95</unitprice>
    </item>
    <item id="54">
      <name>36" Plastic Rake</name>
      <quantity>2</quantity>
      <unitprice>6.95</unitprice>
    </item>
    <item id="68">
      <name>Standard paint thinner</name>
      <quantity>3</quantity>
      <unitprice>8.95</unitprice>
  </items>
</order>
```
The custorder.xsd file is as follows:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
   elementFormDefault="qualified">
  <xs:element name="customer">
    <xs:complexType>
      <xs:attribute name="firstname" type="xs:string" use="required"/>
      <xs:attribute name="lastname" type="xs:string" use="required"/>
      <xs:attribute name="accountNum" type="xs:string" use="required"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="name" type="xs:string"/>
  <xs:element name="quantity" type="xs:string"/>
  <xs:element name="unitprice" type="xs:string"/>
  <xs:element name="item">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="name"/>
        <xs:element ref="quantity"/>
        <xs:element ref="unitprice"/>
      </xs:sequence>
      <xs:attribute name="id" type="xs:integer" use="required"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="items">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="item" maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="order">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="customer"/>
        <xs:element ref="items"/>
      </xs:sequence>
      <xs:attribute name="id" type="xs:string" use="required"/>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

The CFML file is as follows. It uses a filename for the XML file and a URL for the Schema. The XML and URL paths must be absolute.

```cfml
<cfset myResults=XMLValidate("C:\CFusionMX7\wwwroot\examples\custorder.xml", "http://localhost:8500/examples/custorder.xsd")>
<cfoutput>
```

928 Chapter 3: ColdFusion Functions
Did custorder.xml validate against custorder.xsd?

Dump of myResults structure returned by XMLValidate

```cfdump var="#myResults#"```
Year

Description
From a date/time object, gets the year value.

Returns
The year value of date.

Category
Date and time functions

Function syntax
Year(date)

See also
DatePart, IsLeapYear

Parameters
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>date</td>
<td>A date/time object in the range 100 AD–9999 AD.</td>
</tr>
</tbody>
</table>

Usage
When passing a date as a string, enclose it in quotation marks. Otherwise, it is interpreted as a number representation of a date.

Example

```cfml
<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>
```

---

930 Chapter 3: ColdFusion Functions
**YesNoFormat**

**Description**

Evaluates a number or Boolean value.

**Returns**

Yes, for a nonzero value; No for zero, `false`, and `no` Boolean values, and an empty string (`""`).

**Category**

Decision functions, Display and formatting functions

**Function syntax**

```
YesNoFormat(value)
```

**See also**

`IsBinary`, `IsNumeric`

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>A number or Boolean value</td>
</tr>
</tbody>
</table>

**Example**

```
<h3>YesNoFormat Example</h3>
<p>The YesNoFormat function returns non-zero values as "Yes"; zero, false and no Boolean values, and empty strings (""") as "No".
</p>

<cfoutput>
<ul>
  <li>YesNoFormat(1):#YesNoFormat(1)#</li>
  <li>YesNoFormat(0):#YesNoFormat(0)#</li>
  <li>YesNoFormat("123"):#YesNoFormat("123")#</li>
  <li>YesNoFormat("No"):#YesNoFormat("No")#</li>
  <li>YesNoFormat(True):#YesNoFormat(True)#
  </li>
</ul>
</cfoutput>
```
CHAPTER 4
ColdFusion MX Flash Form Style Reference

This chapter describes the styles that you can specify in Macromedia ColdFusion MX 7 forms tags when you display the form or form element in Flash format.

Contents

- Styles valid for all controls ......................................................... 934
- Styles for cfform ........................................................................ 936
- Styles for cfformgroup with horizontal or vertical type attributes. ......................................................... 936
- Styles for box-style cfformgroup elements ................................. 937
- Styles for cfformgroup with accordion type attribute ............... 938
- Styles for cfformgroup with tabnavigator type attribute .......... 939
- Styles for cfformitem with hrule or vrule type attributes. .......... 939
- Styles for cfinput with radioButton, checkbox, button, image, or submit type attributes .... 940
- Styles for cftextarea tag and cfinput with text, password, or hidden type attributes. .......... 941
- Styles for cfselect with size attribute value of 1 ................................. 941
- Styles for cfselect with size attribute value greater than 1 .......... 942
- Styles for cfcalendar tag and cfinput with dateField type attribute ............ 942
- Styles for the cfgrid tag ............................................................. 943
- Styles for the cftree tag ............................................................. 943

Note: The column labeled Inh indicates whether a style is inherited by child controls, such as the form controls in a vbox.
Styles valid for all controls

The following styles are valid for all ColdFusion Flash format form tags except for `cformitem` tags with the following `type` attributes, which do not take `style` attributes:

- html
- space
- text

These styles do not cause errors when used in all other tags. However, many styles do not have any effect when used in some tags.

<table>
<thead>
<tr>
<th>Style</th>
<th>Inh</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>backgroundColor</td>
<td>Y</td>
<td>Format: color; background color of the control. Has no effect if specified in a <code>cform</code> control tag, which uses the <code>background-color</code> style to control the color. Also ignored by <code>cfinput</code> tags of <code>type</code> button, img, submit, radiobutton, and checkbox, because they are completely filled with the button face or other graphics.</td>
</tr>
<tr>
<td>backgroundDisabledColor</td>
<td>Y</td>
<td>Format: color; background color of components when disabled. The default value is ##EFEEEF (light gray).</td>
</tr>
<tr>
<td>backgroundColorSize</td>
<td>N</td>
<td>Scales the image specified by <code>backgroundImage</code> to different percentage sizes. By default, the value is auto, which maintains the original size of the image. A value of 100% stretches the image to fit the entire screen. You must include the percent sign with the value.</td>
</tr>
<tr>
<td>barColor</td>
<td>Y</td>
<td>Format: color; color of the outer bar.</td>
</tr>
<tr>
<td>borderCapColor</td>
<td>Y</td>
<td>Format: color; outside left and outside right color for skins.</td>
</tr>
<tr>
<td>borderColor</td>
<td>Y</td>
<td>Format: color; black section of a three-dimensional border or the color section of a two-dimensional border.</td>
</tr>
<tr>
<td>borderSides</td>
<td>N</td>
<td>Bounding box sides. Only used when <code>borderStyle=&quot;solid&quot;</code>. Space-delimited string containing the sides of the border to show. Order is not important. The default value is &quot;left top right bottom&quot;.</td>
</tr>
<tr>
<td>borderRadius</td>
<td>N</td>
<td>Bounding box thickness. Only used when <code>borderStyle=&quot;solid&quot;</code>. The default value is 1.</td>
</tr>
<tr>
<td>color</td>
<td>Y</td>
<td>Format: color; text color of a component's label.</td>
</tr>
<tr>
<td>cornerRadius</td>
<td>N</td>
<td>Radius of component corners. The default value is 0.</td>
</tr>
<tr>
<td>disabledColor</td>
<td>Y</td>
<td>Format: color; color of the component if it is disabled.</td>
</tr>
<tr>
<td>Style</td>
<td>Inh</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------</td>
<td>-----</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>dropShadow</td>
<td>N</td>
<td>Format: Boolean; controls the visibility of the component’s drop shadow. The default value is false. This style must be used with borderStyle=&quot;solid&quot;. For drop shadows to appear on containers, set backgroundColor or backgroundImage. Otherwise, since the default background of a container is transparent, the shadow appears behind the container.</td>
</tr>
<tr>
<td>errorColor</td>
<td>Y</td>
<td>Format: color; color of the error text.</td>
</tr>
<tr>
<td>fillColors</td>
<td>N</td>
<td>Format: color; colors used to tint the background of the control. Pass the same color for both values for &quot;flat&quot; looking control. The default value is ##E6EEEE,##FFFFFF.</td>
</tr>
<tr>
<td>fontFamily</td>
<td>Y</td>
<td>Comma-separated list of fonts to use, in descending order of desirability. You can use any font family name. If you specify a generic font name, it is converted to an appropriate device font. Flash can only use fonts that are installed on the client system.</td>
</tr>
<tr>
<td>fontSize</td>
<td>Y</td>
<td>Format: length; size of the text.</td>
</tr>
<tr>
<td>fontStyle</td>
<td>Y</td>
<td>Determines whether the text is italic. Recognized values are normal and italic. The default value is normal.</td>
</tr>
<tr>
<td>fontWeight</td>
<td>Y</td>
<td>Determines whether the text is bold. Recognized values are normal and bold. The default value is normal.</td>
</tr>
<tr>
<td>.highlightColor</td>
<td>Y</td>
<td>Format: color; color of the control when it is in focus.</td>
</tr>
<tr>
<td>horizontalGap</td>
<td>N</td>
<td>Format: length; number of pixels between children in the horizontal direction.</td>
</tr>
<tr>
<td>leading</td>
<td>N</td>
<td>Additional vertical space between lines of text. The default value is no leading.</td>
</tr>
<tr>
<td>marginLeft</td>
<td>N</td>
<td>Format: length; number of pixels between the container’s left border and its content area.</td>
</tr>
<tr>
<td>marginRight</td>
<td>N</td>
<td>Format: length; number of pixels between the container’s right border and its content area.</td>
</tr>
<tr>
<td>scrollTrackColor</td>
<td>Y</td>
<td>Format: color; scroll track for a scroll bar. The default value is ##EFEEEF (light gray).</td>
</tr>
<tr>
<td>selectedFillColors</td>
<td>N</td>
<td>Format: colors; two colors used to tint the background of the control when in its selected state. Pass the same color for both values for &quot;flat&quot; looking control. The default value is undefined, which means the colors will be derived from themeColor.</td>
</tr>
<tr>
<td>textAlign</td>
<td>Y</td>
<td>Aligns text in a container. Recognized values are left, right, and center. The default value is right.</td>
</tr>
<tr>
<td>textDecoration</td>
<td>N</td>
<td>Determines whether the text is underlined or not. Recognized values are none and underline. The default value is none.</td>
</tr>
<tr>
<td>textIndent</td>
<td>Y</td>
<td>Format: length; offset of first line of text from the left side of the container. The default value is 0.</td>
</tr>
</tbody>
</table>

Styles valid for all controls 935
### Styles for cfform

The following styles apply to the `cfform` tag:

<table>
<thead>
<tr>
<th>Style</th>
<th>Inh</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>themeColor</code></td>
<td>Y</td>
<td>Format: color; background color of a component. The possible values are:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• haloGreen</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• haloBlue</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• haloOrange</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• haloSilver</td>
</tr>
<tr>
<td><code>verticalGap</code></td>
<td>N</td>
<td>Format: length; number of pixels between children in the vertical direction.</td>
</tr>
</tbody>
</table>

#### Styles for cfformgroup with horizontal or vertical type attributes

The following styles apply to the `cfformgroup` tag with `type` attributes `horizontal` or `vertical`:

<table>
<thead>
<tr>
<th>Style</th>
<th>Inh</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>horizontalAlign</code></td>
<td>N</td>
<td>Horizontal alignment of children. Possible values are left, center, and right. The default value is left.</td>
</tr>
<tr>
<td><code>horizontalGap</code></td>
<td>N</td>
<td>Format: length; number of pixels between children in the horizontal direction. The default value is 6.</td>
</tr>
<tr>
<td><code>indicatorGap</code></td>
<td>Y</td>
<td>Format: length; number of pixels between the label and child components. The default value is 14.</td>
</tr>
<tr>
<td><code>labelWidth</code></td>
<td>Y</td>
<td>Format: length; width of the form labels. The default value is the length of the longest label in the form.</td>
</tr>
<tr>
<td><code>marginBottom</code></td>
<td>N</td>
<td>Format: length; number of pixels between the container’s bottom border and its content area. The default value is 16.</td>
</tr>
<tr>
<td><code>marginTop</code></td>
<td>N</td>
<td>Format: length; number of pixels between the container’s top border and its content area. The default value is 16.</td>
</tr>
<tr>
<td><code>verticalGap</code></td>
<td>N</td>
<td>Format: length; number of pixels between children in the vertical direction. The default value is 8.</td>
</tr>
</tbody>
</table>
The following styles apply to the `cfformgroup` tag with the following `type` attributes. Some types have additional attributes, which are listed in the following sections.

- hbox
- vbox
- hdividedbox
- vdividedbox
- panel
- tile
- page

### Styles specific to `cfformgroup` with hdividedbox or vdividedbox type attributes

The following additional styles apply to the `cfformgroup` tag with `type="hdividedbox"`, or `type="vdividedbox"`:

<table>
<thead>
<tr>
<th>Style</th>
<th>Inh</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dividerAffordance</td>
<td>N</td>
<td>Format: length; width (hdividedbox) or height (vdividedbox) in pixels of the area of the divider that the user can select with the mouse pointer. The default value is 6.</td>
</tr>
</tbody>
</table>
Styles specific to `cfformgroup` with panel type attribute

The following additional styles apply to the `cfformgroup` tag with `type="panel"`:

<table>
<thead>
<tr>
<th>Style</th>
<th>Inh</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dividerColor</td>
<td>Y</td>
<td>Format: color; color of the dividers in their up state. The default value is ##AAAAAA.</td>
</tr>
<tr>
<td>dividerThickness</td>
<td>N</td>
<td>Format: length; thickness in pixels of the dividers. The default value is 4.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Style</th>
<th>Inh</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cornerRadius</td>
<td>N</td>
<td>Format: length; radius of corners of the window frame. The default value is 8.</td>
</tr>
<tr>
<td>dropShadow</td>
<td>N</td>
<td>Boolean value specifying whether the panel has a drop shadow. The default value is true.</td>
</tr>
<tr>
<td>footerColors</td>
<td>Y</td>
<td>Format: color; comma-delimited list of two colors used to draw the footer (ControlBar) background. The first color is the top color. The second color is the bottom color. The default value is ##F4F5F7, ##E1E5EB.</td>
</tr>
<tr>
<td>headerColors</td>
<td>Y</td>
<td>Format: color; comma-delimited list of two colors used to draw the header. The first color is the top color. The second color is the bottom color. The default value is ##E1E5EB, ##F4F5F7.</td>
</tr>
<tr>
<td>headerHeight</td>
<td>N</td>
<td>Format: length; height of the header. The default value is 28.</td>
</tr>
<tr>
<td>panelBorderStyle</td>
<td>N</td>
<td>Border style for the bottom two corners of the container. The top two corners are always round. Possible values are default, which configures the container to have square corners, and roundCorners, which defines rounded corners. To configure the top corners to be square, set cornerRadius to 0. The default value is default.</td>
</tr>
<tr>
<td>shadowDirection</td>
<td>N</td>
<td>Direction of drop shadow. Possible values are &quot;left&quot;, &quot;center&quot;, and &quot;right&quot;. The default value is &quot;center&quot;.</td>
</tr>
<tr>
<td>shadowDistance</td>
<td>N</td>
<td>Distance of drop shadow. Negative values move shadow above the panel. The default value is 2.</td>
</tr>
</tbody>
</table>

Styles for `cfformgroup` with accordion type attribute

The following styles apply to the `cfformgroup` tag with `type="accordion"`:

<table>
<thead>
<tr>
<th>Style</th>
<th>Inh</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>headerHeight</td>
<td>N</td>
<td>Format: length; height of the accordion container buttons, in pixels. The default value is 22.</td>
</tr>
<tr>
<td>marginBottom</td>
<td>N</td>
<td>Format: length; number of pixels between the container’s bottom border and its content area. The default value is -1.</td>
</tr>
<tr>
<td>marginTop</td>
<td>N</td>
<td>Format: length; number of pixels between the container’s top border and its content area. The default value is -1.</td>
</tr>
</tbody>
</table>
Styles for cfformgroup with tabnavigator type attribute

The following styles apply to the cfformgroup tag with the type="tabnavigator":

<table>
<thead>
<tr>
<th>Style</th>
<th>Inh</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>horizontalAlign</td>
<td>N</td>
<td>Horizontal alignment of children. The default value is left. Possible values are left, center, and right. Because the preferred width of each tab in the tab navigator container is the size of the label text, you must use the tabWidth style to increase the width of the tab to a size larger than its preferred width to see different alignments.</td>
</tr>
<tr>
<td>horizontalGap</td>
<td>N</td>
<td>Format: length; number of pixels between children in the horizontal direction. The default value is 6.</td>
</tr>
<tr>
<td>tabHeight</td>
<td>N</td>
<td>Format: length; default tab height, in pixels. The default value is 22.</td>
</tr>
<tr>
<td>tabWidth</td>
<td>N</td>
<td>Format: length; width of the tabs, in pixels. If undefined, the default tab widths are automatically calculated from the label text. If the width of the container is smaller than the width of the label text, the labels are truncated. If a tab label is truncated, Flash displays a tooltip with the full label text when a user moves the mouse pointer over the tab. If you specify an explicit tab width, labels do not automatically shrink to fit if they do not fit in the available space.</td>
</tr>
</tbody>
</table>

Styles for cfformitem with hrule or vrule type attributes

The following styles apply to the formitem tag with type="hrule" or type="vrule":

<table>
<thead>
<tr>
<th>Style</th>
<th>Inh</th>
<th>Description</th>
</tr>
</thead>
</table>
| color | Y   | Format: color; color of the line, according to the following rules:  
  • If strokeWidth is 1, the color of the entire line.  
  • If strokeWidth is 2 (default), the color of the top line.  
  • If strokeWidth is greater than 2, the color of the top and left edges of the rectangle.  
  The default value is ##C4CCCC. |
Styles for `cfinput` with radioButton, checkbox, button, image, or submit type attributes

The following styles apply `cfinput` tags with the following `type` attribute values:

- button
- checkbox
- image
- radiobutton
- submit

In some cases, a style applies only to the subset of these input types, as specified in the description.

<table>
<thead>
<tr>
<th>Style</th>
<th>Inh</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>shadowColor</td>
<td>Y</td>
<td>Format: color; shadow color of the line, as follows:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• If <code>strokeWidth</code> is 1, does nothing.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• If <code>strokeWidth</code> is 2 (default), the color of the bottom line.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• If <code>strokeWidth</code> is greater than 2, the color of the bottom and right</td>
</tr>
<tr>
<td></td>
<td></td>
<td>edges of the rectangle.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The default value is #D4D0C8.</td>
</tr>
<tr>
<td>strokeWidth</td>
<td>Y</td>
<td>Thickness of the rule in pixels, as follows:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• If <code>strokeWidth</code> is 1, the rule is a 1-pixel-wide line.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• If <code>strokeWidth</code> is 2 (default), the rule is two adjacent 1-pixel-wide</td>
</tr>
<tr>
<td></td>
<td></td>
<td>horizontal lines.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• If <code>strokeWidth</code> is greater than 2, the rule is a hollow rectangle with</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1-pixel-wide edges.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The default value is 2.</td>
</tr>
</tbody>
</table>

#### Styles for `cfinput` with `radioButton`, `checkbox`, `button`, `image`, or `submit` type attributes

The following styles apply `cfinput` tags with the following `type` attribute values:

- button
- checkbox
- image
- radiobutton
- submit

In some cases, a style applies only to the subset of these input types, as specified in the description.

<table>
<thead>
<tr>
<th>Style</th>
<th>Inh</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>borderThickness</td>
<td>N</td>
<td>Thickness of border &quot;ring&quot;. A value of 0 means no border. Any value greater</td>
</tr>
<tr>
<td></td>
<td></td>
<td>than 2 creates a glowing &quot;ring&quot; around the button. The default value is 3.</td>
</tr>
<tr>
<td>cornerRadius</td>
<td>N</td>
<td>Radius of corners. The default value is 5.</td>
</tr>
<tr>
<td>horizontalGap</td>
<td>N</td>
<td>Gap between the label and the image in an <code>img</code> input when <code>labelPlacement</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td>= &quot;left&quot; or &quot;right&quot;. The default value is 2.</td>
</tr>
<tr>
<td>repeatDelay</td>
<td>N</td>
<td>Format: time; number of milliseconds to wait after the first <code>buttonDown</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td>event before repeating <code>buttonDown</code> events at the <code>repeatInterval</code>. The</td>
</tr>
<tr>
<td></td>
<td></td>
<td>default value is 500.</td>
</tr>
<tr>
<td>repeatInterval</td>
<td>N</td>
<td>Format: time; number of milliseconds between <code>buttonDown</code> events if you</td>
</tr>
<tr>
<td></td>
<td></td>
<td>press and hold a button. The default value is 35.</td>
</tr>
<tr>
<td>symbolBackgroundColor</td>
<td>Y</td>
<td>Format: color; background color of check boxes and radio buttons. The</td>
</tr>
<tr>
<td></td>
<td></td>
<td>default value is #FFFFFF (white).</td>
</tr>
<tr>
<td>symbolBackground</td>
<td>Y</td>
<td>Format: color; background color of check boxes and radio buttons when</td>
</tr>
<tr>
<td>DisabledColor</td>
<td></td>
<td>disabled. The default value is #EFEEEE (light gray).</td>
</tr>
</tbody>
</table>
The following style applies to the following tags and tag-attribute combinations:

- `textarea`
- `cfinput type="hidden"`
- `cfinput type="password"`
- `cfinput type="text"`

### Styles for cfselect with size attribute value of 1

The following styles apply to the `cfselect` tag when the `size` attribute is 1; that is, if the control displays one option at a time, with a drop-down list (also known as a combobox):

<table>
<thead>
<tr>
<th>Style</th>
<th>Inh</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>alternatingRowColors</td>
<td>Y</td>
<td>Format: comma delimited list of colors for rows in an alternating pattern. Value can be a list of two or more colors. Use only if you do not specify a <code>backgroundColor</code> style.</td>
</tr>
<tr>
<td>closeDuration</td>
<td>N</td>
<td>Time to close the drop-down list, in milliseconds. The default value is 250.</td>
</tr>
<tr>
<td>openDuration</td>
<td>N</td>
<td>Time to close the drop-down list, in milliseconds. The default value is 250.</td>
</tr>
<tr>
<td>rollOverColor</td>
<td>Y</td>
<td>Format: color; color of the background when the user rolls over an item. The default value is ##0EFFD6.</td>
</tr>
<tr>
<td>selectionColor</td>
<td>Y</td>
<td>Format: color; color of the background when the user selects an item. The default value is ##0DFFC1.</td>
</tr>
</tbody>
</table>

### Styles for cftextarea tag and cfinput with text, password, or hidden type attributes

The following style applies to the following tags and tag-attribute combinations:

- `textarea`
- `cfinput type="hidden"`
- `cfinput type="password"`
- `cfinput type="text"`

<table>
<thead>
<tr>
<th>Style</th>
<th>Inh</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>disabledColor</td>
<td>Y</td>
<td>Format: color; disabled color of the Text Area.</td>
</tr>
</tbody>
</table>

### Styles for cfselect with size attribute value of 1

The following styles apply to the `cfselect` tag when the `size` attribute is 1; that is, if the control displays one option at a time, with a drop-down list (also known as a combobox):

<table>
<thead>
<tr>
<th>Style</th>
<th>Inh</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>alternatingRowColors</td>
<td>Y</td>
<td>Format: comma delimited list of colors for rows in an alternating pattern. Value can be a list of two or more colors. Use only if you do not specify a <code>backgroundColor</code> style.</td>
</tr>
<tr>
<td>closeDuration</td>
<td>N</td>
<td>Time to close the drop-down list, in milliseconds. The default value is 250.</td>
</tr>
<tr>
<td>openDuration</td>
<td>N</td>
<td>Time to close the drop-down list, in milliseconds. The default value is 250.</td>
</tr>
<tr>
<td>rollOverColor</td>
<td>Y</td>
<td>Format: color; color of the background when the user rolls over an item. The default value is ##0EFFD6.</td>
</tr>
<tr>
<td>selectionColor</td>
<td>Y</td>
<td>Format: color; color of the background when the user selects an item. The default value is ##0DFFC1.</td>
</tr>
</tbody>
</table>
Styles for cfselect with size attribute value greater than 1

The following styles apply to the cfselect tag when the size attribute is greater than 1; that is, if the control is a list box that displays two or more options at a time:

<table>
<thead>
<tr>
<th>Style</th>
<th>Inh</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>alternatingRowColors</td>
<td>Y</td>
<td>Type: comma-delimited list of colors for rows in an alternating pattern. Value can be a list of two or more colors.</td>
</tr>
<tr>
<td>marginBottom</td>
<td>N</td>
<td>Format: length; number of pixels between the bottom of the row and the bottom of the text in the row. The default value is 0.</td>
</tr>
<tr>
<td>marginTop</td>
<td>N</td>
<td>Format: length; number of pixels between the top of the row and the top of the text in the row. The default value is 0.</td>
</tr>
<tr>
<td>rollOverColor</td>
<td>Y</td>
<td>Format: color; color of the background when the user moves the mouse pointer over the link. The default value is ##00EFFD6.</td>
</tr>
<tr>
<td>selectionColor</td>
<td>Y</td>
<td>Format: color; color of the background when the user selects the link. The default value is ##0DFFC1.</td>
</tr>
<tr>
<td>selectionDuration</td>
<td>N</td>
<td>The duration of the selection animation, in milliseconds. The default value is 250. Set to 0 to disable animation.</td>
</tr>
<tr>
<td>textRollOverColor</td>
<td>Y</td>
<td>Format: color; text color when the user moves the mouse pointer over the selection. The default value is ##02B33C.</td>
</tr>
<tr>
<td>textSelectedColor</td>
<td>Y</td>
<td>Format: color; text color when selected. The default value is ##005F33.</td>
</tr>
</tbody>
</table>

Styles for cfcalendar tag and cfinput with dateField type attribute

The following styles apply to the cfcalendar tag and dateField type of the cfinput tag:

<table>
<thead>
<tr>
<th>Style</th>
<th>Inh</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>headerColors</td>
<td>Y</td>
<td>Format: color; colors of the band at the top of the DateChooser control. Specify two values, separated by a comma. For a solid band, use the same color for both values. The default value is ##E6EEEE,##FFFFFF.</td>
</tr>
<tr>
<td>rollOverColor</td>
<td>Y</td>
<td>Format: color; color of the background when the user moves the mouse pointer over the DateField. The default value is ##E3FFD6.</td>
</tr>
<tr>
<td>selectionColor</td>
<td>Y</td>
<td>Format: color; color of the background when the user selects the DateField. The default value is ##CDFFC1.</td>
</tr>
<tr>
<td>todayColor</td>
<td>Y</td>
<td>Format: color; color of today’s date. The default value is ##2B333C.</td>
</tr>
</tbody>
</table>
Styles for the cfgrid tag

The following styles apply to the cfgrid tag:

<table>
<thead>
<tr>
<th>Style</th>
<th>Inh</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>horizontalAlign</td>
<td>N</td>
<td>Horizontal alignment of children in the container. The default value is left. Possible values are left, center, and right.</td>
</tr>
<tr>
<td>horizontalGap</td>
<td>N</td>
<td>Number of pixels between children in the horizontal direction. The default value is 8.</td>
</tr>
<tr>
<td>marginBottom</td>
<td>N</td>
<td>Number of pixels between the container’s bottom border and its content area. The default value is 0.</td>
</tr>
<tr>
<td>marginTop</td>
<td>N</td>
<td>Number of pixels between the container’s top border and its content area. The default value is 0.</td>
</tr>
<tr>
<td>verticalAlign</td>
<td>N</td>
<td>Vertical alignment of children in the container. The default value is top. Possible values are top, middle, and bottom.</td>
</tr>
<tr>
<td>verticalGap</td>
<td>N</td>
<td>Number of pixels between children in the vertical direction. The default value is 8.</td>
</tr>
</tbody>
</table>

Styles for the cftree tag

The following styles apply to the cftree tag:

<table>
<thead>
<tr>
<th>Style</th>
<th>Inh</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>alternatingRowColors</td>
<td>Y</td>
<td>Type: Array; colors for rows in an alternating pattern. Value can be an Array of two or more colors.</td>
</tr>
<tr>
<td>depthColors</td>
<td>Y</td>
<td>Type: Array; array of colors used in the Tree control, in descending order.</td>
</tr>
<tr>
<td>indentation</td>
<td>N</td>
<td>Indentation for each tree level, in pixels. The default value is 8.</td>
</tr>
<tr>
<td>openDuration</td>
<td>N</td>
<td>Format: time; length of an open or close transition, in milliseconds. The default value is 250.</td>
</tr>
<tr>
<td>rollOverColor</td>
<td>Y</td>
<td>Format: color; color of the background when the user moves the mouse pointer over the link. The default value is ##E3FFD6.</td>
</tr>
<tr>
<td>selectionColor</td>
<td>Y</td>
<td>Format: color; color of the background when the user selects the link. The default value is ##CDFFC1.</td>
</tr>
<tr>
<td>selectionDuration</td>
<td>N</td>
<td>The duration of the selection animation, in milliseconds. The default value is 250. Set to 0 to disable animation.</td>
</tr>
<tr>
<td>textRollOverColor</td>
<td>Y</td>
<td>Format: color; color of the text when the user moves the mouse pointer over the entry. The default value is ##02B33C.</td>
</tr>
<tr>
<td>textSelectedColor</td>
<td>Y</td>
<td>Format: color; color of the text when the user selects the entry. The default value is ##005F33.</td>
</tr>
</tbody>
</table>
This chapter describes the methods that you implement in Application.cfc to handle Macromedia ColdFusion MX 7 application events. It also describes the variables that you set in the CFC to configure application characteristics.

Contents

Application variables ................................................................. 945
Method summary ................................................................. 946
Method descriptions ......................................................... 948

Note: Although Windows is case-insensitive, Macromedia recommends that you always start the Application.cfc filename with an uppercase A. Both application.cfc and Application.cfc are reserved words.

Note: If your application has an Application.cfc, and an Application.cfm or onRequestend.cfm page, ColdFusion MX ignores the CFM pages.

Application variables

The This scope for the Application.cfc contains several built-in variables which correspond to the attributes that you set in the cfapplication tag. You set the values of these variables in the CFC initialization code, before you define the CFC methods. You can access the variables in any method.
The following table briefly describes the variables that you can set to control the application behavior. For more details, see the `cfapplication` tag.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>no name</td>
<td>The application name. If you do not set this variable, or set it to the empty string, your CFC applies to the unnamed application scope, which is the ColdFusion MX J2EE servlet context. For more information on unnamed scopes see “Sharing data between ColdFusion pages and JSP pages or servlets” in ColdFusion MX Developer’s Guide.</td>
</tr>
<tr>
<td>applicationTimeout</td>
<td>Administrator value</td>
<td>Life span, as a real number of days, of the application, including all Application scope variables. Use the CFML <code>CreateTimeSpan</code> function to generate this variable’s value.</td>
</tr>
<tr>
<td>clientManagement</td>
<td>Administrator value</td>
<td>Whether the application supports Client scope variables.</td>
</tr>
<tr>
<td>clientStorage</td>
<td>Administrator value</td>
<td>Where Client variables are stored; can be cookie, registry, or the name of a data source.</td>
</tr>
<tr>
<td>loginStorage</td>
<td>cookie</td>
<td>Whether to store login information in the Cookie scope or the Session scope.</td>
</tr>
<tr>
<td>sessionManagement</td>
<td>no</td>
<td>Whether the application supports Session scope variables.</td>
</tr>
<tr>
<td>sessionTimeout</td>
<td>Administrator value</td>
<td>Life span, as a real number of days, of the user session, including all Session variables. Use the CFML <code>CreateTimeSpan</code> function to generate this variable’s value.</td>
</tr>
<tr>
<td>setClientCookies</td>
<td>True</td>
<td>Whether to send CFID and CFTOKEN cookies to the client browser.</td>
</tr>
<tr>
<td>setDomainCookies</td>
<td>False</td>
<td>Whether to set CFID and CFTOKEN cookies for a domain (not just a host).</td>
</tr>
<tr>
<td>scriptProtect</td>
<td>Administrator value</td>
<td>Whether to protect variables from cross-site scripting attacks.</td>
</tr>
</tbody>
</table>

**Method summary**

The following table briefly describes the application event methods that you can implement in `Application.CFC`:

<table>
<thead>
<tr>
<th>Method name</th>
<th>Method runs when</th>
</tr>
</thead>
<tbody>
<tr>
<td>onApplicationEnd</td>
<td>The application ends: the application times out, or the server is stopped.</td>
</tr>
<tr>
<td>onApplicationStart</td>
<td>The application first starts: the first request for a page is processed or the first CFC method is invoked by an event gateway instance, or a web services or Macromedia Flash Remoting CFC.</td>
</tr>
<tr>
<td>onError</td>
<td>An exception occurs that is not caught by a try/catch block.</td>
</tr>
<tr>
<td>onRequest</td>
<td>The <code>onRequestStart</code> method finishes. (This method can filter request contents.)</td>
</tr>
<tr>
<td>onRequestEnd</td>
<td>All pages in the request have been processed:</td>
</tr>
</tbody>
</table>
All parameters to these methods are positional. You can use any names for these parameters.

When a request executes, ColdFusion MX runs the CFC methods in the following order:

1. onRequestStart
2. onSessionStart
3. onRequest
4. onRequestEnd
5. onApplicationStart (if not run before for this application)
6. onSessionStart (if not run before for this session)
7. onRequest
8. onRequestEnd
9. onApplicationEnd
10. onSessionEnd
11. onError

The onApplicationEnd, onSessionEnd, and onError CFCs are triggered by specific events.
onApplicationEnd

Description
Runs when an application times out or the server is shutting down.

Syntax
<cffunction name="onApplicationEnd" returnType="void">
  <cfargument name="ApplicationScope" required=true/>
  ...
</cffunction>

See also
onApplicationStart, Method summary, “Managing the application with Application.cfc” in Chapter 13, “Designing and Optimizing a ColdFusion Application,” in ColdFusion MX Developer’s Guide

Parameters
ColdFusion MX passes the following parameters to the method:

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ApplicationScope</td>
<td>The application scope.</td>
</tr>
</tbody>
</table>

Returns
This method does not return a value; do not use the cfreturn tag.

Usage
Use this method for any clean-up activities that your application requires when it shuts down, such as saving data in memory to a database, or to log the application end to a file. You cannot use this method to display data on a user page, because it is not associated with a request. The application ends, even if this method throws an exception.

If you call this method explicitly, ColdFusion does not end the application; it does execute the method code, but does not lock the Application scope while the method executes.

You must use the ApplicationScope parameter to access the application scope; you cannot reference the scope directly; for example, use Arguments.ApplicationScope.myVariable, not Application.myVariable. This method can access the Server scope directly, but it does not have access to Session or Request scopes.

Note: The application times out only if it is inactive for the time-out period. Sessions do not end, and the onSessionEnd method is not called when an application ends. For more information, see onSessionEnd.

Example
<cffunction name="onApplicationEnd">
  <cfargument name="ApplicationScope" required=true/>
  <cflog file="#This.Name#" type="Information" text="Application #Arguments.ApplicationScope.applicationname# Ended" />
</cffunction>
onApplicationStart

Description
Runs when ColdFusion MX receives the first request for a page in the application.

Syntax
<cffunction name="onApplicationStart" returntype="boolean">
...
<cfreturn Boolean>
</cffunction>

See also

Returns
A Boolean value: True if the application startup code ran successfully; False, otherwise. You do not need to explicitly return a True value if you omit the cffunction tag returntype attribute.

Usage
Use this method for application initialization code; for example, use it to set Application scope variables, to determine whether a required data source or other resource is available, or to log the application start. You do not have to lock the Application scope if you set Application variables in this method, and you can reference Application scope variables as you normally do; for example, as Application.myVariable.

This method can access the requested page’s Variables scope only if the Application.cfc file includes an onRequest method that calls the page.

If you call this method explicitly, ColdFusion does not start the application; it does execute the method code, but does not lock the Application scope while the method executes.

If this method throws an uncaught exception or returns False, the application does not start and ColdFusion MX does not process any pages in the application. In this case, ColdFusion will run the onApplicationStart method the next time a user requests a page in the application.

Example
The following example tests for the availability of a database. If the database is not available it reports and logs the error, and does not start the application; if it is available, the method initializes two Application scope variables.
<cffunction name="onApplicationStart">
<cftry>
<!--- Test whether the DB is accessible by selecting some data. --->
<cfquery name="testDB" dataSource="cfdocexamples" maxrows="2">
SELECT Emp_ID FROM employee
</cfquery>
<!--- If we get a database error, report an error to the user, log the error information, and do not start the application. --->
<cfcatch type="database">
</cffunction>
<cfoutput>
This application encountered an error<br>
Please contact support.
</cfoutput>
<cflog file="#This.Name#" type="error"
   text="cfdocexamples DB not available. message: #cfcatch.message#
   Detail: #cfcatch.detail# Native Error: #cfcatch.NativeErrorCode#" >
<cfreturn False>
</cfcatch>
</cftry>
<cflog file="#This.Name#" type="Information" text="Application Started">
<!--- You do not have to lock code in the onApplicationStart method that sets
   Application scope variables. --->
<cfscript>
  Application.availableResources=0;
  Application.counter1=1;
</cfscript>
<cfreturn True>
</cffunction>
onError

Description

Runs when an uncaught exception occurs in the application.

Syntax

```<cffunction name="onError" returnType="void">
  <cfargument name="Exception" required=true/>
  <cfargument name="EventName" type="String" required=true/>
  ...
</cffunction>```

See also

Method summary, “Handling errors in Application.cfc” in ColdFusion MX Developer's Guide

Parameters

ColdFusion MX passes the following parameters to the method:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exception</td>
<td>The ColdFusion MX Exception object. For information on the structure of this object, see the description of the cfcatch variable in the cfcatch description.</td>
</tr>
<tr>
<td>EventName</td>
<td>The name of the event handler that generated the exception. If the error occurs during request processing and you do not implement an onRequest method, this is the empty string.</td>
</tr>
</tbody>
</table>

Returns

This method does not return a value; do not use the cfreturn tag.

Usage

Use this method to handle errors in an application-specific manner. This method overrides any error handlers that you set in the ColdFusion MX Administrator or in cferror tags. It does not override try/catch blocks.

Whether the onError method can display output depends on where the error takes place, as follows:

- The onError method can display a message to the user if an error occurs during an onApplicationStart, onSessionStart, onRequestStart, onRequest, or onRequestEnd event method, or while processing a request.
- The onError method cannot display output to the user if the error occurs during an onApplicationEnd or onSessionEnd event method, because there is no available page context; however, it can log an error message.

If the onError event handler is triggered by a scope-specific event method, such as onSessionStart, the error prevents further processing at the level of that scope and any lower scopes. An onError event triggered by an onSessionStart method, for example, prevents further processing in the session, but not in the application.
If an exception occurs while processing the `onError` method, or if the `onError` method uses a `cfthrow` tag, the ColdFusion MX standard error handling mechanisms handle the exception. These mechanisms include: any error handlers specified by `cferror` tags in the Application.cfc initialization code, the site-wide error handler specified in the ColdFusion MX Administrator, and ColdFusion MX default error page. Therefore, you can use the `onError` method as a filter to handle selected errors, and use other ColdFusion error-handling techniques for the remaining errors.

Example

```cfc
<cffunction name="onError">
  <cfargument name="Exception" required=true/>
  <cfargument type="String" name="EventName" required=true/>
  <!--- Log all errors. --->
  <cflog file="#This.Name#" type="error"
    text="Event Name: #Arguments.Eventname#" >
  <cflog file="#This.Name#" type="error"
    text="Message: #Arguments.Exception.message#">
  <cflog file="#This.Name#" type="error"
    text="Root Cause Message: #Arguments.Exception.rootcause.message#">
  <!--- Display an error message if there is a page context. --->
  <cfif NOT (Arguments.EventName IS "onSessionEnd") OR
    (Arguments.EventName IS "onApplicationEnd")>
    <cfoutput>
      <h2>An unexpected error occurred.</h2>
      <p>Please provide the following information to technical support:</p>
      <p>Error Event: #Arguments.EventName#</p>
      <p>Error details:<br>
      <cfdump var=#Arguments.Exception#></p>
    </cfoutput>
  </cfif>
</cffunction>
```
onRequest

Description

Runs when a request starts, after the onRequestStart event handler. If you implement this method, it must explicitly call the requested page to process it.

Syntax

```xml
<cffunction name="onRequest" returnType="void">
  <cfargument name="targetPage" type="String" required=true/>
  ...
  <cfinclude template="#Arguments.targetPage#">
  ...
</cffunction>
```

See also

onRequestStart, onRequestEnd, Method summary, “Managing requests in Application.cfc” in ColdFusion MX Developer’s Guide

Parameters

ColdFusion MX passes the following parameters to the method:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>targetPage</td>
<td>Path from the web root to the requested page.</td>
</tr>
</tbody>
</table>

Returns

This method does not return a value; do not use the cfreturn tag.

Usage

This event handler provides an optional request filter mechanism for CFML page requests (that is, .cfm pages requested using a browser). Use it to intercept requests to target pages and override the default behavior of running the requested pages. The following rules specify where and how you use the onRequest method.

- Implement this method only if the following are true:
  - The directory, and any subdirectories affected by this Application.cfc contain CFM files and do not contain any CFC files that are intended to be accessed as web services, using Flash Remoting, or using an event gateway.
  - You want to intercept the request and process it in a special way.
- If you do not implement this method, ColdFusion automatically calls the target page (or the CFC for a web service, Flash Remoting, or event gateway event).
- If you implement this method, it must explicitly call the target page, normally by using a cfinclude tag.
- Do not implement the onRequest method in any Application.cfc file that affects .cfc files that implement web services, process Flash Remoting or event gateway requests; ColdFusion MX will not execute the requests if you implement this method.
• Code in this method that precedes the call to the target page can perform the same functions as the `onRequestStart` method, and shares the Variables scope with the target page.

• Code in this method that follows the call to the target page can perform the same functions as the `onRequestEnd` method, and shares the Variables scope with the target page.

• If you implement this method, you can also implement the `onRequestStart` and `onRequestEnd` methods.

You can use this method to do preprocessing that is required for all requests. Typical uses include filtering and modifying request page contents (such as removing extraneous white space), or creating a switching mechanism that determines the exact page to display based on available parameters.

**Example**

```cffunction name="onRequest">
  <cfargument name="targetPage" type="String" required=true/>
  <cfset var content="">
  <cfsavecontent variable="content">
    <cfinclude template="#Arguments.targetPage#">
  </cfsavecontent>
  <cfoutput>
    #replace(content, "report", "MyCompany Quarterly Report", "all")#
  </cfoutput>
</cffunction>```
onRequestEnd

**Description**

Runs at the end of a request, after all other CFML code.

**Syntax**

```cfml
<cffunction name="onRequestEnd" returnType="void">
  <cfargument type="String" name="targetPage" required=true/>
  ...
</cffunction>
```

**See also**

`onRequestStart`, `onRequest`, *Method summary*, “Managing requests in Application.cfc” in *ColdFusion MX Developer's Guide*

**Parameters**

ColdFusion MX passes the following parameters to the method:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>targetPage</td>
<td>Path from the web root to the requested page.</td>
</tr>
</tbody>
</table>

**Returns**

This method does not return a value; do not use the `cfreturn` tag.

**Usage**

This method has the same purpose as the onRequestEnd.cfm page. (You cannot use an onRequestEnd.cfm page if you have an Application.cfc file for your application.) This method runs before the request terminates; therefore, it can access the page context, and can generate output.

This method can be useful for gathering performance metrics, or for displaying dynamic footer information.

This method can access the requested page’s Variables scope only if the Application.cfc file includes an `onRequest` method that calls the page. You can use Request scope variables to share data with the requested page, even if the Application.cfc file does not have an `onRequest` method.

If you call this method explicitly, ColdFusion does not end the request, but does execute the method code.

**Example**

The following example displays one of two footer pages depending on whether the user has logged in:

The `onRequestEnd` method in Application.cfc contains the following code:

```cfml
<cffunction name="onRequestEnd">
  <cfargument type="String" name="targetPage" required=true/>
  <cfset theAuthuser=getauthuser()>
  <cfif theAuthUser NEQ ">
  ...  
</cfif>
</cffunction>
```
A very simple authuserfooter.cfm page consists of the following code:

```
<cfoutput>
  <h3>Thank you for shopping at our store, #theAuthUser#!</h3>
</cfoutput>
```

A very simple noauthuserfooter.cfm page consists of the following code:

```
<cfoutput>
  <h3>Remember, only registered users get all our benefits!</h3>
</cfoutput>
```

To test this example, implement code for logging in a user, or try the example with and without the following line in the onRequestStart Application.cfc method:

```
<cfloginuser name="Robert Smith" password="secret" roles="customer">
onRequestStart

Description
Runs when a request starts.

Syntax
```<cffunction name="onRequestStart" returnType="boolean">
<cfargument type="String" name="targetPage" required=true/>
...
<cfreturn Boolean>
</cffunction>```

See also
- `onRequest`, `onRequestEnd`, Method summary, “Managing requests in Application.cfc” in ColdFusion MX Developer’s Guide

Parameters
ColdFusion MX passes the following parameters to the method:

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>targetPage</td>
<td>Path from the web root to the requested page.</td>
</tr>
</tbody>
</table>

Returns
A Boolean value. Return False to prevent ColdFusion from processing the request. You do not need to explicitly return a True value if you omit the `cffunction` tag `returntype` attribute.

Usage
This method runs at the beginning of the request. It is useful for user authorization (login handling), and for request-specific variable initialization, such as gathering performance statistics.

If this method throws an exception (for example, if it uses the `cfthrow` tag), ColdFusion handles the error and does not process the request further.

If you call this method explicitly, ColdFusion does not start a request, but does execute the method code.

This method can access the requested page's Variables scope only if the Application.cfc file includes an `onRequest` method that calls the page. You can use Request scope variables to share data with the requested page even if Application.cfc does not have an `onRequest` method.

Example
This example uses the authentication code generated by the ColdFusion MX Dreamweaver Login wizard to ensure that the user is logged in. For Beta 2, the wizard generates code that is appropriate for Application.cfm only. To use this code with the Application.CFC, delete the generated Application.CFM.

```<cffunction name="onRequestStart">
<cfargument name="requestname" required=true/>
<!--- Authentication code, generated by the Dreamweaver Login wizard. --->
<cfinclude template="mm_wizard_application_include.cfm">```
<!--- Regular maintenance is done late at night. During those hours, tell people to come back later, and do not process the request further. --->
<cfscript>
   if ((Hour(now()) gt 1) and (Hour(now()) lt 3)) {
      WriteOutput("The system is undergoing periodic maintenance. Please return after 3:00 AM Eastern time.");
      return false;
   } else {
      this.start=now();
      return true;
   }
</cfscript>
</cffunction>
onSessionEnd

Description
Runs when a session ends.

Syntax
```cfc
<cffunction name="onSessionEnd" returnType="void">
  <cfargument name="SessionScope" required=True/>
  <cfargument name="ApplicationScope" required=False/>
  ...
</cffunction>
```

See also
onSessionStart, Method summary, “Managing sessions in Application.cfc” in ColdFusion MX Developer’s Guide

Parameters
ColdFusion MX passes the following parameters to the method:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SessionScope</td>
<td>The Session scope</td>
</tr>
<tr>
<td>ApplicationScope</td>
<td>The Application scope;</td>
</tr>
</tbody>
</table>

Returns
This method does not return a value; do not use the cfreturn tag.

Usage
Use this method for any clean-up activities when the session ends. A session ends when the session is inactive for the session time-out period or, if using J2EE sessions, the user closes the browser. You can, for example, save session-related data, such as shopping cart contents or whether the user has not completed an order, in a database, or do any other required processing based on the user's status. You might also want to log the end of the session, or other session related information, to a file for diagnostic use.

If you call this method explicitly, ColdFusion does not end the session; it does execute the method code, but does not lock the Session.

You cannot use this method to display data on a user page, because it is not associated with a request.

You can access shared scope variables as follows:

- You must use the SessionScope parameter to access the Session scope. You cannot reference the Session scope directly; for example, use Arguments.SessionScope.myVariable, not Session.myVariable.
• You must use the `ApplicationScope` parameter to access the Application scope. You cannot reference the Application scope directly; for example, use `Arguments.ApplicationScope.myVariable`, not `Application.myVariable`. Use a named lock when you reference variables in the Application scope, as shown in the example.

• You can access the Server scope directly; for example, `Server.myVariable`.

• You cannot access the Request scope.

Sessions do not end, and the `onSessionEnd` method is not called when an application ends. The `onSessionEnd` does not execute if there is no active application, however.

**Example**

The following method decrements an Application scope session count variable and logs the session length.

```cfc
<cffunction name="onSessionEnd">
  <cfargument name = "SessionScope" required=true/>
  <cfargument name = "AppScope" required=true/>
  <cfset var sessionLength = TimeFormat(Now() - SessionScope.started, "H:mm:ss")>
  <cflock name="AppLock" timeout="5" type="Exclusive">
    <cfset Arguments.AppScope.sessions = Arguments.AppScope.sessions - 1>
  </cflock>
  <clog file="#This.Name#" type="Information"
    text="#Session #Arguments.SessionScope.sessionId# ended. Length: #sessionLength# Active sessions: #Arguments.AppScope.sessions#>
</cffunction>
```
onSessionStart

Description
Runs when a session starts.

Syntax
<cffunction name="onSessionStart" returnType="void">
... </cffunction>

See also
onSessionEnd, Method summary, “Managing sessions in Application.cfc” in ColdFusion MX Developer’s Guide

Returns
This method does not return a value; do not use the cfreturn tag.

Usage
This method is useful for initializing Session scope data, such as a shopping cart, or setting session-specific Application scope variables, such as for tracking the number of active sessions. You never need to lock the Session scope to set its variables using this method.

If you call this method explicitly, ColdFusion does not start a session; it does execute the method code, but does not lock the Session scope.

This method can access the requested page’s Variables scope only if the Application.cfc file includes an onRequest method that calls the page.

Example
The following onSessionStart example initializes some Session scope variables and increments and Application scope counter of active sessions.
<cffunction name="onSessionStart">
<cfscript>
Session.started = now();
Session.shoppingCart = StructNew();
Session.shoppingCart.items = 0;
</cfscript>
<cflock scope="Application" timeout="5" type="Exclusive">
<cfset Application.sessions = Application.sessions + 1>
</cflock>
</cffunction>
CHAPTER 6
ColdFusion MX Event Gateway Reference

This chapter describes the Java interfaces available for building Macromedia ColdFusion MX 7 custom CFXs in Java.

Contents
Gateway development interfaces and classes ..................................................... 963
CFML CFEvent structure ............................................................................. 1004
IM gateway methods and commands ......................................................... 1004
SMS Gateway CFEvent structure and commands ..................................... 1042
CFML event gateway SendGatewayMessage data parameter ...................... 1052

Note: The following CFML functions also apply to gateway application development:
GetGatewayHelper, SendGatewayMessage.

Gateway development interfaces and classes
The ColdFusion MX event gateway system is defined in the coldfusion.eventgateway package. Gateway developers implement two interfaces and use several classes, as follows:

• Gateway interface
• GatewayHelper interface
• GatewayServices class
• CFEvent class
• Logger class
Gateway interface

coldfusion.eventgateway.Gateway

Interface for implementing ColdFusion MX event gateways.

A class that implements this interface defines a ColdFusion MX event gateway type that you can use in ColdFusion MX applications. The class must implement the following methods:

<table>
<thead>
<tr>
<th>Signature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GatewayName([String id[, String configFile]])</td>
<td>The gateway constructor.</td>
</tr>
<tr>
<td>String getGatewayID()</td>
<td>Returns the gateway ID.</td>
</tr>
<tr>
<td>GatewayHelper getHelper()</td>
<td>Returns an instance of the GatewayHelper class for this gateway type. instance, or null if the gateway does not have a GatewayHelper class.</td>
</tr>
<tr>
<td>int getStatus()</td>
<td>Gets the event gateway status.</td>
</tr>
<tr>
<td>String outgoingMessage(coldfusion.eventgateway.CFEvent cfmessage)</td>
<td>Handles a message sent by ColdFusion and processes it to send to a message receiver.</td>
</tr>
<tr>
<td>void restart()</td>
<td>Restarts a running event gateway.</td>
</tr>
<tr>
<td>void setCFCListeners(String[] listeners)</td>
<td>Identifies the CFCs that listen for incoming messages from the event gateway.</td>
</tr>
<tr>
<td>void setGatewayID(String id)</td>
<td>Sets the gateway ID that uniquely identifies the Gateway instance.</td>
</tr>
<tr>
<td>void start()</td>
<td>Starts the event gateway.</td>
</tr>
<tr>
<td>void stop()</td>
<td>Stops the event gateway.</td>
</tr>
</tbody>
</table>
**Constructor**

**Description**

Instantiates a gateway.

**Category**

Event Gateway Development

**Syntax**

```java
public void gatewayName()
public void gatewayName(String id)
public void gatewayName(String id, String configFile)
```

**See also**

`setGatewayID`, “Class constructor” in Chapter 45, “Creating Custom Event Gateways” in *ColdFusion MX Developer’s Guide*

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>The identifier for the gateway instance</td>
</tr>
<tr>
<td>configFile</td>
<td>The absolute path to the gateway configuration file.</td>
</tr>
</tbody>
</table>

**Usage**

If your gateway requires a configuration file, use the constructor with two parameters. Otherwise, you can use either the default constructor or the single parameter version; ColdFusion always uses the `setGatewayID` method to set the ID.

**Example**

The following example shows the two argument constructor implemented in the ColdFusion MX `SocketGateway` class:

```java
public SocketGateway(String id, String configpath) {
    propsFilePath=configpath;
    try {
        FileInputStream propsFile = new FileInputStream(propsFilePath);
        properties.load(propsFile);
        propsFile.close();
        this.loadProperties();
    }
    catch (FileNotFoundException f) {
        // do nothing. use default value for port.
    }
    catch (IOException e) {
        e.printStackTrace();
    }
    gatewayID = id;
    gatewayService = GatewayServices.getGatewayServices();
}
```
getGatewayID

Description
Returns the gateway ID that identifies the Gateway instance.

Category
Event Gateway Development

Syntax
public String getGatewayID()

See also
setGatewayID, “Providing Gateway class service and information routines” in Chapter 45, “Creating Custom Event Gateways” in *ColdFusion MX Developer’s Guide*

Usage
This method returns a string value that is set by the setGatewayID method.

Example
The following example is the ColdFusion MX SocketGateway class getGatewayID method:

```java
public String getGatewayID()
{
    return gatewayID;
}
```
getHelper

Description
Returns an instance of the gatewayHelper class, if any for the gateway type.

Category
Event Gateway Development

Syntax
```java
public GatewayHelper getHelper()
```

See also
- GatewayHelper interface; “Providing Gateway class service and information routines” in Chapter 45, “Creating Custom Event Gateways” in ColdFusion MX Developer’s Guide

Returns
A coldfusion.eventgateway.GatewayHelper class instance, or null if the gateway does not have a GatewayHelper class.

Usage
ColdFusion calls this method when a ColdFusion MX application calls the CFML GetGatewayHelper function. The application then uses the gatewayHelper object methods to call gateway-specific utility methods, such as instant message buddy management methods.

Example
The following example is the ColdFusion MX SocketGateway class getHelper method:
```java
public GatewayHelper getHelper()
{
    // SocketHelper class implements the GatewayHelper interface
    return new SocketHelper();
}
```
getStatus

Description

Returns the gateway status.

Category

Event Gateway Development

Syntax

public int getStatus()

See also

“Providing Gateway class service and information routines” in Chapter 45, “Creating Custom Event Gateways” in ColdFusion MX Developer’s Guide

Returns

An integer status value. The Gateway interface defines the following status constants:

- STARTING
- RUNNING
- STOPPING
- STOPPED
- FAILED

Example

The following example is the ColdFusion MX SocketGateway class gettStatus method:

```java
public int getStatus()
{
    return status;
}
```
outgoingMessage

Description
Sends a message from ColdFusion to a message receiver.

Category
Event Gateway Development

Syntax
public String outgoingMessage(coldfusion.eventgateway.CFEvent message)

See also
“Responding to a ColdFusion function or listener CFC” in Chapter 45, “Creating Custom Event Gateways” in ColdFusion MX Developer’s Guide

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>message</td>
<td>A coldfusion.eventgateway.CFEvent instance containing the message to be sent.</td>
</tr>
</tbody>
</table>

Returns
A gateway-specific string, such as a message ID or a status indicator.

Usage
This method handles a message sent by ColdFusion MX and processes it as needed by the gateway type to send a message to the (usually external) message receiver. ColdFusion MX calls this method when the listener method of a listener CFC returns a message or when a ColdFusion application calls the SendGatewayMessage function. ColdFusion MX passes the String returned by this method back as the return value of a CFML SendGatewayMessage function.

Example
The following example is the ColdFusion MX SocketGateway class outgoingMessage method:

```java
public String outgoingMessage(coldfusion.eventgateway.CFEvent cfmsg)
{
    String retcode="ok";
    // Get the table of data returned from the event handler
    Map data = cfmsg.getData();
    String message = (String) data.get("MESSAGE");
    // find the right socket to write to from the socketRegistry hashtable
    if (cfmsg.getOriginatorID() != null && message != null)
    {
        SocketServerThread st =
            ((SocketServerThread)socketRegistry.get(cfmsg.getOriginatorID()));
        if(st != null)
            st.writeOutput(message);
        else
        {
            log.error("Cannot send outgoing message. OriginatorID " +
                cfmsg.getOriginatorID() + " is not a valid socket id.");
        }
    }
}
```
else if (data.get("OriginatorID") != null && message != null)
{
    SocketServerThread st = ((SocketServerThread)socketRegistry.get(data.get("OriginatorID")));
    if (st != null)
    {
        st.writeOutput(message);
    }
    else
    {
        log.error("Cannot send outgoing message. OriginatorID '" +
        data.get("OriginatorID") + '" is not a valid socket id."");
        retcode="failed";
    }
}
else
{
    log.error("Cannot send outgoing message. OriginatorID/MESSAGE is not
    available.");
    retcode="failed";
}
return retcode;
restart

Description
Stops a gateway if it is running and starts it up.

Category
Event Gateway Development

Syntax
public void restart()

See also
start, stop

Usage
In most cases, you implement this method as a call to the stop method followed by a start method, but you may be able to optimize the restart method based on the type of gateway.

Example
The following example is the ColdFusion MX SocketGateway class restart method:

```java
public void restart()
{
    stop();
    start();
}
```
setCFCLListeners

Description
Sets the array of listener CFCs that the gateway sends messages to.

Category
Event Gateway Development

Syntax
public void setCFCListeners(String[] listeners)

See also
Constructor, getGatewayID, setCFCPath, “Providing Gateway class service and information routines” in Chapter 45, “Creating Custom Event Gateways” in ColdFusion MX Developer’s Guide

Parameters
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>listeners</td>
<td>Array of absolute file paths to CFCs to which the gateway forwards messages when it gets events.</td>
</tr>
</tbody>
</table>

Usage
When ColdFusion MX starts a gateway instance, it calls this method with the names in the instance's listener list in the ColdFusion MX Administrator. ColdFusion MX can also call this method if the ColdFusion MX Administrator listener list changes while the gateway is running.

Example
The following example is the ColdFusion MX SocketGateway class setCFCLListeners method:

```java
public void setCFCListeners(String[] listeners)
{
    ArrayList aListeners = new ArrayList();
    for(int i = 0; i<listeners.length; i++)
    {
        aListeners.add(listeners[i]);
    }
    // Try not to pull the rug out from underneath a running message
    synchronized (cfcListeners)
    {
        cfcListeners = aListeners;
    }
}
```
setGatewayID

Description
Sets the gateway ID that uniquely identifies the Gateway instance.

Category
Event Gateway Development

Syntax
public void setGatewayID(String id)

See also
Constructor, getGatewayID, “Providing Gateway class service and information routines” in Chapter 45, “Creating Custom Event Gateways” in ColdFusion MX Developer's Guide

Parameters
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>The identifier for this gateway instance.</td>
</tr>
</tbody>
</table>

Usage
This method sets a string value that is returned by the getGatewayID method. ColdFusion MX calls this method to set the gateway ID with the value specified in the gateway instance configuration in the ColdFusion MX Administrator before it starts the event gateway, even if the Gateway constructor also sets the ID.

Example
The following example is the ColdFusion MX SocketGateway class setGatewayID method:

```java
public void setGatewayID(String id)
{
    gatewayID = id;
}
```
start

Description
starts a gateway running.

Category
Event Gateway Development

Syntax
public void start()

See also

Usage
Start a gateway by performing any required initialization. This method starts any listener thread or threads that monitor the gateway’s event source. The ColdFusion MX Administrator calls this function when it starts a gateway instance.

This method should update the status information that is returned by the getStatus method to indicate when the gateway is starting and when the gateway is running.

The ColdFusion MX Administrator Gateway Types page lets you specify a time-out for the gateway startup, and whether to kill the gateway on startup time-out. If you enable the kill option and the start method does not return in the time-out period, ColdFusion MX will kill the thread that called this function.

Example
The following example is the ColdFusion MX SocketGateway class restart method:

```java
public void start() {
    status = STARTING;
    listening=true;
    // Start up event generator thread
    Runnable r = new Runnable() {
        public void run() {
            socketServer();
        }
    };
    Thread t = new Thread(r);
    t.start();
    status = RUNNING;
}
```
stop

Description

Stops a gateway if it is running.

Category

Event Gateway Development

Syntax

public void stop()

See also


Usage

Stops a gateway by performing any required clean-up operations. This method stops any listener thread or threads that monitor the gateway's event source and releases any other resources. The ColdFusion MX Administrator calls this function when it stops a gateway instance.

This method should update the status information that is returned by the getStatus method to indicate when the gateway is stopping and when the gateway is stopped.

Example

The following example is the ColdFusion MX SocketGateway class stop method:

```java
public void stop()
{
    status = STOPPING;
    listening=false;
    Enumeration e = socketRegistry.elements();
    while (e.hasMoreElements()) {
        try {
            ((SocketServerThread)e.nextElement()).socket.close();
        } catch (IOException e1) {
            e1.printStackTrace();
        }
    }
    if (serverSocket != null) {
        try {
            serverSocket.close();
        } catch (IOException e1) {
            serverSocket = null;
        }
    }
    status = STOPPED;
}
```
GatewayHelper interface

coldfusion.eventgateway.GatewayHelper

ColdFusion MX includes a coldfusion.eventgateway.GatewayHelper Java marker interface, with no methods. Implement this interface to define a class that provides gateway-specific utility methods to the ColdFusion application or listener CFC. For example, an instant messaging event gateway might use a helper class to provide buddy list management methods to the application. The Gateway class must implement a getHelper method that returns the helper class, or null if you do not implement the interface.

For information on GatewayHelper classes, see “GatewayHelper class” on page 1092.
GatewayServices class
coldfusion.eventgateway.GatewayServices

The Gateway class uses the coldfusion.eventgateway.GatewayServices class to interact with the ColdFusion event gateway services. This class has the following methods:

<table>
<thead>
<tr>
<th>Signature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GatewayServices <code>getGatewayServices()</code></td>
<td>Static method that returns the GatewayServices object.</td>
</tr>
<tr>
<td>boolean <code>addEvent(CFEvent msg)</code></td>
<td>Sends a CFEvent instance to ColdFusion for dispatching to a listener CFC.</td>
</tr>
<tr>
<td><code>coldfusion.eventgateway.Logger getLogger([[String logfile]])</code></td>
<td>Returns a ColdFusion logger object that the event gateway can use to log information in a file.</td>
</tr>
<tr>
<td>int <code>getMaxQueueSize()</code></td>
<td>Returns the maximum size of the ColdFusion event queue, as set in the ColdFusion MX Administrator.</td>
</tr>
<tr>
<td>int <code>getMaxQueueSize()</code></td>
<td>Returns the current size of the ColdFusion event queue that handles all messages for all gateways.</td>
</tr>
</tbody>
</table>
getGatewayServices

Description
Static method that returns the GatewayServices object. Gateway code can call this method at any
time, if required.

Category
Event Gateway Development

Syntax
GatewayServices getGatewayServices()

See also
“GatewayServices class” in Chapter 45, “Creating Custom Event Gateways” in ColdFusion MX
Developer’s Guide

Returns
The GatewayServices object.

Usage
Gateway constructors can call this method to get a convenient reference to the GatewayServices
class and its methods.

Example
The following Socket gateway constructor code sets the GatewayServices variable:

```java
public SocketGateway(String id)
{
    gatewayID = id;
    gatewayService = GatewayServices.getGatewayServices();
}
```

Calls to GatewayServices methods, such as the following, use the returned value.

```java
boolean sent = gatewayService.addEvent(event);
```
addEvent

Description
Sends a CFEvent instance to ColdFusion for dispatching to a listener CFC.

Category
Event Gateway Development

Syntax
boolean addEvent(CFEvent msg)

See also
gQueueSize, getMaxQueueSize, “Responding to incoming messages” in Chapter 45, “Creating Custom Event Gateways” in ColdFusion MX Developer’s Guide

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>msg</td>
<td>The CFEvent object containing the message to be queued for delivery to the listener CFC.</td>
</tr>
</tbody>
</table>

Returns
True if the event was added to the gateway services queue for delivery, false, otherwise. Therefore, a true response does not indicate that the message was delivered.

Usage
The event gateway must use this method to send incoming messages to the application for processing.

Example
The following example from the ColdFusion MX SocketGateway code sends an event to all listener CFCs:

```java
for (int i = 0; i < listeners.length; i++) {
    String path = listeners[i];
    CFEvent event = new CFEvent(gatewayID);
    Hashtable mydata = new Hashtable();
    mydata.put("MESSAGE", theInput);
    event.setData(mydata);
    event.setGatewayType("SocketGateway");
    event.setOriginatorID(theKey);
    event.setCfcMethod(cfcEntryPoint);
    event.setCfcTimeOut(10);
    if (path != null)
        event.setCfcPath(path);
    boolean sent = gatewayService.addEvent(event);
    if (!sent)
        log.error("SocketGateway(" + gatewayID + ") Unable to put message on event queue. Message not sent from "+ gatewayID + ". thread "+ theKey + ". Message was "+ theInput);
}
```
getLogger

Description
Returns a ColdFusion Logger object that the event gateway can use to log information in a file.

Category
Event Gateway Development

Syntax
coldfusion.eventgateway.Logger getLogger([String logfile])

See also

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>logfile</td>
<td>The name, without an extension, of a log file in the ColdFusion MX logs directory. ColdFusion MX automatically appends a .log extension to the name. If the file does not exist, ColdFusion MX creates it when it logs the first message. By default, ColdFusion logs to the eventgateway.log file.</td>
</tr>
</tbody>
</table>

Returns
A ColdFusion MX logger object to use in

Usage
The Logger class has five methods: debug, info, warn, error, and fatal, that correspond to the severity level that is set in the log message. Each method takes a message string, a Throwable class object, or both.

If you pass a Throwable object to these methods, ColdFusion MX writes the exception information in the exceptions.log file.

Example
The ColdFusion MX example DirectoryWatcherGateway includes the following line in the constructor to get a logger object:

```java
// We create our own log file, which will be named "watcher.log"
logger = gatewayService.getLogger("watcher");
```

The following code, from the start of the routine that loads information from the configuration file, uses this object to log the initialization.

```java
// Load the properties file to get our settings
protected void loadconfig() throws ServiceRuntimeException {
    // load config
    logger.info("DirectoryWatcher (+ gatewayID + ") Initializing DirectoryWatcher gateway with configuration file " + config);
    
    //
    //
    //
```


**getMaxQueueSize**

**Description**

Returns the maximum size of the ColdFusion event queue, as set in the ColdFusion MX Administrator.

**Category**

Event Gateway Development

**Syntax**

```java
int getMaxQueueSize()
```

**See also**

`addEvent`, `getQueueSize`

**Returns**

The integer maximum number of messages that the gateway services queue can hold.

**Usage**

If the queue length reaches this value, the `addEvent` method will not add its message to the processing queue. You can use this method and the `getQueueSize` method to control the rate of event queuing and to help diagnose any throughput problems in your gateways.

**Example**

The following example logs the queue size, maximum queue size, and other information if a `gatewayService.addEvent` method fails to queue a message for delivery to a listener CFC. (It uses an internal method to construct the error message string.)

```java
boolean sent = gatewayService.addEvent(cfmsg);
if (!sent)
{
    logger.error(RB.getString(this, "IMGateway.cantAddToQueue",
                        gatewayType, gatewayID, ((path != null) ? path : "default"),
                        Integer.ToString(gatewayService.getQueueSize()),
                        Integer.ToString(gatewayService.getMaxQueueSize())));
}
```
getQueueSize

Description

Returns the current size of the ColdFusion event queue that handles all messages for all gateways.

Category

Event Gateway Development

Syntax

```java
int getQueueSize()
```

See also

`addEvent`, `getMaxQueueSize`

Returns

The integer number of messages in the gateway message queue that are waiting to be delivered to CFCs.

Usage

You can use this method and the `getMaxQueueSize` method to control the rate of event queuing and to help diagnose any throughput problems in your gateways.

Example

The following example logs the queue size, maximum queue size, and other information if a `gatewayService.addEvent` method fails to queue a message for delivery to a listener CFC. (It uses an internal method to construct the error message string.)

```java
boolean sent = gatewayService.addEvent(cfmsg);
if (!sent)
{
    logger.error(RB.getString(this, "IMGateway.cantAddToQueue", 
        gatewayType, gatewayID, ((path != null) ? path : "default"), 
        Integer.ToString(gatewayService.getQueueSize()), 
        Integer.ToString(gatewayService.getMaxQueueSize()));
}
```
CFEvent class

coldfusion.gateway.CFEvent

The Gateway class sends and receives CFEvent instances to communicate with the ColdFusion listener CFC or application. The CFEvent instances correspond to CFML CFEvent structures that ColdFusion application listener CFC methods receive and contain the message structures that ColdFusion application code sends to the gateway.

- The Gateway notifies ColdFusion of a message by sending a CFEvent instance in GatewayServices.addEvent method.
- The Gateway receives a CFEvent instance when ColdFusion calls the gateway's outgoingMessage method.

The CFEvent Class extends the java.util.Hashtable class and has the following methods:

<table>
<thead>
<tr>
<th>Methods</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CFEvent(String gatewayID)</td>
<td>CFEvent constructor.</td>
</tr>
<tr>
<td>String getGatewayID()</td>
<td>Returns the gateway ID (set in the CFEvent constructor).</td>
</tr>
<tr>
<td>void setCFCMethod(String method) String getCFCMethod()</td>
<td>Sets or gets the name of the CFC method that receives an incoming message.</td>
</tr>
<tr>
<td>void setCFCPath(String path) String getCFCPath()</td>
<td>Sets or gets the path to the application listener CFC that processes the event.</td>
</tr>
<tr>
<td>void setCFCTimeout(String seconds) String getCFCTimeout()</td>
<td>Sets or gets the time-out, in seconds, for the listener CFC to process the event request.</td>
</tr>
<tr>
<td>void setData(Map data) Map getData()</td>
<td>Sets or gets the event data structure, which contains the message contents and any other gateway-specific information.</td>
</tr>
<tr>
<td>void setGatewayType(String type) String getGatewayType()</td>
<td>Sets or gets the event gateway type identifier, such as SMS.</td>
</tr>
<tr>
<td>void setOriginatorID(String id) String getOriginatorID()</td>
<td>Sets or gets the gateway- or protocol-specific Identity of the originator of a message.</td>
</tr>
</tbody>
</table>
CFEvent

Description

CFEvent constructor.

Category

Event Gateway Development

Syntax

CFEvent(String gatewayID)

See also


Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>gatewayID</td>
<td>The ID of the gateway. This parameter indicates the source of the message and must be the value that is passed in the Gateway constructor or set using the Gateway setGatewayID method.</td>
</tr>
</tbody>
</table>

Usage

This method creates a container for an event gateway message that you send to ColdFusion MX gateway services in a gatewayServices.addEvent method for delivery to a CFC listener method.

Example

The following example, based on code for the ColdFusion asynchronous CFML gateway, sends a message to that the gateway has received to a CFC:

```java
public String outgoingMessage(coldfusion.eventgateway.CFEvent cfmsg)
{
    // Get the data
    Map data = cfmsg.getData();
    boolean status = true;
    if (data != null)
    {
        // create an event
        CFEvent event = new coldfusion.eventgateway.CFEvent(gatewayID);
        // set the event field values
        event.setGatewayType("CFMLGateway");
        event.setOriginatorID("CFMLGateway");
        event.setData(data);
        // send it to the event service
        status = gatewayService.addEvent(event);
    }
    return new Boolean(status).ToString();
}
```
getCFCCMethod

Description
Gets the name of the CFC method that processes the message.

Category
Event Gateway Development

Syntax
String getCFCMethod()

See also

Returns
For incoming messages, the name of the method that gateway services will call in the listener CFC, as set by the setCFCMethod method. If setCFCMethod has not been called, returns null, and not onIncomingMessage, which ColdFusion gateway services uses by default. Outgoing messages that are returned by a CFC in response to an incoming message also have the CFC method name in this field if the gateway set the field on the incoming message.

Usage
Most event gateways do not need to use this method. This method could be useful if a gateway sends messages to multiple CFC Methods and must determine which method is responding.
**getCFPath**

**Description**

Gets the path to the listener CFC that processes this message.

**Category**

Event Gateway Development

**Syntax**

```java
String getCFPath()
```

**See also**


**Returns**

An absolute path to the application listener CFC that will process the event, as set by the setCFPath method. If the setCFPath method has not been called, returns null, not the path specified in the ColdFusion Administrator and used by default by gateway services. Outgoing messages that are returned by a CFC in response to an incoming message also have the CFC method name in this field if the gateway set the field on the incoming message.

**Usage**

Most event gateways do not need to use this method. This method could be useful if a gateway sends messages to multiple CFCs and must determine which CFC is responding.
getCFCTimeout

Description
Gets the time-out, in seconds, for the listener CFC to process the event request.

Category
Event Gateway Development

Syntax
String getCFCTimeout()

See also

Returns
The listener CFC time-out, in seconds, as set by the setCFCTimeout method, or null.

Usage
Most gateways do not need to use this function.

When ColdFusion calls a listener CFC method to process the event, and the CFC does not process the event in the specified time-out period, ColdFusion terminates the request and logs an error in application.log file. By default ColdFusion uses the Timeout Request value set on the Server Settings page in the ColdFusion MX Administrator.
getData

Description

Returns the data Map that contains the message contents and other gateway-specific information.

Category

Event Gateway Development

Syntax

Map getData()

See also


Returns

The event data structure, or null. This structure includes the message contents being passed by the gateway and any other gateway-specific information.

Usage

The contents of the data Map depends on the event gateway type. Typical fields include the message contents, originator ID, destination ID, and if a gateway (such as the ColdFusion MX SMS gateway) supports multiple commands, the command.

Note: The returned Map object has case-insensitive keys.

Example

The following outgoingMessage method from the SocketGateway example gateway gets the message contents from the CFEvent data field of an outgoing message. If the CFEvent object does not include an OriginatorID field, it also tries to get the originator ID from the data field.

```java
public String outgoingMessage(coldfusion.eventgateway.CFEvent cfmsg)
{
    String retcode="ok";
    // Get the table of data returned from the event handler
    Map data = cfmsg.getData();
    String message = (String) data.get("MESSAGE");
    // find the right socket to write to from the socketRegistry hashtable
    if (cfmsg.getOriginatorID() != null)
        ((SocketServerThread)socketRegistry.get(cfmsg.getOriginatorID())).writeOutput(message);
    else if (data.get("OriginatorID") != null)
        ((SocketServerThread)socketRegistry.get(data.get("OriginatorID"))).writeOutput(message);
    else {
        System.out.println("cannot send outgoing message. OriginatorID is not available.");
        retcode="failed";
    }
    return retcode;
}
```
**getGatewayID**

**Description**
Returns the gateway ID field of the CFEvent object.

**Category**
Event Gateway Development

**Syntax**
```
String getGatewayID(CFEvent event)
```

**See also**

**Returns**
The gateway ID of the CFEvent object, or null.

**Usage**
Most gateways do not need to use this method. The gateway ID is set in the **CFEvent constructor** and normally corresponds to the gateway that is handling the event.
getGatewayType

Description
Returns the gateway type field of the CFEvent object.

Category
Event Gateway Development

Syntax
String getGatewayType()

See also

Returns
The gateway type of the CFEvent object, or null.

Usage
Most gateways do not need to use this method.
getOriginatorID

Description
Identifies the originator of an incoming message. Some gateway types also use this field for the
destination of an outgoing message.

Category
Event Gateway Development

Syntax
String getOriginatorID()

See also
setOriginatorID, “CFML CEvent structure” on page 1004, “CFEvent class” in Chapter 45,
“Creating Custom Event Gateways” in ColdFusion MX Developer’s Guide

Returns
The protocol-specific identifier of the message originator, or null.

Example
The outgoingMessage method of the SocketGateway example gateway uses the
getOriginatorID method to determine the destination of an outgoing message. This way, a
listener CFC that sends a response back to the originator does not have to explicitly set a
destination in the return variable. If the field is empty, (as it is in messages sent by the CFML
SendGatewayMessage function) the gateway tries to get the destination from the CEvent data
field.

```java
public String outgoingMessage(coldfusion.eventgateway.CFEvent cfmsg)

    String retcode="ok";
    // Get the table of data returned from the event handler
    Map data = cfmsg.getData();
    String message = (String) data.get("MESSAGE");
    // find the right socket to write to from the socketRegistry hashtable
    if (cfmsg.getOriginatorID() != null)
        ((SocketServerThread) socketRegistry.get(cfmsg.getOriginatorID())).
            writeOutput(message);
    else if (data.get("OriginatorID") != null)
        ((SocketServerThread) socketRegistry.get(data.get("OriginatorID"))).
            writeOutput(message);
    else
        { System.out.println("cannot send outgoing message. OriginatorID is not
        available.");
            retcode="failed";
        }
    return retcode;
```
setCFCMethod

Description
Sets the name of the CFC method that will processes an incoming message.

Category
Event Gateway Development

Syntax
void setCFCMethod(String method)

See also

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>method</td>
<td>The method in the listener CFC that ColdFusion will call to process this event. If you do not use this method in your gateway, ColdFusion invokes the onIncomingMessage method.</td>
</tr>
</tbody>
</table>

Usage
Gateways that use a single CFC listener method do not need to use this method if the listener CFC method is named onIncomingMessage. For the sake of consistency, Macromedia recommends that any event gateway with a single listener not override this default.

A gateway, such as the ColdFusion MX XMPP gateway, that uses different listener methods for different message types uses this method to identify the destination method.

Example
The following example code comes from the ColdFusion XMPP gateway incoming message handler. It creates a CFEvent object and sets the method that will handle tests based on the message type.

```java
CFEvent cfmsg = new CFEvent(gatewayID);
        cfmsg.setOriginatorID(sender);
        cfmsg.setGatewayType(gatewayType);
        if(messageType == IMMessage.IM)
            {
                // default for normal messages
                cfmsg.setCfcMethod(onIncomingMessageFunction);
            }
        //if the message is an authorization request
        else if(messageType == IMMessage.AUTH_REQUEST)
            {
                cfmsg.setCfcMethod(onAddBuddyRequestFunction);
                message = "Requesting authorization to add " + recipient + " to " + sender + "'s buddy list and view " + recipient + "'s presence."
            }
        // Code snipped here for brevity.
```
setCFCPath

Description
Specifies the listener CFC that will process this event.

Category
Event Gateway Development

Syntax
void setCFCPath(String path)

See also

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>path</td>
<td>An absolute path to the application listener CFC that will process the event. If you do not call this method in your gateway, ColdFusion uses the first path configured for the event gateway instance on the Event Gateways page in the ColdFusion MX Administrator.</td>
</tr>
</tbody>
</table>

Usage
By default, ColdFusion delivers messages to the CFC in the first path configured for the event gateway instance on the Event Gateways page in the ColdFusion MX Administrator.

If your application supports multiple listener CFCs, use this method to set each listener CFC and then call the gatewayService.addEvent method to send the event to the CFC.

Example
The following example code is based on the Socket gateway processInput method that takes input from the socket and sends it to the CFC listener methods. The listeners variable contains an array of listener CFCS and is set by the gateway’s setCFCListeners method, which ColdFusion calls when it starts the gateway.

```java
for (int i = 0; i < listeners.length; i++)
{
    String path = listeners[i];
    CFEvent event = new CFEvent(gatewayID);
    Hashtable mydata = new Hashtable();
    mydata.put("MESSAGE", theInput);
    event.setData(mydata);
    event.setGatewayType("SocketGateway");
    event.setOriginatorID(theKey);
    event.setCFCMethod(cfcEntryPoint);
    event.setCFCTimeout(10);
    if (path != null)
        event.setCFCPath(path); boolean sent = gatewayService.addEvent(event);
}
```
setCFCTimeout

Description
Sets the time-out, in seconds, during which the listener CFC must process the event request and return before ColdFusion gateway services terminates the request.

Category
Event Gateway Development

Syntax
void setCFCTimeout(String timeout)

See also

Parameters
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>timeout</td>
<td>The CFC time-out period, in seconds.</td>
</tr>
</tbody>
</table>

Usage
When ColdFusion calls a listener CFC method to process the event, and the CFC does not return in the specified time-out period, ColdFusion terminates the request and logs an error in the application.log file.

If you do not use this method, ColdFusion uses the Timeout Request value set on the Server Settings page in the ColdFusion MX Administrator.

Use this method if your messages require a longer or shorter time-out period than standard ColdFusion MX HTML requests.

Example
The following example code is based on the Socket gateway processInput method that takes input from the socket and sends it to the CFC listener methods. It sets the CFC time-out to 10 seconds.

```java
for (int i = 0; i < listeners.length; i++)
{
    String path = listeners[i];
    CFEvent event = new CFEvent(gatewayID);
    Hashtable mydata = new Hashtable();
    mydata.put("MESSAGE", theInput);
    event.setData(mydata);
    event.setGatewayType("SocketGateway");
    event.setOriginatorID(theKey);
    event.setCfcMethod(cfcEntryPoint);
    event.setCfcTimeOut(10);
    if (path != null)
        event.setCfcPath(path);
    boolean sent = gatewayService.addEvent(event);
}
```
**setData**

**Description**

Adds the gateway-specific data, including any message contents, as a Java Map to the CFEvent object.

**Category**

Event Gateway Development

**Syntax**

```java
void setData(Map data)
```

**See also**


**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>data</code></td>
<td>The incoming message and any additional gateway-specific event data.</td>
</tr>
</tbody>
</table>

**Usage**

The number of fields and their contents depend on the event gateway type. The Map keys must be strings.

Because ColdFusion is not case sensitive, it converts the Map passed in the `setData` method to a case insensitive Map. As a result, do not create entries in the data with names that differ only in case.

**Example**

The following code shows the routine from the example JMS gateway that handles incoming messages. It puts the JMS message ID and contents in a data HashMap, and uses it in the `setData` method:

```java
public void handleMessage(String msg, String topicName, String msgID) {  
    coldfusion.eventgateway.Logger log = getGatewayServices().getLogger();  
    Map data = new HashMap();  
    CFEvent cfMsg = new CFEvent(getGatewayID());  
    data.put("msg", msg);  
    data.put("id", msgID);  
    cfMsg.setData(data);  
    cfMsg.setOriginatorID(topicName);  
    cfMsg.setGatewayType("JMS");  
    if (sendMessage(cfMsg)) {  
        log.info("Added message "+ msgID + ":" + msgID + " to queue.");  
    } else {  
        log.error("Failed to add message ":" + msgID + " to queue.");  
    }
}
```
setGatewayType

Description
Identifies the type of event gateway.

Category
Event Gateway Development

Syntax
void setGatewayType(String gatewayType)

See also

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>gatewayType</td>
<td>A gateway type identifier.</td>
</tr>
</tbody>
</table>

getGatewayType, “CFML CFEvent structure” on page 1004, “CFEvent class” in Chapter 45, “Creating Custom Event Gateways” in ColdFusion MX Developer’s Guide

Usage
For the sake of consistency, use the same name in this method and in the Type Name field when you add the event gateway type in the ColdFusion MX Administrator. Gateway application CFCs that handle multiple gateway types, such as those in an instant messaging application that handles multiple instant messaging providers, could use this field to determine the protocol type and any gateway type-specific actions.

Example
The following code shows the routine from the example JMS gateway that handles incoming messages. It sets the gateway type to JMS:

```java
public void handleMessage(String msg, String topicName, String msgID) {
    coldfusion.eventgateway.Logger log = getGatewayServices().getLogger();
    Map data = new HashMap();
    CFEvent cfMsg = new CFEvent(getGatewayID());
    data.put("msg", msg);
    data.put("id", msgID);
    cfMsg.setData(data);
    cfMsg.setOriginatorID(topicName);
    cfMsg.setGatewayType("JMS");
    if (sendMessage(cfMsg)) {
        log.info("Added message "+ msgID + " to queue.");
    } else {
        log.error("Failed to add message "+ msgID + " to queue.");
    }
}
```
setOriginatorID

Description
Identifies the originator of an incoming message.

Category
Event Gateway Development

Syntax
void setOriginatorID(String originatorID)

See also

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>originatorID</td>
<td>The gateway or protocol-specific ID of the message originator.</td>
</tr>
</tbody>
</table>

Example
The following code shows the routine from the example JMS gateway that handles incoming messages. It sets the originator ID to the name of the JMS topic that the gateway handles:

```java
public void handleMessage(String msg, String topicName, String msgID) {
    coldfusion.eventgateway.Logger log = getGatewayServices().getLogger();
    Map data = new HashMap();
    CFEvent cfMsg = new CFEvent(getGatewayID());
    data.put("msg", msg);
    data.put("id", msgID);
    cfMsg.setData(data);
    cfMsg.setOriginatorID(topicName);
    cfMsg.setGatewayType("JMS");
    if (sendMessage(cfMsg)) {
        log.info("Added message "+ msgID + " to queue.");
    } else {
        log.error("Failed to add message "+ msgID + " to queue.");
    }
}
```
Logger class

coldfusion.eventgateway.Logger

Note: This class is in the coldfusion.log package, not the coldfusion.eventgateway package, which contains all other event gateway-related interfaces and classes.

The Logger class logs messages to a file in the ColdFusion logs directory. (You set this directory on the ColdFusion MX Administrator Logging Settings page.) The coldfusion.eventgateway.GatewayServices.getLogger() method returns an instance of the Logger class. The Logger class has the following methods:

<table>
<thead>
<tr>
<th>Signature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>debug</td>
<td>Writes a debugging message to the log file.</td>
</tr>
<tr>
<td>error</td>
<td>Writes an error message to the log file.</td>
</tr>
<tr>
<td>fatal</td>
<td>Writes a fatal error to the log file.</td>
</tr>
<tr>
<td>info</td>
<td>Writes an informational message to the log file.</td>
</tr>
<tr>
<td>warn</td>
<td>Writes a warning message to the log file.</td>
</tr>
</tbody>
</table>
debug

Description
Writes a log entry with a debugging severity to the ColdFusion MX logger. The entry includes the severity, thread ID, date, time, and a text message.

Category
Event Gateway Development

Syntax
- `debug(String message)`
- `debug(Throwable th)`
- `debug(String message, Throwable th)`

See also
- error, fatal, info, warn, getLogger

See also

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>message</td>
<td>The message to include in the log entry.</td>
</tr>
<tr>
<td>th</td>
<td>A throwable object, normally an exception. ColdFusion MX logs the exception information in the exception.log file in the ColdFusion MX logs directory.</td>
</tr>
</tbody>
</table>

Usage

Use this method to send a debugging message to the ColdFusion MX logging subsystem.

By default, ColdFusion does not write debugging messages to the log file. To have debug messages appear in the log file, change the priority entry in `cf_root\lib\neo-logging.xml` (in the server configuration) or `cf_root\WEB-INF\cfusion\lib\neo-logging.xml` (in the J2EE configuration). Change the following entry:

```xml
<var name='priority'>
  <string>information</string>
</var>
```

To the following:

```xml
<var name='priority'>
  <string>debug</string>
</var>
```

With debug priority, ColdFusion MX writes messages with a severity of “debug” to the log file specified in the `getLogger` method that returned the Logger instance (or the default log file).

Example

The ColdFusion MX instant messaging gateways use the following line to log information about incoming administrative messages or errors only when debugging priority is on.

```java
// code to process incoming administrative messages or errors
logger.debug(gatewayType + "Gateway (" + gatewayID + ") admin message: " + msg.getMessage());
```
error

Description
Writes a log entry with an error severity to the ColdFusion MX logger. The entry includes the severity, thread ID, date, time, and a text message.

Category
Event Gateway Development

Syntax
error(String message)
error(Throwable th)
error(String message, Throwable th)

See also

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>message</td>
<td>The message to include in the log entry.</td>
</tr>
<tr>
<td>th</td>
<td>A throw object, normally an exception. ColdFusion MX logs the exception information in the exception.log file in the ColdFusion MX logs directory.</td>
</tr>
</tbody>
</table>

Usage
Use this method to send an error message to the ColdFusion MX logging subsystem. ColdFusion MX will write a message with a severity of “error” to the log file specified in the getLogger method that returned the Logger instance (or the default log file).

Example
The ColdFusion MX example SocketGateway class includes the following code in the outgoingMessage method. It writes an error message if the message’s originator ID does not correspond to an open socket.

```java
SocketServerThread st =
    ((SocketServerThread)socketRegistry.get(cfmsg.getOriginatorID()));
if(st != null)
    st.writeOutput(message);
else {
    log.error("Cannot send outgoing message. OriginatorID " +
        cfmsg.getOriginatorID() + " is not a valid socket id.");
    retcode="failed";
}
```
fatal

Description

Writes a log entry with a fatal severity to the ColdFusion MX logger. The entry includes the severity, thread ID, date, time, and a text message.

Category

Event Gateway Development

Syntax

fatal(String message)
fatal(Throwable th)
fatal(String message, Throwable th)

See also


Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>message</td>
<td>The message to include in the log entry.</td>
</tr>
<tr>
<td>th</td>
<td>A throwable object, normally an exception. ColdFusion MX logs the exception information in the exception.log file in the ColdFusion MX logs directory.</td>
</tr>
</tbody>
</table>

Usage

Use this method to send a fatal error message to the ColdFusion MX logging subsystem. ColdFusion MX will write messages with a severity of “fatal” to the log file specified in the getLogger method that returned the Logger instance (or the default log file).
info

Description

Writes a log entry with an information severity to the ColdFusion MX logger. The entry includes the severity, thread ID, date, time, and a text message.

Category

Event Gateway Development

Syntax

info(String message)
info(Throwable th)
info(String message, Throwable th)

See also

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>message</td>
<td>The message to include in the log entry.</td>
</tr>
<tr>
<td>th</td>
<td>A throwable object, normally an exception. ColdFusion MX logs the exception information in the exception.log file in the ColdFusion MX logs directory. Not normally used with this method.</td>
</tr>
</tbody>
</table>

Usage

Use this method to send an informational message to the ColdFusion MX logging subsystem. ColdFusion MX will write a messages with a severity of “information” to the log file specified in the getLogger method that returned the Logger instance (or the default log file).

ColdFusion normally logs all information severity messages, so you should not use this severity for debugging messages or for events that happen frequently.

Example

The ColdFusion MX example DirectoryWatcherGateway class includes the following line at the top of its loadconfig method that loads the gateway’s configuration file. It writes a message including the gateway ID and configuration file.

```
logger.info("DirectoryWatcher (" + gatewayID + ") Initializing
DirectoryWatcher gateway with configuration file " + config);
```
**warn**

**Description**

Writes a log entry with a warning severity to the ColdFusion MX logger. The entry includes the severity, thread ID, date, time, and a text message.

**Category**

Event Gateway Development

**Syntax**

```
warn(String message)
warn(Throwable th)
warn(String message, Throwable th)
```

**See also**


**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>message</td>
<td>The message to include in the log entry.</td>
</tr>
<tr>
<td>th</td>
<td>A throwable object, normally an exception. ColdFusion MX logs the exception information in the exception.log file in the ColdFusion MX logs directory.</td>
</tr>
</tbody>
</table>

**Usage**

Use this method to send a warning message to the ColdFusion MX logging subsystem. ColdFusion MX will write a message with a severity of “warning” to the log file specified in the getLogger method that returned the Logger instance (or the default log file).

**Example**

The ColdFusion MX example SocketWatcherGateway class includes the following code in its constructor to load a configuration file. If it cannot load the file, it converts the exception information to a string and logs a warning that includes the gateway ID, and the exception information. It also passes the exception to the warn method.

```java
propsFilePath=configPath;
try {
    FileInputStream propsFile = new FileInputStream(propsFilePath);
    properties.load(propsFile);
    propsFile.close();
    this.loadProperties();
} catch (IOException e) {
    // do nothing. use default value for port.
    log.warn("SocketGateway(" + gatewayID + ") Unable to read configuration file " + propsFilePath + ":[" + e.ToString() + ". Using default port.", e);
}
```
CFML CFEvent structure

The CFML listener CFC methods receive messages in the form of a CFEvent structure that corresponds to the CFEvent class that gateway developers use. This structure has the following fields. Some of the fields might not be used by all gateways. All fields contain text or numeric values except the Data field, which contains a structure.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GatewayID</td>
<td>The event gateway that sent the event or will handle the outgoing message. The value is the ID of an event gateway instance configured on the ColdFusion MX Administrator Gateways page. If the application calls the <code>SendGatewayMessage</code> function to respond to the event gateway, it uses this ID as the function's first parameter.</td>
</tr>
<tr>
<td>Data</td>
<td>A structure containing the event data, including the message. The Data structure contents depend on the event gateway type. This field corresponds to the <code>SendGatewayMessage</code> function's second parameter.</td>
</tr>
<tr>
<td>OriginatorID</td>
<td>The originator of the message. The value depends on the protocol or event gateway type. Some event gateways might require this value in response messages to identify the destination of the response. Identifies the sender of the message.</td>
</tr>
<tr>
<td>GatewayType</td>
<td>The type of event gateway, such as SMS. An application that can process messages from multiple event gateway types can use this field. This value is the gateway type name that is specified by the event Gateway class. It is not necessarily the same as the gateway type name in the ColdFusion MX Administrator.</td>
</tr>
<tr>
<td>CFCPATH</td>
<td>The location of the listener CFC. The listener CFC does not need to use this field.</td>
</tr>
<tr>
<td>CFCMethod</td>
<td>The listener method that ColdFusion invokes to process the event. The listener CFC does not need to use this field.</td>
</tr>
<tr>
<td>CFCTimeout</td>
<td>The time-out, in seconds, for the listener CFC to process the event request. The listener CFC does not need to use this field.</td>
</tr>
</tbody>
</table>

IM gateway methods and commands

The XMPP and IBM Sametime gateways implement CFC methods to receive messages, use the gatewayHelper object methods to manage the gateway, and use outgoing message commands to send messages. The following sections describe these methods and commands:

- IM Gateway CFC incoming message methods
- IM gateway message sending commands
- IM Gateway GatewayHelper class methods
**IM Gateway CFC incoming message methods**

You write the following CFC methods to handle incoming messages from an XMPP or Lotus Sametime instant messaging gateway.

*Note:* The method names assume a default gateway configuration. ColdFusion lets you change the method names and disable event types in the gateway configuration file.

<table>
<thead>
<tr>
<th>Method</th>
<th>Message type</th>
</tr>
</thead>
<tbody>
<tr>
<td>onAddBuddyRequest</td>
<td>Requests from other IM users to add the gateway ID as their buddy</td>
</tr>
<tr>
<td>onAddBuddyResponse</td>
<td>Responses from others to requests from your gateway to add them to your buddy lists. Also used by buddies to ask to be removed from your list.</td>
</tr>
<tr>
<td>onBuddyStatus</td>
<td>Online status information messages</td>
</tr>
<tr>
<td>onIMServerMessage</td>
<td>Error and administrative messages from the IM server</td>
</tr>
<tr>
<td>onIncomingMessage</td>
<td>Instant messages</td>
</tr>
</tbody>
</table>
onAddBuddyRequest

Description

Handles incoming requests for users to add the gateway user name as one of their buddies.

Syntax

onAddBuddyRequest(CFEvent)

See also

onIncomingMessage, onAddBuddyResponse, onBuddyStatus, onIMServerMessage

Parameters

The method must take one parameter, a CFEvent structure with the following fields:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>gatewayType</td>
<td>Gateway type, either XMPP or SAMETIME</td>
</tr>
<tr>
<td>gatewayID</td>
<td>The ID of the gateway instance, as configured in ColdFusion MX Administrator</td>
</tr>
<tr>
<td>originatorID</td>
<td>The IM ID of the message originator</td>
</tr>
<tr>
<td>cfcMethod</td>
<td>This CFC method; by default, onAddBuddyRequest.</td>
</tr>
<tr>
<td>data.MESSAGE</td>
<td>The message that was sent with the request</td>
</tr>
<tr>
<td>data.SENDER</td>
<td>The sender’s ID; identical to the originatorID field value</td>
</tr>
<tr>
<td>data.RECIPIENT</td>
<td>The recipient’s ID, as specified in the gateway’s configuration file</td>
</tr>
<tr>
<td>data.TIMESTAMP</td>
<td>The date and time when the message was sent</td>
</tr>
</tbody>
</table>

Returns

The function can optionally return a value to send a response message. The return structure must contain the following fields:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>command</td>
<td>One of the following:</td>
</tr>
<tr>
<td></td>
<td>• accept Accept the request to add you as a buddy. ColdFusion adds the user</td>
</tr>
<tr>
<td></td>
<td>to the permit list of users that can get status information.</td>
</tr>
<tr>
<td></td>
<td>• decline Deny request to add you as a buddy. ColdFusion adds the user to</td>
</tr>
<tr>
<td></td>
<td>the deny list of users that can get status information.</td>
</tr>
<tr>
<td></td>
<td>• noact Take no action. ColdFusion does not respond to the requestor.</td>
</tr>
</tbody>
</table>

buddyID    | ID to which to send the message. Normally, the value of the CFEvent.data.SENDER field. Not used with the noact command. |
reason     | A text message describing the reason for the action. Not used with the noact command. |
Example

The following example searches for the requested buddy's name in a data source and, if it finds a unique entry, adds the buddy and updates the buddy's status information in an Application scope buddyStatus structure. If it doesn't find the name, it declines the buddy. If there are multiple entries for the buddy name in the database, it tells the gateway not to respond. It logs all actions.

```cfc
<cffunction name="onAddBuddyRequest">
  <cfargument name="CFEvent" type="struct" required="YES">
  <cfquery name="buddysearch" datasource="cfdocexamples">
    select IM_ID
    from Employees
    where IM_ID = '#CFEvent.Data.SENDER#'
  </cfquery>
  <cflock scope="APPLICATION" timeout="10" type="EXCLUSIVE">
    <cfscript>
      // If the name is in the DB once, accept; if it is missing, decline.
      // If it is in the DB multiple times, take no action.
      if (buddysearch.RecordCount IS 0) {
        action='decline';
        reason='Invalid ID';
      } else if (buddysearch.RecordCount IS 1) {
        action='accept';
        reason='Valid ID';
        // Add the buddy to the buddy status structure only if accepted.
        if (NOT StructKeyExists(Application, "buddyStatus")) {
          Application.buddyStatus=StructNew();
        }
        if (NOT StructKeyExists(Application.buddyStatus, CFEvent.Data.SENDER)) {
          Application.buddyStatus[CFEvent.Data.SENDER]=StructNew();
        }
        Application.buddyStatus[CFEvent.Data.SENDER].status="Accepted Buddy Request";
        Application.buddyStatus[CFEvent.Data.SENDER].timeStamp=CFEvent.Data.TIMESTAMP;
        Application.buddyStatus[CFEvent.Data.SENDER].message=CFEvent.Data.MESSAGE;
      } else {
        action='noact';
      }
    </cfscript>
  </cflock>
  <!--- Log the request and decision information. --->
  <cflog file="#CFEvent.GatewayID#Status" text="onAddBuddyRequest; SENDER: #CFEvent.Data.SENDER# MESSAGE: #CFEvent.Data.MESSAGE# TIMESTAMP: #CFEvent.Data.TIMESTAMP# ACTION: #action#"/>
  <!--- Return the action decision. --->
  <cfset retValue = structNew()>
  <cfset retValue.command = action>
  <cfset retValue.BuddyID = CFEvent.DATA.SENDER>
</cffunction>
```
<cfset retValue.Reason = reason>
<cfreturn retValue>
</cffunction>
onAddBuddyResponse

Description
Handles incoming responses from other users to requests from the gateway to be added to their buddy lists. Also receives requests from buddies to have you remove them from your buddy list.

Syntax
onAddBuddyResponse(CFEvent)

See also
onIncomingMessage, onAddBuddyRequest, onBuddyStatus, onIMServerMessage

Parameters
The method must take one parameter, a CFEvent structure with the following fields:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>gatewayType</td>
<td>Gateway type, either XMPP or SAMETIME.</td>
</tr>
<tr>
<td>gatewayID</td>
<td>The ID of the gateway instance, as configured in ColdFusion MX Administrator.</td>
</tr>
<tr>
<td>originatorID</td>
<td>The IM ID of the message originator.</td>
</tr>
<tr>
<td>cfcMethod</td>
<td>This CFC method; by default, onAddBuddyResponse.</td>
</tr>
<tr>
<td>data.MESSAGE</td>
<td>One of the following:</td>
</tr>
<tr>
<td></td>
<td>• accept The request was accepted.</td>
</tr>
<tr>
<td></td>
<td>• decline The request was declined, or the buddy is asking you to remove them from your list.</td>
</tr>
<tr>
<td>data.SENDER</td>
<td>The sender's ID; identical to the originatorID.</td>
</tr>
<tr>
<td>data.RECIPIENT</td>
<td>The recipient's ID, as specified in the gateway's configuration file.</td>
</tr>
<tr>
<td>data.TIMESTAMP</td>
<td>The date and time when the message was sent.</td>
</tr>
</tbody>
</table>

Returns
The function does not return a value.

Example
The following example adds the buddy's status to the Application scope buddyStatus structure if the message sender accepted an add buddy request. It logs all responses.

```c 누구세요 "onAddBuddyResponse"
<cfargument name="CFEvent" type="struct" required="YES"/>
<cflock scope="APPLICATION" timeout="10" type="EXCLUSIVE">
<cfscript>
  //Do the following only if the buddy accepted the request.
  if (NOT StructKeyExists(Application, "buddyStatus")) {
    Application.buddyStatus=StructNew();
  }
  if (@CFEvent.Data.MESSAGE# IS "accept") {
    //Create a new entry in the buddyStatus record for the buddy.
    if (NOT StructKeyExists(Application.buddyStatus, CFEvent.Data.SENDER)) {
```
Application.buddyStatus[#CFEvent.Data.SENDER#]=StructNew();

// Set the buddy status information to indicate buddy was added.
Application.buddyStatus[#CFEvent.Data.SENDER#].status="Buddy accepted us";
Application.buddyStatus[#CFEvent.Data.SENDER#].timeStamp=CFEvent.Data.TIMESTAMP;
Application.buddyStatus[#CFEvent.Data.SENDER#].message=CFEvent.Data.MESSAGE;
}
</cfscript>
</cflock>
<!--- Log the information for all responses. --->
<cflog file="#CFEvent.GatewayID#Status"
  text="onAddBuddyResponse; BUDDY: #CFEvent.Data.SENDER# RESPONSE: 
#CFEvent.Data.MESSAGE# TIMESTAMP: #CFEvent.Data.TIMESTAMP#"> 
</cffunction>
onBuddyStatus

Description
Handles incoming messages indicating online status (presence) changes of users on the gateway’s buddy list.

Syntax
onBuddyStatus(CFEvent)

See also
onIncomingMessage, onAddBuddyRequest, onAddBuddyResponse, onIMServerMessage

Parameters
The method must take one parameter, a CFEvent structure with the following fields:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>gatewayType</td>
<td>Gateway type, either XMPP or SAMETIME.</td>
</tr>
<tr>
<td>gatewayID</td>
<td>The ID of the Gateway instance, as configured in ColdFusion MX Administrator.</td>
</tr>
<tr>
<td>originatorID</td>
<td>The IM ID (buddy name) of the message originator.</td>
</tr>
<tr>
<td>cfcMethod</td>
<td>This CFC method; by default, onIMServerMessage.</td>
</tr>
<tr>
<td>data.BUDDYNAME</td>
<td>The sender’s buddy name, or ID; identical to the originatorID.</td>
</tr>
<tr>
<td>data.BUDDYNICK</td>
<td>The buddy's display name or nickname.</td>
</tr>
<tr>
<td>data.BUDDYSTATUS</td>
<td>The buddy's status; one of the following:</td>
</tr>
<tr>
<td></td>
<td>• ONLINE</td>
</tr>
<tr>
<td></td>
<td>• OFFLINE</td>
</tr>
<tr>
<td></td>
<td>• AWAY</td>
</tr>
<tr>
<td></td>
<td>• DO NOT DISTURB XMPP only</td>
</tr>
<tr>
<td></td>
<td>• NOT AVAILABLE Sametime only</td>
</tr>
<tr>
<td></td>
<td>• FREE TO CHAT</td>
</tr>
<tr>
<td></td>
<td>• IDLE</td>
</tr>
<tr>
<td></td>
<td>Use the IMGatewayHelper getCusto...</td>
</tr>
<tr>
<td></td>
<td>awayMessage method to get any custom message that the buddy sent when changing status.</td>
</tr>
<tr>
<td>data.BUDDYGROUP</td>
<td>The group that the buddy belongs to.</td>
</tr>
<tr>
<td>data.RECIPIENT</td>
<td>The recipient's ID, as specified in the gateway's configuration file.</td>
</tr>
<tr>
<td>data.TIMESTAMP</td>
<td>The date and time when the message was sent.</td>
</tr>
</tbody>
</table>

Note: You configure the buddy’s nickname and group when you use the gatewayHelper object addBuddy method to add a buddy.

Returns
The function does not return a value.
Example

The following example keeps an Application scope structure up-to-date with a buddy's status. It also uses the gatewayhelper object getBuddyStatus method to get the buddy's custom away message, if any.

```coldfusion
<cffunction name="onBuddyStatus">
    <cfargument name="CFEvent" type="struct" required="YES">
    <!--- Get the gatewayhelper object and to get the info for this buddy. --->
    <!--- This is used to get the buddy's custom away message. --->
    <cfset helper = getGatewayHelper("MYIM" invokescript="false")>
    <cfset mybuddyinfo=helper.getBuddyInfo(CFEvent.Data.BUDDYNAME)>

    <cflog file="#CFEvent.GatewayID#Status" type="Information" text="In OnbuddyStatus, sender is #CFEvent.OriginatorID#">
    <cflock scope="APPLICATION" timeout="10" type="EXCLUSIVE">
        <cfscript>
            // Create the status structures if they don't exist.
            if (NOT StructKeyExists(Application,"buddyStatus")) {
                Application.buddyStatus=StructNew();
            }
            if (NOT StructKeyExists(Application.buddyStatus,
                CFEvent.Data.BUDDYNAME)) {
                Application.buddyStatus[#CFEvent.Data.BUDDYNAME#]=StructNew();
            }
            // Save the buddy status, timestamp, and custom away message
            Application.buddyStatus[#CFEvent.Data.BUDDYNAME#].status=CFEvent.Data.BUDDYSTATUS;
            Application.buddyStatus[#CFEvent.Data.BUDDYNAME#].timeStamp=CFEvent.Data.TIMESTAMP;
            // The following assumes that the buddy is in only one group.
            Application.buddyStatus[#CFEvent.Data.BUDDYNAME#].customAway=mybuddyinfo[1].BUDDYCUSTOMAWAYMESSAGE;
        </cfscript>
    </cflock>

    <!--- log the info, for debugging purposes only --->
    <cfset temp=Application.buddyStatus[#CFEvent.Data.BUDDYNAME#].status>
    <cflog file="#CFEvent.GatewayID#Status" type="Information" text="Application.buddyStatus[#CFEvent.Data.BUDDYNAME#].status is #temp#">
    <cfset temp=Application.buddyStatus[#CFEvent.Data.BUDDYNAME#].timeStamp>
    <cflog file="#CFEvent.GatewayID#Status" type="Information" text="Application.buddyStatus[#CFEvent.Data.BUDDYNAME#].timestamp is #temp#">
    <cflog file="#CFEvent.GatewayID#Status" type="Information" text="Buddy Custom Away Message is mybuddyinfo[1].BUDDYCUSTOMAWAYMESSAGE#">
</cffunction>
```
onIMServerMessage

Description

Handles incoming error and status messages from the IM server.

Syntax

onIMServerMessage(CFEvent)

See also

onIncomingMessage, onAddBuddyRequest, onAddBuddyResponse, onBuddyStatus

Parameters

This method must take one parameter, a CFEvent structure with the following fields:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>gatewayType</td>
<td>Gateway type, either XMPP or SAMETIME</td>
</tr>
<tr>
<td>gatewayID</td>
<td>The ID of the gateway instance, as configured in ColdFusion MX Administrator</td>
</tr>
<tr>
<td>originatorID</td>
<td>The IM ID (buddy name) of the message originator</td>
</tr>
<tr>
<td>cfcMethod</td>
<td>This CFC method; by default, onIMServerMessage</td>
</tr>
<tr>
<td>data.MESSAGE</td>
<td>The message sent by the server</td>
</tr>
<tr>
<td>data.SENDER</td>
<td>The sender's ID, identical to the originatorID</td>
</tr>
<tr>
<td>data.RECIPIENT</td>
<td>The recipient's ID, as specified in the gateway's configuration file</td>
</tr>
<tr>
<td>data.TIMESTAMP</td>
<td>The date and time when the message was sent</td>
</tr>
</tbody>
</table>

Example

The following example logs the sender, message, and a timestamp when an IM server sends an error or status message:

```cffunction name="onIMServerMessage">
<!--- This function just logs the message. --->
<cfargument name="CFEvent" type="struct" required="YES">
<cflog file="#CFEvent.GatewayID#Status" text="onIMServerMessage; SENDER: #CFEvent.OriginatorID#  MESSAGE: #CFEvent.Data.MESSAGE#  TIMESTAMP: #CFEvent.Data.TIMESTAMP#">
</cffunction>
**onIncomingMessage**

**Description**

Handles incoming instant messages from other users. Optionally returns a response to the message sender.

**Syntax**

```
onIncomingMessage(CFEvent)
```

**See also**

`onAddBuddyRequest, onAddBuddyResponse, onBuddyStatus, onIMServerMessage`, “Handling incoming messages” and “Sample IM message handling application” in Chapter 43, “Using the Instant Messaging Event Gateways” in ColdFusion MX Developer’s Guide

**Parameters**

The method must take one parameter, a CFEvent structure with the following fields:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>gatewayType</td>
<td>Gateway type, either XMPP or SAMETIME.</td>
</tr>
<tr>
<td>gatewayID</td>
<td>The ID of the Gateway instance as configured in ColdFusion MX Administrator.</td>
</tr>
<tr>
<td>originatorID</td>
<td>The IM ID of the message originator.</td>
</tr>
<tr>
<td>cfcMethod</td>
<td>This CFC method; by default, onIncomingMessage.</td>
</tr>
<tr>
<td>data.MESSAGE</td>
<td>The message that was received.</td>
</tr>
<tr>
<td>data.SENDER</td>
<td>The sender’s ID; identical to the originatorID</td>
</tr>
<tr>
<td>data.RECIPIENT</td>
<td>The recipient’s ID, as specified in the gateway’s configuration file</td>
</tr>
<tr>
<td>data.TIMESTAMP</td>
<td>The date and time when the message was sent</td>
</tr>
</tbody>
</table>

**Returns**

The function can optionally return a value to send a response message. The return structure must contain the following fields:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>command</td>
<td>Normally omitted. You can also specify submit.</td>
</tr>
<tr>
<td>buddyID</td>
<td>ID to which to send the message. Normally, the value of the input parameter’s Data.SENDER field.</td>
</tr>
<tr>
<td>message</td>
<td>The message contents.</td>
</tr>
</tbody>
</table>

**Example**

The following example shows a simple `onIncomingMessage` method that echoes a message back to the sender.

```
<cffunction name="onIncomingMessage">
  <cfargument name="CFEvent" type="struct" required="YES">
  <cfset input_msg = CFEvent.data.MESSAGE>
```

1014   Chapter 6: ColdFusion MX Event Gateway Reference
<cfset retValue = structNew()>
<cfset retValue.command = "submit">
<cfset retValue.buddyID = CFEvent.originatorID>
<cfset retValue.message = "Message Received:" & input_mesg>
<cfreturn retValue>
</cffunction>
IM gateway message sending commands

You use the `SendGatewayMessage` CFML function or the return value of a CFC listener method to send outgoing messages. The ColdFusion MX 7 IM gateway accepts the following outgoing message commands:

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>submit</td>
<td>(Default) Sends a normal message to another IM user.</td>
</tr>
<tr>
<td>accept</td>
<td>Accepts an add buddy request. Adds the buddy to the list of IDs that get your presence information and sends an acceptance message to the buddy ID.</td>
</tr>
<tr>
<td>decline</td>
<td>Declines an add buddy request and sends a rejection message to the buddy ID.</td>
</tr>
<tr>
<td>noact</td>
<td>Tells the gateway to take no action. The gateway logs a message that indicates that it took no action, and contains the gateway type, gateway ID, and buddy ID.</td>
</tr>
</tbody>
</table>

The message structure that you return in the gateway listener CFC function or use as the second parameter in the CFML `SendGatewayMessage` function can have the following fields. The table lists the fields and the commands in which they are used, and describes the field's use.

<table>
<thead>
<tr>
<th>Field</th>
<th>Commands</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>buddyID</td>
<td>All</td>
<td>The destination user ID</td>
</tr>
<tr>
<td>command</td>
<td>All</td>
<td>The command; defaults to submit if omitted</td>
</tr>
<tr>
<td>message</td>
<td>submit</td>
<td>A text message to send to the destination user</td>
</tr>
<tr>
<td>reason</td>
<td>accept, decline</td>
<td>A text description of the reason for the action or other message to send to the add buddy requestor</td>
</tr>
</tbody>
</table>

In typical use, a ColdFusion application uses the accept, decline, and noact commands in the return value of the `onAddBuddyRequest` method, and uses the submit command (or no command, because submit is the default command) in `SendGatewayMessage` CFML functions and the return value of the `onIncomingMessage` CFC method.
### IM Gateway GatewayHelper class methods

The GatewayHelper class returned by the CFML `GetGatewayHelper` function includes the following methods:

<table>
<thead>
<tr>
<th>Method</th>
<th>Method</th>
<th>Method</th>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>addBuddy</code></td>
<td><code>getDenyList</code></td>
<td><code>getStatusAsString</code></td>
<td><code>removeDeny</code></td>
</tr>
<tr>
<td><code>addDeny</code></td>
<td><code>getName</code></td>
<td><code>getStatusTimeStamp</code></td>
<td><code>removePermit</code></td>
</tr>
<tr>
<td><code>addPermit</code></td>
<td><code>getNickName</code></td>
<td><code>isOnline</code></td>
<td><code>setNickName</code></td>
</tr>
<tr>
<td><code>getBuddyInfo</code></td>
<td><code>getPermitList</code></td>
<td><code>numberOfMessagesReceived</code></td>
<td><code>setPermitMode</code></td>
</tr>
<tr>
<td><code>getBuddyList</code></td>
<td><code>getPermitMode</code></td>
<td><code>numberOfMessagesSent</code></td>
<td><code>setStatus</code></td>
</tr>
<tr>
<td><code>getCustomAwayMessage</code></td>
<td><code>getProtocolName</code></td>
<td><code>removeBuddy</code></td>
<td></td>
</tr>
</tbody>
</table>
**addBuddy**

**Description**

Adds a buddy to the buddy list for the gateway user ID and asks to have the IM server send messages with the buddy's online presence state to the gateway.

**Syntax**

```java
Boolean = addBuddy(name, nickname, group)
```

**See also**


**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>The unique instant messaging user name for the person about whom you want to receive periodic status messages.</td>
</tr>
<tr>
<td>nickname</td>
<td>The nickname that the application can use to refer to the user.</td>
</tr>
<tr>
<td>group</td>
<td>The name of the group you wish to add the user to in your Buddy List. If the group specified does not exist, it will be created. If the group parameter is the empty string, the gateway uses the General group.</td>
</tr>
</tbody>
</table>

**Returns**

True if the ID was added to the gateway's buddy list; False, otherwise.

**Usage**

This method adds the buddy to the buddy list for the gateway's ID and sends a subscription request (to automatically get presence information about the buddy's online status) to the remote buddy. It does not wait for a response from the buddy, so it returns True (and the gateway adds the buddy to the list) even if the buddy denies the subscription request. Use the listener CFC onAddBuddyResponse method to monitor the buddy's response. If the CFEvent.data.MESSAGE field value is decline, the listener method can call the gatewayHelper object removeBuddy method to remove the buddy from the buddy list.

**Example**

See “GatewayHelper example”, in Chapter 43, “Using the Instant Messaging Event Gateways” in ColdFusion MX Developer’s Guide, which uses all GatewayHelper class methods.
addDeny

Description

Tells the IM server to add the specified user to the deny list for the gateway's user ID. If the gateway's permit mode value is DENY_SOME, the specified user cannot receive messages on the gateway's presence state.

Syntax

Boolean = addDeny(name, nickname, group)

See also

addPermit, getDenyList, getPermitList, getPermitMode, removeDeny, removePermit, setPermitMode, “Using the GatewayHelper object” in Chapter 43, “Using the Instant Messaging Event Gateways” of ColdFusion MX Developer's Guide

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>The unique instant messaging user name for the person about whom you want to deny access to status messages.</td>
</tr>
<tr>
<td>nickname</td>
<td>The nickname that the application can use to refer to the user. Can be the empty string.</td>
</tr>
<tr>
<td>group</td>
<td>The name of the group that you want to add the user to in your buddy list. If the group specified does not exist, it is created. If the group parameter is the empty string, the gateway uses the General group.</td>
</tr>
</tbody>
</table>

Returns

True if the ID was added to the deny list; False, otherwise.

Note: XMPP permission management is included in the XMPP 1.0 draft specification, but several XMPP servers that were available at the time of the ColdFusion MX 7 release do not support permission management. If the server does not support permission management, this function always returns False.

Example

addPermit

Description

Tells the IM server to add the specified user to the permit list for the gateway's user ID. If the
gateway's permit mode is PERMIT_SOME, the specified user receive messages on the gateway’s
presence state.

Syntax

Boolean = addPermit(name, nickname, group)

See also

addDeny, getDenyList, getPermitList, getPermitMode, removeDeny, removePermit,
setPermitMode, “Using the GatewayHelper object” in Chapter 43, “Using the Instant Messaging
Event Gateways” in ColdFusion MX Developer’s Guide

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| name      | The unique instant messaging user name for the person about whom you want to
deny access to status messages. |
| nickname  | The nickname that the application can use to refer to the user. Can be the empty
string. |
| group     | The name of the group you want to add the user to in your Buddy List. If the group
specified does not exist, it is created. If the group parameter is the empty string, the
gateway uses the General group. |

Returns

True if the ID was added to the permit list; false, otherwise.

Note: XMPP permission management is included in the XMPP 1.0 draft specification, but several
XMPP servers that were available at the time of the ColdFusion MX 7 release do not support
permission management. If the server does not support permission management, this function
always returns False.

Example

See “GatewayHelper example”, in Chapter 43, “Using the Instant Messaging Event Gateways” in
ColdFusion MX Developer’s Guide, which uses all GatewayHelper class methods.
getBuddyInfo

Description

Gets information about the specified user from the buddy list, deny list, and permit list.

Syntax

array = getBuddyInfo(name)

See also

addBuddy, getBuddyList, removeBuddy, “Using the GatewayHelper object” in Chapter 43,
“Using the Instant Messaging Event Gateways” of ColdFusion MX Developer's Guide

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>The unique instant messaging user name for the person about whom you want to</td>
</tr>
<tr>
<td></td>
<td>get information.</td>
</tr>
</tbody>
</table>

Returns

An array of structures, with one structure for each information record found. The method finds
one record for each group that the user belongs to in each of the lists (buddy, permit, deny) that
contains the specified name. Each structure has the following fields. Some fields might not be
meaningful for some IM protocols. If there is no information for a field, it is blank.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BUDDYNAME</td>
<td>The user’s unique ID.</td>
</tr>
<tr>
<td>BUDDYGROUP</td>
<td>The group to which the user belongs.</td>
</tr>
<tr>
<td>BUDDYNICKNAME</td>
<td>The nickname that you have assigned to the user.</td>
</tr>
<tr>
<td>BUDDYPROTOCOL</td>
<td>The instant messaging protocol. JABBER (for XMPP) or SAMETIME, or an empty</td>
</tr>
<tr>
<td></td>
<td>string (if the server did not return a value).</td>
</tr>
<tr>
<td>BUDDYSTATUS</td>
<td>The user’s presence state, can by any of the following:</td>
</tr>
<tr>
<td></td>
<td>• ONLINE</td>
</tr>
<tr>
<td></td>
<td>• OFFLINE</td>
</tr>
<tr>
<td></td>
<td>• AWAY</td>
</tr>
<tr>
<td></td>
<td>• DND (displays as DO NOT DISTURB)</td>
</tr>
<tr>
<td></td>
<td>XMPP only</td>
</tr>
<tr>
<td></td>
<td>• NA (displays as NOT AVAILABLE)</td>
</tr>
<tr>
<td></td>
<td>• FREE_TO_CHAT (displays as FREE TO CHAT)</td>
</tr>
<tr>
<td></td>
<td>Sametime only</td>
</tr>
<tr>
<td></td>
<td>• IDLE</td>
</tr>
<tr>
<td>BUDDYSIGNONETIME</td>
<td>The date and time when the user signed onto the IM server. Empty if the</td>
</tr>
<tr>
<td></td>
<td>user is not currently signed on. Always an empty string for XMPP and</td>
</tr>
<tr>
<td></td>
<td>Sametime.</td>
</tr>
<tr>
<td>BUDDYSTATUSTIME</td>
<td>The date and time when the user’s status most recently changed.</td>
</tr>
</tbody>
</table>
Example

See “GatewayHelper example”, in Chapter 43, “Using the Instant Messaging Event Gateways” in ColdFusion MX Developer’s Guide, which uses all GatewayHelper class methods. For an example of using this method to get the buddy custom away message, see onBuddyStatus.
**getBuddyList**

**Description**

Gets the buddy list for the gateway's user ID.

**Syntax**

`array = getBuddyList()`

**See also**

`addBuddy, getBuddyInfo, removeBuddy`, “Using the GatewayHelper object” in Chapter 43, “Using the Instant Messaging Event Gateways” in *ColdFusion MX Developer’s Guide*

**Returns**

An array of IDs (buddy names) of the users on the gateway’s buddy list, a list of instant messaging IDs that this gateway normally communicates with.

**Example**

See “GatewayHelper example”, in Chapter 43, “Using the Instant Messaging Event Gateways” in *ColdFusion MX Developer’s Guide*, which uses all GatewayHelper class methods.
**getCustomAwayMessage**

**Description**

Returns the gateway's custom away message if it has been set by the `gatewayHelper` object `setStatus` method.

**Syntax**

```java
string = getCustomAwayMessage()
```

**See also**

`getStatusAsString`, `getStatusTimeStamp`, `isOnline`, `setStatus`, “Using the GatewayHelper object” in Chapter 43, “Using the Instant Messaging Event Gateways” of *ColdFusion MX Developer’s Guide*

**Returns**

The gateway’s custom away message if it has been set by the `GatewayHelper` object `setStatus` method.

**Example**

See “GatewayHelper example”, in Chapter 43, “Using the Instant Messaging Event Gateways” in *ColdFusion MX Developer’s Guide*, which uses all `GatewayHelper` class methods.
getDenyList

Description
Returns the list of users that the IM server has been told not to send state information about the
gateway, if the permit mode is set to DENY_SOME.

Syntax
array = getDenyList()

See also
addDeny, addPermit, getPermitList, getPermitMode, removeDeny, removePermit,
setPermitMode, “Using the GatewayHelper object” in Chapter 43, “Using the Instant Messaging
Event Gateways” of ColdFusion MX Developer’s Guide

Returns
An array of IDs (buddy names) of the users on the gateway’s deny list, the list of IDs to which the
IM server does not send presence status information.

Note: XMPP permission management is included in the XMPP 1.0 draft specification, but several
XMPP servers that were available at the time of the ColdFusion MX 7 release do not support
permission management. If the server does not support permission management, this function
always returns False.

Example
See “GatewayHelper example”, in Chapter 43, “Using the Instant Messaging Event Gateways” in
ColdFusion MX Developer’s Guide, which uses all GatewayHelper class methods.
getName

Description

Returns the gateway’s user name.

Syntax

```
string = getName()
```

See also

```
getProtocolName, numberOfMessagesReceived, numberOfMessagesSent, setNickName,
```
“Using the GatewayHelper object” in Chapter 43, “Using the Instant Messaging Event Gateways” of *ColdFusion MX Developer’s Guide*

Returns

The gateway’s user name, as specified in gateway configuration file.

Example

See “GatewayHelper example”, in Chapter 43, “Using the Instant Messaging Event Gateways” in *ColdFusion MX Developer’s Guide*, which uses all GatewayHelper class methods.
**getNickName**

**Description**

Returns the gateway's nickname (display name), if it has been set using the gatewayHelper object `setNickName` method.

**Syntax**

```plaintext
string = getNickName()
```

**See also**

`getName`, `getProtocolName`, `numberOfMessagesReceived`, `numberOfMessagesSent`, `setNickName`, “Using the GatewayHelper object” in Chapter 43, “Using the Instant Messaging Event Gateways” of *ColdFusion MX Developer’s Guide*

**Returns**

The gateway's nickname, if any; empty string, otherwise.

**Example**

See “GatewayHelper example”, in Chapter 43, “Using the Instant Messaging Event Gateways” in *ColdFusion MX Developer’s Guide*, which uses all GatewayHelper class methods.
getPermitList

**Description**

Returns the list of users that the IM server has been told to send state information about the gateway.

**Syntax**

```javascript
array = getPermitList()
```

**See also**

`addDeny, addPermit, getDenyList, getPermitMode, removeDeny, removePermit, setPermitMode`, “Using the GatewayHelper object” in Chapter 43, “Using the Instant Messaging Event Gateways” of *ColdFusion MX Developer's Guide*

**Returns**

An array of IDs (buddy names) of the users on the gateway's permit list, the list of IDs to which the IM server sends presence status information if the permit mode is set to PERMIT_SOME.

**Note:** XMPP permission management is included in the XMPP 1.0 draft specification, but several XMPP servers that were available at the time of the ColdFusion MX 7 release do not support permission management. If the server does not support permission management, this function always returns False.

**Example**

getPermitMode

Description

Gets the gateway's permit mode from the IM server. The permit mode determines whether all
users can get the gateway's online state information, or whether the server uses a permit list or a
deny list to control which users get state information.

Syntax

string = getPermitMode()

See also

addDeny, addPermit, getDenyList, getPermitList, removeDeny, removePermit,
setPermitMode, “Using the GatewayHelper object” in Chapter 43, “Using the Instant Messaging
Event Gateways” in ColdFusion MX Developer's Guide

Returns

The gateway's permit mode; one of the following values:

<table>
<thead>
<tr>
<th>Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PERMIT_ALL</td>
<td>(Default) Permits all users to be aware of the gateway's online presence and state.</td>
</tr>
<tr>
<td>PERMIT_SOME</td>
<td>Permits only users in the permit list to be aware of the gateway's online presence and state.</td>
</tr>
<tr>
<td>DENY_SOME</td>
<td>Prevents the users in the deny list from being aware of the gateway's online presence and state.</td>
</tr>
</tbody>
</table>

Note: XMPP permission management is included in the XMPP 1.0 draft specification, but several
XMPP servers that were available at the time of the ColdFusion MX 7 release do not support
permission management. If the server does not support permission management, this function
always returns PERMIT_ALL.

Example

See “GatewayHelper example”, in Chapter 43, “Using the Instant Messaging Event Gateways” in
ColdFusion MX Developer’s Guide, which uses all GatewayHelper class methods.
getProtocolName

Description

Gets the name of the gateway's instant messaging protocol.

Syntax

```
string = getProtocolName()
```

See also

```
getName, getNickName, numberOfMessagesReceived, numberOfMessagesSent, setNickName,
```

Returns

The gateway's protocol, as determined by the gateway type; one of the following values:

- JABBER (for XMPP)
- SAMETIME

Example

**getDescription**

**Description**

Gets the online status of the gateway as a text string.

**Syntax**

```
string = getStatusAsString()
```

**See also**

`getCustomAwayMessage, getStatusTimeStamp, isOnline, setStatus`, “Using the GatewayHelper object” in Chapter 43, “Using the Instant Messaging Event Gateways” in *ColdFusion MX Developer’s Guide*

**Returns**

The gateway’s online status; one of the following:

- ONLINE
- OFFLINE
- AWAY
- DO NOT DISTURB
- XMPP only
  - NOT AVAILABLE
  - FREE TO CHAT
- Sametime only
  - IDLE

**Usage**

The DO NOT DISTURB, NOT AVAILABLE, and FREE TO CHAT strings differ from the status values that you use in the `setStatus` method, which does not allow spaces in the status names.

**Example**

See “GatewayHelper example”, in Chapter 43, “Using the Instant Messaging Event Gateways” in *ColdFusion MX Developer’s Guide*, which uses all GatewayHelper class methods.
getStatusTimeStamp

Description

Gets the date and time that the gateway changed its online status.

Syntax

date-time object = getStatusTimeStamp()

See also

getCustomAwayMessage, getStatusAsString, isOnline, setStatus, “Using the GatewayHelper object” in Chapter 43, “Using the Instant Messaging Event Gateways” in ColdFusion MX Developer’s Guide

Returns

The date and time that the gateway changed its online status, normally by calling the setStatus gatewayHelper object method.

Example

See “GatewayHelper example”, in Chapter 43, “Using the Instant Messaging Event Gateways” in ColdFusion MX Developer’s Guide, which uses all GatewayHelper class methods.
**isOnline**

**Description**

Determines whether the gateway is connected to the instant messaging server.

**Syntax**

```
Boolean = isOnline()
```

**See also**


**Returns**

True, if the gateway is connected to the IM server; False, otherwise.

**Example**

See “GatewayHelper example”, in Chapter 43, “Using the Instant Messaging Event Gateways” in ColdFusion MX Developer’s Guide, which uses all GatewayHelper class methods.
**numberOfMessagesReceived**

**Description**

Gets the number of messages received by the gateway since it was started.

**Syntax**

```
integer = numberOfMessagesReceived()
```

**See also**

getName, getNickName, getProtocolName, numberOfMessagesSent, setNickName, “Using the GatewayHelper object” in Chapter 43, “Using the Instant Messaging Event Gateways” in ColdFusion MX Developer’s Guide

**Returns**

The number of messages received by the gateway since it was started.

**Example**

See “GatewayHelper example”, in Chapter 43, “Using the Instant Messaging Event Gateways” in ColdFusion MX Developer’s Guide, which uses all GatewayHelper class methods.
**numberOfMessagesSent**

**Description**

Gets the number of messages sent by the gateway since it was started.

**Syntax**

```java
Integer = numberOfMessagesSent()
```

**See also**

- `getName`, `getNickName`, `getProtocolName`, `numberOfMessagesReceived`, `setNickName`,

**Returns**

The number of messages sent by the gateway since it was started.

**Example**

removeBuddy

Description

Removes an ID from a group in the buddy list for the gateway and tells the IM server not to send the gateway messages with the buddy's online presence state.

Syntax

\[
\text{Boolean} = \text{removeBuddy}(\text{name}, \text{group})
\]

See also


Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>The unique instant messaging user name for the person to remove from the buddy list.</td>
</tr>
<tr>
<td>group</td>
<td>The name of the group from which you want to remove the user. If the parameter is the empty string, the gateway uses the General group.</td>
</tr>
</tbody>
</table>

Returns

True if the ID was removed from the group; False, otherwise.

Usage

If the user is in multiple groups in your buddy list, you remove the buddy separately from each group. The IM server does not stop sending status updates until you remove the name from all groups.

Example

See “GatewayHelper example”, in Chapter 43, “Using the Instant Messaging Event Gateways” in ColdFusion MX Developer’s Guide, which uses all GatewayHelper class methods.
removeDeny

**Description**

Removes an ID from a group in the deny list for the gateway. If the gateway’s permit mode is DENY_SOME, the specified user can receive messages on the gateway’s presence state.

**Syntax**

```java
Boolean = removeDeny(name, group)
```

**See also**

addDeny, addPermit, getDenyList, getPermitList, getPermitMode, removeBuddy, removePermit, setPermitMode, “Using the GatewayHelper object” in Chapter 43, “Using the Instant Messaging Event Gateways” in *ColdFusion MX Developer’s Guide*

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>The unique instant messaging user name for the person to remove from the deny list.</td>
</tr>
<tr>
<td>group</td>
<td>The name of the group from which you want to remove the user. If the parameter is the empty string, the gateway uses the General group.</td>
</tr>
</tbody>
</table>

**Returns**

True if the ID was removed from the group; False, otherwise.

**Note:** XMPP permission management is included in the XMPP 1.0 draft specification, but several XMPP servers that were available at the time of the ColdFusion MX 7 release do not support permission management. If the server does not support permission management, this function always returns False.

**Usage**

If the user is in multiple groups in your deny list, you remove the user separately from each group. The IM server enables sending status updates if you remove the name any group.

**Example**

See “GatewayHelper example”, in Chapter 43, “Using the Instant Messaging Event Gateways” in *ColdFusion MX Developer’s Guide*, which uses all GatewayHelper class methods.
removePermit

Description

Removes an ID from a group in the permit list for the gateway. If the gateway's permit mode is PERMIT_SOME, the specified user cannot receive messages on the gateway's presence state.

Syntax

`Boolean = removePermit(name, group)`

See also


Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>The unique instant messaging user name for the person to remove from the permit list.</td>
</tr>
<tr>
<td>group</td>
<td>The name of the group from which you want to remove the user. If the parameter is the empty string, the gateway uses the General group.</td>
</tr>
</tbody>
</table>

Returns

True if the ID was removed from the group; False, otherwise.

Note: XMPP permission management is included in the XMPP 1.0 draft specification, but several XMPP servers that were available at the time of the ColdFusion MX 7 release do not support permission management. If the server does not support permission management, this function always returns False.

Usage

If the user is in multiple groups in your permit list, you remove the user separately from each group. However, the IM server stops sending status updates when you remove the user from the first group.

Example

setNickName

Description
Sets the gateway’s nickname (display name).

Syntax
Boolean = setNickName(name)

See also
getName, getNickName, getProtocolName, numberOfMessagesReceived, numberOfMessagesSent, “Using the GatewayHelper object” in Chapter 43, “Using the Instant Messaging Event Gateways” in ColdFusion MX Developer’s Guide

Parameters
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>The display name that you want to associate with this gateway. This name is not guaranteed to be unique for the protocol.</td>
</tr>
</tbody>
</table>

Returns
True if the nickname got set; false, otherwise.

Example
See “GatewayHelper example”, in Chapter 43, “Using the Instant Messaging Event Gateways” in ColdFusion MX Developer’s Guide, which uses all GatewayHelper class methods.
setPermitMode

Description
Sets the gateway’s permit mode on the IM server. The permit mode determines whether all users can get the gateway’s online state information, or whether the server uses a permit list or a deny list to control which users get state information.

Syntax
```java
Boolean = setPermitMode(permitMode)
```

See also

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| permitMode | The permission mode, one of the following:
  - PERMIT_ALL  Permits all users to be aware of the gateway’s online presence and state. This is the default mode if you do not call this function.
  - PERMIT_SOME Permits only users in the permit list to be aware of the gateway’s online presence and state.
  - DENY_SOME Prevents all users in the deny list from being aware of the gateway’s online presence and state. |

Returns
True if the permit mode was set; False otherwise.

Note: XMPP permission management is included in the XMPP 1.0 draft specification, but several XMPP servers that were available at the time of the ColdFusion MX 7 release do not support permission management. If the server does not support permission management, this function returns False to all values except PERMIT_ALL.

Example
See “GatewayHelper example”, in Chapter 43, “Using the Instant Messaging Event Gateways” in ColdFusion MX Developer’s Guide, which uses all GatewayHelper class methods.
**setStatus**

**Description**
Sets the online presence status of the gateway, including any custom away message.

**Syntax**
```
Boolean = setStatus(status, customAwayMsg)
```

**See also**
getCustomAwayMessage, getStatusAsString, getStatusTimeStamp, isOnline, “Using the GatewayHelper object” in Chapter 43, “Using the Instant Messaging Event Gateways” in ColdFusion MX Developer's Guide

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>status</td>
<td>The gateway’s online presence status; one of the following:</td>
</tr>
<tr>
<td></td>
<td>• ONLINE</td>
</tr>
<tr>
<td></td>
<td>• AWAY</td>
</tr>
<tr>
<td></td>
<td>• DND (Do Not Disturb) XMPP only</td>
</tr>
<tr>
<td></td>
<td>• NA (Not Available)</td>
</tr>
<tr>
<td></td>
<td>• FREE_TO_CHAT Sametime only:</td>
</tr>
<tr>
<td></td>
<td>• IDLE</td>
</tr>
<tr>
<td>customAwayMsg</td>
<td>A text string containing a custom message for the status. Can be the empty</td>
</tr>
<tr>
<td></td>
<td>string if you do not need a custom away message.</td>
</tr>
</tbody>
</table>

**Returns**
True, if the operation was successful; False. otherwise. Passing an invalid status for the protocol causes this method to return False.

**Usage**
Do not use the setStatus method to go offline. Although the method accepts a parameter of OFFLINE, the gateway immediately resets itself to be online. To set the gateway offline, stop the gateway instance in the ColdFusion MX Administrator, or use the stopGatewayInstance method in the CFIDE.adminapi.eventgateway CFC.

**Example**
See “GatewayHelper example”, in Chapter 43, “Using the Instant Messaging Event Gateways” in ColdFusion MX Developer’s Guide, which uses all GatewayHelper class methods.
SMS Gateway CFEvent structure and commands

This section describes the detailed contents of the following structures that you use in the SMS Gateway listener CFCs and CFML SendGatewayMessage functions:

- SMS Gateway incoming message CFEvent structure
- SMS gateway message sending commands
The SMS gateway puts the following information in a CFEvent instance that it sends to the CFC listener method:

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>OriginatorID</td>
<td>Contents of the PDU source_addr field, the address of the device that sent the message.</td>
</tr>
<tr>
<td>CfcMethod</td>
<td>Listener CFC method name. Value of the configuration file cfc-method entry, or onIncomingMessage if the configuration file does not have this entry.</td>
</tr>
<tr>
<td>Data.MESSAGE</td>
<td>Contents of the short_message field of the PDU.</td>
</tr>
<tr>
<td>Data.sourceAddress</td>
<td>The address of the device that sent this message.</td>
</tr>
<tr>
<td>Data.destAddress</td>
<td>The address to which the message was sent; an address in the range specified by the gateway configuration file address-range setting.</td>
</tr>
</tbody>
</table>
| Data.esmClass     | Contents of the PDU esm_class field. Identifies the message type. A number in the range 0-255 representing a Byte value, where bits 2-5 (0-indexed) indicate the message type, and therefore the contents of the data.MESSAGE field, as follows. (Reserved values are omitted.)
|                   | xx000000   Normal message                                             |
|                   | xx000001   SMSC delivery receipt                                     |
|                   | xx000100   SME Delivery Acknowledgement                              |
|                   | xx010000   SME Manual/User Acknowledgement                           |
|                   | xx011000   Conversation abort (Korean CDMA only)                      |
|                   | xx100000   Intermediate Delivery Notification                       |
|                   | For more information on this field, see the SMPP specification.     |
| Data.protocol     | Contents of the PDU protocol_id field. Meaningful for messages sent from GSM networks only. For more information, see the GSM 03.40 specification. |
| Data.priority     | Contents of the PDU priority_flag field. A message priority level set by the originating SME, in the range 0-3; 0 is the lowest priority and 3 is the highest priority. The specific priority level meaning depends on the originating network. For more details, see the SMPP specification. |
| Data.registeredDelivery | Contents of the PDU registered_delivery field, indicating the type of delivery receipt or acknowledgement that the sender requested. A number in the range 0-32, representing the sum of the following values: 0 No SMS delivery receipt requested or 1 SMSC delivery receipt requested on delivery success or failure or 2 SMSC delivery receipt requested on delivery failure only Plus 0 No SME acknowledgement requested or 4 SME Delivery Acknowledgement requested or 8 SME Manual/User Acknowledgement requested or 12 Both Delivery and Manual/User Acknowledgements requested Plus 0 No Intermediate notification requested or 16 Intermediate notification requested |
For more information on the meanings of some of these fields and how to handle incoming SMS messages an SMS gateway listener CFC method, see “Handling incoming messages” in Chapter 44, “Using the SMS Event Gateway” in *ColdFusion MX Developer’s Guide*. 
**SMS gateway message sending commands**

ColdFusion applications that use gateways of the Short Message Service (SMS) type can send the following commands to the event gateway in an outgoing message:

- submit command
- submitMulti command
- data command
submit command

To send a message to a single destination address in an SMPP SUBMIT_SM PDU, the structure that you used in the Data parameter of a SendGatewayMessage function or the return variable of the CFC listener method has the following fields. For more information about these fields, see the documentation for the SUBMIT_MULTI PDU in the SMPP3.4 specification, which you can download from the SMS Forum at www.smsforum.net/.

**Required fields**

<table>
<thead>
<tr>
<th>Field</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>command</td>
<td>If present, the value must be submit. If you omit this field, the event gateway sends a submit message.</td>
</tr>
<tr>
<td>shortMessage or</td>
<td>The message contents. You must specify one of these fields, but not both. The SMPP specification imposes a maximum size of 254 bytes on the</td>
</tr>
<tr>
<td>messagePayload</td>
<td>shortMessage field, and some carriers might limit its size further. The messagePayload field can contain up to 64K bytes; it must start with 0x0424,</td>
</tr>
<tr>
<td></td>
<td>followed by two bytes specifying the payload length, followed by the message contents.</td>
</tr>
<tr>
<td>destAddress</td>
<td>Required. The address to which to send the message.</td>
</tr>
<tr>
<td>sourceAddress</td>
<td>The address of this application. You can omit this field; the configuration file specifies the application address.</td>
</tr>
</tbody>
</table>

**Optional fields**

You can set default values for the following optional fields in the SMS event gateway configuration file. For information on the default values, see “Configuring an SMS event gateway” in Chapter 44, “Using the SMS Event Gateway” in ColdFusion MX Developer’s Guide.

<table>
<thead>
<tr>
<th>Field</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>destAddress_npi</td>
<td>destAddress_ton serviceType</td>
</tr>
<tr>
<td>alertOnMsgDelivery</td>
<td>EsmClass priorityFlag smDefaultMsgId</td>
</tr>
<tr>
<td>callbackNum</td>
<td>ItsReplyType PrivacyIndicator SmsSignal</td>
</tr>
<tr>
<td>callbackNumAtag</td>
<td>ItsSessionInfo protocolId SourceAddrSubunit</td>
</tr>
<tr>
<td>callbackNumPresInd</td>
<td>LanguageIndicator registeredDelivery SourcePort</td>
</tr>
<tr>
<td>dataCoding</td>
<td>MoreMsgsToSend replacelfPresent SourceSubaddress</td>
</tr>
<tr>
<td>DestAddrSubunit</td>
<td>MsMsgWaitFacilities SarMsgRefNum UserMessageReference</td>
</tr>
<tr>
<td>DestinationPort</td>
<td>MsValidity SarSegmentSeqnum UserResponseCode</td>
</tr>
<tr>
<td>DestSubaddress</td>
<td>NumberOfMessages SarTotalSegments UssdServiceOp</td>
</tr>
<tr>
<td>DisplayTime</td>
<td>PayloadType scheduleDeliveryTime validityPeriod</td>
</tr>
</tbody>
</table>

The following optional fields do not have default values:
Example

The following example `onIncomingMessage` method of a listener CFC uses the submit command to echo incoming SMS messages to the message originator:

```
<cffunction name="onIncomingMessage" output="no">
  <cfargument name="CFEvent" type="struct" required="yes">
  <!--- Create a return structure that contains the message. --->
  <cfset retValue = structNew()>
  <cfset retValue.command = "submit">
  <cfset retValue.destAddress = arguments.CFEvent.originatorid>
  <!--- Send the message back. --->
  <cfreturn retValue>
</cffunction>
```
submitMulti command

To send a single text message to multiple recipients using an SMPP SUBMIT_MULTI PDU, the Data parameter of a SendGatewayMessage function or the return variable of the CFC listener method usually has the following fields. For more information about these fields, see the documentation for the SUBMIT_MULTI PDU in the SMPP3.4 specification, which you can download from the SMS Forum at www.smsforum.net/.

**Required fields**

<table>
<thead>
<tr>
<th>Field</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>command</td>
<td>Must be submitMulti.</td>
</tr>
<tr>
<td>shortMessage or messagePayload</td>
<td>The message contents. You must specify one of these fields, but not both. The SMPP specification imposes a maximum size of 254 bytes on the shortMessage field, and some carriers might limit its size further. The messagePayload field can contain up to 64K bytes; it must start with Ox0424, followed by two bytes specifying the payload length, followed by the message contents.</td>
</tr>
<tr>
<td>destAddress</td>
<td>A ColdFusion array of destination addresses (required). You cannot specify individual TON and NPI values for these addresses; all must conform to a single setting.</td>
</tr>
<tr>
<td>sourceAddress</td>
<td>The address of this application. You can omit this field; the configuration file specifies the application address.</td>
</tr>
</tbody>
</table>

**Optional fields**

The following optional fields can have default values set in the SMS event gateway configuration file. For information on the default values see “Configuring an SMS event gateway” in Chapter 44, “Using the SMS Event Gateway” in ColdFusion MX Developer's Guide.

<table>
<thead>
<tr>
<th>destAddress_npi</th>
<th>destAddress_ton</th>
<th>serviceType</th>
</tr>
</thead>
</table>

The following optional fields do not have default values:

<table>
<thead>
<tr>
<th>alertOnMsgDelivery</th>
<th>DisplayTime</th>
<th>protocolId</th>
<th>SmsSignal</th>
</tr>
</thead>
<tbody>
<tr>
<td>callbackNum</td>
<td>EsmClass</td>
<td>registeredDelivery</td>
<td>SourceAddrSubunit</td>
</tr>
<tr>
<td>callbackNumAtag</td>
<td>LanguageIndicator</td>
<td>replaceIfPresent</td>
<td>SourcePort</td>
</tr>
<tr>
<td>callbackNumPresInd</td>
<td>MsMsgWaitFacilities</td>
<td>SarMsgRefNum</td>
<td>SourceSubaddress</td>
</tr>
<tr>
<td>dataCoding</td>
<td>MsValidity</td>
<td>SarSegmentSeqnum</td>
<td>UserMessageReference</td>
</tr>
<tr>
<td>DestAddrSubunit</td>
<td>PayloadType</td>
<td>SarTotalSegments</td>
<td>validityPeriod</td>
</tr>
<tr>
<td>DestinationPort</td>
<td>priorityFlag</td>
<td>scheduleDeliveryTime</td>
<td></td>
</tr>
<tr>
<td>DestSubaddress</td>
<td>PrivacyIndicator</td>
<td>smDefaultMsgId</td>
<td></td>
</tr>
</tbody>
</table>
Example

The following example onIncomingMessage method sends a response that echoes an incoming message to the originator address, and sends a copy of the response to a second address:

<cffunction name="onIncomingMessage" output="no">
  <cfargument name="CFEvent" type="struct" required="yes" />
  <!--- Get the message. --->
  <cfset data=cfevent.DATA>
  <!--- Create the return structure. --->
  <cfset retValue = structNew()>
  <cfset retValue.command = "submitmulti"/>
  <cfset retValue.destAddresses = arrayNew(1)>
  <!--- One destination is incoming message originator; get the address from CFEvent originator ID. --->
  <cfset retValue.destAddresses[1] = arguments.CFEvent.originatorID>
  <cfset retValue.destAddresses[2] = "12345">  
  <cfset retValue.shortMessage = "echo: " & message>
  <cfreturn retValue>
</cffunction>
**data command**

To send binary data to a single destination address in an SMPP DATA_SM PDU, the *Data* parameter of a `SendGatewayMessage` function or the return variable of the CFC listener method must have the following fields. For more information about these fields, see the documentation for the SUBMIT_MULTI PDU in the SMPP3.4 specification, which you can download from the SMS Forum at [www.smsforum.net/](http://www.smsforum.net/).

### Required fields

<table>
<thead>
<tr>
<th>Field</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>command</td>
<td>Must be <code>data</code>.</td>
</tr>
<tr>
<td>messagePayload</td>
<td>The message data. To convert data to binary format, use the ColdFusion MX <code>ToBinary</code> function.</td>
</tr>
<tr>
<td>destAddress</td>
<td>The address to which to send the message.</td>
</tr>
<tr>
<td>sourceAddress</td>
<td>The address of this application. You can omit this field; the configuration file specifies the application address.</td>
</tr>
</tbody>
</table>

### Optional fields

The following optional fields can have default values set in the SMS event gateway configuration file. For information on the default values see “Configuring an SMS event gateway” in Chapter 44, “Using the SMS Event Gateway” in *ColdFusion MX Developer’s Guide*.

<table>
<thead>
<tr>
<th>Field</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>destAddress_npi</td>
<td>destAddress_ton</td>
</tr>
<tr>
<td>serviceType</td>
<td></td>
</tr>
</tbody>
</table>

The following optional fields do not have default values:

<table>
<thead>
<tr>
<th>Field</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>alertOnMsgDelivery</td>
<td>DestTelematicsId</td>
</tr>
<tr>
<td>callbackNum</td>
<td>DisplayTime</td>
</tr>
<tr>
<td>callbackNumAtag</td>
<td>EsmClass</td>
</tr>
<tr>
<td>callbackNumPresInd</td>
<td>ItsReplyType</td>
</tr>
<tr>
<td>dataCoding</td>
<td>ItsSessionInfo</td>
</tr>
<tr>
<td>DestAddrSubunit</td>
<td>LanguageIndicator</td>
</tr>
<tr>
<td>DestBearerType</td>
<td>MessageState</td>
</tr>
<tr>
<td>DestNetworkType</td>
<td>MoreMsgsToSend</td>
</tr>
<tr>
<td>DestinationPort</td>
<td>MsMsgWaitFacilities</td>
</tr>
<tr>
<td>DestSubaddress</td>
<td>MsValidity</td>
</tr>
<tr>
<td>NetworkErrorCode</td>
<td>ReceiptedMessageId</td>
</tr>
<tr>
<td>SetDpf</td>
<td>registeredDelivery</td>
</tr>
<tr>
<td>SmsSignal</td>
<td>SourceAddrSubunit</td>
</tr>
<tr>
<td>SourceBearerType</td>
<td>SourceTelematicsId</td>
</tr>
<tr>
<td>SourceNetworkType</td>
<td>SourcePort</td>
</tr>
<tr>
<td>SourceSubaddress</td>
<td>SourceSubaddress</td>
</tr>
<tr>
<td>SourceTelematicsId</td>
<td>SourceTelematicsId</td>
</tr>
<tr>
<td>SmsSignal</td>
<td>SarMsgRefNum</td>
</tr>
<tr>
<td>UserMessageReference</td>
<td>SarTotalSegments</td>
</tr>
<tr>
<td>UserResponseCode</td>
<td></td>
</tr>
</tbody>
</table>

1050 Chapter 6: ColdFusion MX Event Gateway Reference
Example

The following example `onIncomingMessage` method converts an incoming message to binary data, and sends the binary version of the message back to the originator address:

```cfnl
<cffunction name="onIncomingMessage" output="no">
  <cfargument name="CFEvent" type="struct" required="yes">
  <!--- Get the message --->
  <cfset data=CFEvent.DATA>
  <cfset message="#data.message#">
  <!--- Create the return structure --->
  <cfset retValue = structNew()>
  <cfset retValue.command = "data">
  <!--- Sending to incoming message originator; get value from CFEvent. --->
  <cfset retValue.destAddress = arguments.CFEvent.originatorid>
  <cfset retValue.messagePayload = tobinary(tobase64("echo: " & message))>
  <cfreturn retValue>
</cffunction>
```
CFML event gateway SendGatewayMessage data parameter

The ColdFusion CFML gateway type enables you to invoke CFC methods asynchronously. The structure that you use in the SendGatewayMessage function data parameter can include two types of fields:

- Any number of fields can contain arbitrary contents for use in by the CFC.
- Several optional fields can configure how the gateway delivers the information to the CFC.

The CFML gateway looks for the following optional fields, and, if they exist, uses them to determine how it delivers the message. Do not use these field names for data that you send to your CFC method.

<table>
<thead>
<tr>
<th>Field</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>cfcpath</td>
<td>Overrides the CFC path specified in the ColdFusion MX Administrator. This field lets you use a single gateway configuration in the ColdFusion MX Administrator multiple CFCs. This field sets the CFEvent object CFCPath variable.</td>
</tr>
<tr>
<td>method</td>
<td>Specifies the name of the method to invoke in the CFC. The default method is onIncomingMessage. This field lets you use a single gateway configuration in the ColdFusion MX Administrator for a CFC that has several methods. This field sets the CFEvent object CFCMethod variable.</td>
</tr>
<tr>
<td>originatorID</td>
<td>Sets the originatorID field of the CFEvent object that ColdFusion MX delivers to the CFC. The default value is CFMLGateway.</td>
</tr>
<tr>
<td>timeout</td>
<td>Sets the time-out, in seconds, during which the listener CFC must process the event request and return before ColdFusion gateway services terminates the request. The default value is the Timeout Request value set on the Server Settings page in the ColdFusion MX Administrator. Set this value if a request might validly take longer to process than the default time-out; for example, if the request involves a very long processing time. This field sets the CFEvent object CFCTimeout variable.</td>
</tr>
</tbody>
</table>

Example

The following example consists of a CFML page that sends a message to a logEvent method in the file logger.CFC. The CFML page specifies the CFC and method to call, and sets the OriginatorID.

```cfs
<h3>Sending an event using a generic CFML event gateway and specifying the CFC and method.</h3>
<cfscript>
status = False;
props = structNew();
props.cfcpath="C:\CFusionMX7\gateway\cfc\MyCFCs\logger.cfc";
props.method="logEvent";
props.OriginatorID=CGI.SCRIPT_NAME;
props.Message="Replace me with a variable with data to log";
props.file="GenericCFCtest";
props.type="warning";
status = SendGatewayMessage("DefaultCFC", props);
if (status IS True)
  WriteOutput('Event Message "#props.Message#" has been sent.');
</cfscript>
```
The CFC method uses the OriginatorID and the message, file, and type fields of the CFEvent parameter's data field to specify the log file and message.

```cfcomponent
<cfcomponent>
  <cffunction name="logEvent" output="no">
    <cfargument name="CFEvent" type="struct" required="yes">
    <cfscript>
      if (NOT IsDefined("CFEvent.Data.file")) {
        CFEvent.Data.file="defaultEventLog";
      }
      if (NOT IsDefined("CFEvent.Data.type")) {
        CFEvent.Data.type="information";
      }
    </cfscript>
    <cflog text="#CFEvent.originatorID": #CFEvent.Data.message#" file="#CFEvent.data.file#" type="#CFEvent.Data.type#">
  </cffunction>
</cfcomponent>
This chapter describes the Macromedia ColdFusion MX 7 CFXAPI classes and methods.

Contents

C++ class overview ................................................................. 1055
Deprecated class methods ......................................................... 1056
CCFXException class ................................................................. 1056
CCFXQuery class ................................................................. 1058
CCFXRequest class ................................................................. 1062
CCFXStringSet class ................................................................. 1072

C++ class overview

The following table lists the CFXAPI classes and methods:

<table>
<thead>
<tr>
<th>Class</th>
<th>Methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>CCFXException class</td>
<td>CCFXException::GetError</td>
</tr>
<tr>
<td></td>
<td>CCFXException::GetDiagnostics</td>
</tr>
<tr>
<td>CCFXQuery class</td>
<td>CCFXQuery::AddRow</td>
</tr>
<tr>
<td></td>
<td>CCFXQuery::GetColumns</td>
</tr>
<tr>
<td></td>
<td>CCFXQuery::GetData</td>
</tr>
<tr>
<td></td>
<td>CCFXQuery::GetRowName</td>
</tr>
<tr>
<td></td>
<td>CCFXQuery::GetRowCount</td>
</tr>
<tr>
<td></td>
<td>CCFXQuery::SetData</td>
</tr>
</tbody>
</table>
 Deprecated class methods

The following CFXAPI classes and methods are deprecated. They do not work, and might cause an error, in later releases.

<table>
<thead>
<tr>
<th>Class</th>
<th>Methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>CCFXRequest class</td>
<td>CCFXRequest::AddQuery</td>
</tr>
<tr>
<td></td>
<td>CCFXRequest::AttributeExists</td>
</tr>
<tr>
<td></td>
<td>CCFXRequest::CreateStringSet</td>
</tr>
<tr>
<td></td>
<td>CCFXRequest::Debug</td>
</tr>
<tr>
<td></td>
<td>CCFXRequest::GetAttribute</td>
</tr>
<tr>
<td></td>
<td>CCFXRequest::GetAttributeList</td>
</tr>
<tr>
<td></td>
<td>CCFXRequest::GetCustomData</td>
</tr>
<tr>
<td></td>
<td>CCFXRequest::GetQuery</td>
</tr>
<tr>
<td></td>
<td>CCFXRequest::ReThrowException</td>
</tr>
<tr>
<td></td>
<td>CCFXRequest::SetCustomData</td>
</tr>
<tr>
<td></td>
<td>CCFXRequest::SetVariable</td>
</tr>
<tr>
<td></td>
<td>CCFXRequest::ThrowException</td>
</tr>
<tr>
<td></td>
<td>CCFXRequest::Write</td>
</tr>
<tr>
<td></td>
<td>CCFXRequest::WriteDebug</td>
</tr>
<tr>
<td>CCFXStringSet class</td>
<td>CCFXStringSet::AddString</td>
</tr>
<tr>
<td></td>
<td>CCFXStringSet::GetCount</td>
</tr>
<tr>
<td></td>
<td>CCFXStringSet::GetIndexForString</td>
</tr>
<tr>
<td></td>
<td>CCFXStringSet::GetString</td>
</tr>
</tbody>
</table>

Class Deprecated member Deprecated as of this ColdFusion release

<table>
<thead>
<tr>
<th>Class</th>
<th>Deprecated member</th>
<th>Deprecated as of this ColdFusion release</th>
</tr>
</thead>
<tbody>
<tr>
<td>CCFXQuery Class</td>
<td>CCFXQuery::SetQueryString</td>
<td>ColdFusion MX</td>
</tr>
<tr>
<td></td>
<td>CCFXQuery::SetTotalTime</td>
<td>ColdFusion MX</td>
</tr>
<tr>
<td>CCFXRequest Class</td>
<td>CCFXRequest::GetSetting</td>
<td>ColdFusion MX</td>
</tr>
</tbody>
</table>

CCFXException class

An abstract class that represents an exception thrown during processing of a ColdFusion Extension (CFX) procedure.

Exceptions of this type can be thrown by CCFXRequest class, CCFXQuery class, and CCFXStringSet class. Your ColdFusion Extension code must be written to handle exceptions of this type. For more information, see CCFXRequest::ThrowException and CCFXRequest::ReThrowException.

Class methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>virtual LPCSTR GetError()</td>
<td>The CCFXException::GetError function returns a general error message.</td>
</tr>
<tr>
<td>virtual LPCSTR GetDiagnostics()</td>
<td>The CCFXException::GetDiagnostics function returns detailed error information.</td>
</tr>
</tbody>
</table>
CCFXException::GetError

Description

Provides basic user output for exceptions that occur during processing.

CCFXException::GetDiagnostics

Description

Provides detailed user output for exceptions that occur during processing.

Example

This code block shows how GetError and GetDiagnostics work with ThrowException and ReThrowException.

```cpp
// Write output back to the user here...
pRequest->Write( "Hello from CFX_FOO2!" );
pRequest->ThrowException( "User Error", "You goof'd...");

// Output optional debug info
if ( pRequest->Debug() )
{
    pRequest->WriteDebug( "Debug info..." );
}

// Catch ColdFusion exceptions & re-raise them
catch( CCFXException* e )
{
    // This is how you would pull the error information
    LPCTSTR strError = e->GetError();
    LPCTSTR strDiagnostic = e->GetDiagnostics();

    pRequest->ReThrowException( e );
}

// Catch ALL other exceptions and throw them as
// ColdFusion exceptions (DO NOT REMOVE! --
// this prevents the server from crashing in
// case of an unexpected exception)
catch( ... )
{
    pRequest->ThrowException(
        "Error occurred in tag CFX_FOO2",
        "Unexpected error occurred while processing tag." );
}
```
CCFXQuery class

An abstract class that represents a query used or created by a ColdFusion Extension (CFX). Queries contain one or more columns of data that extend over a varying number of rows.

Class methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>virtual int AddRow()</td>
<td>CCFXQuery::AddRow adds a row to a query.</td>
</tr>
<tr>
<td>virtual CCFXStringSet* GetColumns</td>
<td>CCFXQuery::GetColumns retrieves a list of a query’s column names.</td>
</tr>
<tr>
<td>virtual LPCSTR GetData( int iRow, int iColumn )</td>
<td>CCFXQuery::GetData retrieves a data element from a row and column of a query.</td>
</tr>
<tr>
<td>virtual LPCSTR GetName()</td>
<td>CCFXQuery::GetName retrieves the name of a query.</td>
</tr>
<tr>
<td>virtual int GetRowCount()</td>
<td>CCFXQuery::GetRowCount retrieves the number of rows in a query.</td>
</tr>
<tr>
<td>virtual void SetData( int iRow, int iColumn, LPCSTR lpszData )</td>
<td>CCFXQuery::SetData sets a data element within a row and column of a query.</td>
</tr>
<tr>
<td>virtual void SetQueryString( LPCSTR lpszQuery )</td>
<td>This function is deprecated. It might not work, and might cause an error, in later releases.</td>
</tr>
<tr>
<td>virtual void SetTotalTime( DWORD dwMilliseconds )</td>
<td>This function is deprecated. It might not work, and might cause an error, in later releases.</td>
</tr>
</tbody>
</table>

CCFXQuery::AddRow

Syntax

```
int CCFXQuery::AddRow(void)
```

Description

Add a row to the query. Call this function to append a row to a query.

Returns

Returns the index of the row that was appended to a query.
Example

The following example shows the addition of two rows to a three-column ('City', 'State', and 'Zip') query:

// First row
int iRow ;
iRow = pQuery->AddRow() ;
pQuery->SetData( iRow, iCity, "Minneapolis" ) ;
pQuery->SetData( iRow, iState, "MN" ) ;
pQuery->SetData( iRow, iZip, "55345" ) ;

// Second row
iRow = pQuery->AddRow() ;
pQuery->SetData( iRow, iCity, "St. Paul" ) ;
pQuery->SetData( iRow, iState, "MN" ) ;
pQuery->SetData( iRow, iZip, "55105" ) ;

CCFXQuery::GetColumns

Syntax

CCFXStringSet* CCFXQuery::GetColumns(void)

Description

Retrieves a list of the column names contained in a query.

Returns

Returns an object of CCFXStringSet class that contains a list of the columns in the query. ColdFusion automatically frees the memory that is allocated for the returned string set, after the request is completed.

Example

The following example gets the list of columns, then iterates over the list, writing each column name back to the user:

// Get the list of columns from the query
CCFXStringSet* pColumns = pQuery->GetColumns() ;
int nNumColumns = pColumns->GetCount() ;

// Print the list of columns to the user
pRequest->Write( "Columns in query: " ) ;
for( int i=1; i<=nNumColumns; i++ )
{
pRequest->Write( pColumns->GetString( i ) ) ;
pRequest->Write( " " ) ;
}
CCFXQuery::GetData

**Syntax**

LPCSTR CCFXQuery::GetData(int iRow, int iColumn)

**Description**

Gets a data element from a row and column of a query. Row and column indexes begin with 1. You can determine the number of rows in a query by calling `CCFXQuery::GetRowCount`. You can determine the number of columns in a query by retrieving the list of columns using `CCFXQuery::GetColumns`, and then calling `CCFXStringSet::GetCount` on the returned string set.

**Returns**

Returns the value of the requested data element.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>iRow</td>
<td>Row to retrieve data from (1-based)</td>
</tr>
<tr>
<td>iColumn</td>
<td>Column to retrieve data from (1-based)</td>
</tr>
</tbody>
</table>

**Example**

The following example iterates over the elements of a query and writes the data in the query back to the user in a simple, space-delimited format:

```c
int iRow, iCol;
int nNumCols = pQuery->GetColumns()->GetCount();
int nNumRows = pQuery->GetRowCount();
for ( iRow=1; iRow<=nNumRows; iRow++ )
{
    for ( iCol=1; iCol<=nNumCols; iCol++ )
    {
        pRequest->Write( pQuery->GetData( iRow, iCol ) ) ;
        pRequest->Write( " ");
    }
    pRequest->Write( "<BR>" ) ;
}
```

CCFXQuery::GetName

**Syntax**

LPCSTR CCFXQuery::GetName(void)

**Description**

Returns the name of a query.
Example

The following example retrieves the name of a query and writes it back to the user:

```cpp
CCFXQuery* pQuery = pRequest->GetQuery();
pRequest->Write( "The query name is: " );
pRequest->Write( pQuery->GetName() );
```

CCFXQuery::GetRowCount

**Syntax**

```cpp
int CCFXQuery::GetRowCount(void)
```

**Description**

Returns the number of rows contained in a query.

Example

The following example retrieves the number of rows in a query and writes it back to the user:

```cpp
CCFXQuery* pQuery = pRequest->GetQuery();
char buffOutput[256];
wsprintf( buffOutput, "The number of rows in the query is %ld.\n", pQuery->GetRowCount());
pRequest->Write( buffOutput );
```

CCFXQuery::SetData

**Syntax**

```cpp
void CCFXQuery::SetData(int iRow, int iColumn, LPCSTR lpszData)
```

**Description**

Sets a data element within a row and column of a query. Row and column indexes begin with 1. Before calling `SetData` for a given row, call `CCFXQuery::AddRow` and use the return value as the row index for your call to `SetData`.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>iRow</code></td>
<td>Row of data element to set (1-based)</td>
</tr>
<tr>
<td><code>iColumn</code></td>
<td>Column of data element to set (1-based)</td>
</tr>
<tr>
<td><code>lpszData</code></td>
<td>New value for data element</td>
</tr>
</tbody>
</table>
Example

The following example shows the addition of two rows to a three-column ('City', 'State', and 'Zip') query:

```c++
// First row
int iRow;
iRow = pQuery->AddRow();
pQuery->SetData( iCity, iRow, "Minneapolis" );
pQuery->SetData( iState, iRow, "MN" );
pQuery->SetData( iZip, iRow, "55345" );

// Second row
iRow = pQuery->AddRow();
pQuery->SetData( iCity, iRow, "St. Paul" );
pQuery->SetData( iState, iRow, "MN" );
pQuery->SetData( iZip, iRow, "55105" );
```

CCFXRequest class

Abstract class that represents a request made to a ColdFusion Extension (CFX). An instance of this class is passed to the main function of your extension DLL. The class provides interfaces that can be used by the custom extension for the following actions:

- Reading and writing variables
- Returning output
- Creating and using queries
- Throwing exceptions

Class methods

- `virtual BOOL AttributeExists ( LPCSTR lpszName )`
  - `CCFXRequest::AttributeExists` checks whether the attribute was passed to the tag.

- `virtual LPCSTR GetAttribute ( LPCSTR lpszName )`
  - `CCFXRequest::GetAttribute` gets the value of the passed attribute.

- `virtual CCFXStringSet* GetAttributeList()`
  - `CCFXRequest::GetAttributeList` gets an array of attribute names passed to the tag.

- `virtual CCFXQuery* GetQuery()`
  - `CCFXRequest::GetQuery` gets the query that was passed to the tag.

- `virtual LPCSTR GetSetting( LPCSTR lpszSettingName )`
  - `CCFXRequest::GetSetting` This method is deprecated. It might not work, and might cause an error, in later releases.

- `virtual void Write( LPCSTR lpszOutput )`
  - `CCFXRequest::Write` writes text output back to the user.
virtual void SetVariable( LPCSTR lpszName, LPCSTR lpszValue )

virtual CCFXQuery* AddQuery( LPCSTR lpszName, CCFXStringSet* pColumns )

virtual BOOL Debug()

virtual void WriteDebug( LPCSTR lpszOutput )

virtual CCFXStringSet* CreateStringSet()

virtual void ThrowException( LPCSTR lpszError, LPCSTR lpszDiagnostics )

virtual void ReThrowException( CCFXException* e )

virtual void SetCustomData( LPVOID lpvData )

virtual LPVOID GetCustomData()

CCFXRequest::AddQuery

Syntax

CCFXQuery* CCFXRequest::AddQuery(LPCSTR lpszName, CCFXStringSet* pColumns)

Description

Adds a query to the calling template. The query can be accessed by CFML tags (for example, CFOUTPUT or CFTABLE) within the template. After calling AddQuery, the query is empty (it has 0 rows). To populate the query with data, call the CCFXQuery::AddRow and CCFXQuery::SetData functions.

Returns

Returns a pointer to the query that was added to the template (an object of class CCFXQuery). The memory allocated for the returned query is freed automatically by ColdFusion after the request is completed.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lpszName</td>
<td>Name of query to add to the template (must be unique)</td>
</tr>
<tr>
<td>pColumns</td>
<td>List of column names to be used in the query</td>
</tr>
</tbody>
</table>
Example

The following example adds a query named 'People' to the calling template. The query has two columns ('FirstName' and 'LastName') and two rows:

```cpp
// Create a string set and add the column names to it
CCFXStringSet* pColumns = pRequest->CreateStringSet();
int iFirstName = pColumns->AddString( "FirstName" );
int iLastName = pColumns->AddString( "LastName" );

// Create a query that contains these columns
CCFXQuery* pQuery = pRequest->AddQuery( "People", pColumns );

// Add data to the query
int iRow;
for (iRow = 0; iRow < 2; iRow++)
{
pQuery->SetData( iRow, iFirstName, "John" );
pQuery->SetData( iRow, iLastName, "Smith" );
}
```

CCFXRequest::AttributeExists

**Syntax**

```cpp
BOOL CCFXRequest::AttributeExists(LPCSTR lpszName)
```

**Description**

Checks whether the parameter was passed to the tag. Returns True if the parameter is available; False, otherwise.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lpszName</td>
<td>Name of the parameter to check (case insensitive)</td>
</tr>
</tbody>
</table>

**Example**

The following example checks whether the user passed an attribute named DESTINATION to the tag, and throws an exception if the attribute was not passed:

```cpp
if ( pRequest->AttributeExists("DESTINATION") == FALSE )
{
pRequest->ThrowException(
    "Missing DESTINATION parameter",
    "You must pass a DESTINATION parameter in "
    "order for this tag to work correctly." );
}
```
CCFXRequest::CreateStringSet

Syntax
CCFXStringSet* CCFXRequest::CreateStringSet(void)

Description
Allocates and returns an instance. Always use this function to create string sets, as opposed to directly using the new operator.

Returns
Returns an object of CCFXStringSet class. The memory allocated for the returned string set is freed automatically by ColdFusion after the request is completed.

Example
The following example creates a string set and adds three strings to it:

```c
CCFXStringSet* pColors = pRequest->CreateStringSet() ;
pColors->AddString( "Red" ) ;
pColors->AddString( "Green" ) ;
pColors->AddString( "Blue" ) ;
```

CCFXRequest::Debug

Syntax
BOOL CCFXRequest::Debug(void)

Description
Checks whether the tag contains the Debug attribute. Use this function to determine whether to write debug information for a request. For more information, see CCFXRequest::WriteDebug.

Returns
Returns True if the tag contains the Debug attribute; False, otherwise.

Example
The following example checks whether the Debug attribute is present, and if it is, it writes a brief debug message:

```c
if ( pRequest->Debug() )
{
    pRequest->WriteDebug( "Top secret debug info" ) ;
}
```

CCFXRequest::GetAttribute

Syntax
LPCSTR CCFXRequest::GetAttribute(LPCSTR LpszName)

Description
Retrieves the value of the passed attribute. Returns an empty string if the attribute does not exist. (To test whether an attribute was passed to the tag, use CCFXRequest::AttributeExists.)
Returns

Returns the value of the attribute passed to the tag. If no attribute of that name was passed to the tag, an empty string is returned.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lpszName</td>
<td>Name of the attribute to retrieve (case insensitive)</td>
</tr>
</tbody>
</table>

Example

The following example retrieves an attribute named DESTINATION and writes its value back to the user:

```c
LPCSTR lpszDestination = pRequest->GetAttribute("DESTINATION") ;
pRequest->Write( "The destination is: " ) ;
pRequest->Write( lpszDestination ) ;
```

CCFXRequest::GetAttributeList

Syntax

CCFXStringSet* CCFXRequest::GetAttributeList(void)

Description

Gets an array of attribute names passed to the tag. To get the value of one attribute, use CCFXRequest::GetAttribute.

Returns

Returns an object of class CCFXStringSet class that contains a list of attributes passed to the tag. The memory allocated for the returned string set is freed automatically by ColdFusion after the request is completed.

Example

The following example gets the list of attributes and iterates over the list, writing each attribute and its value back to the user.

```c
LPCSTR lpszName, lpszValue ;
CCFXStringSet* pAttribs = pRequest->GetAttributeList() ;
int nNumAttribs = pAttribs->GetCount() ;
for( int i=1; i<=nNumAttribs; i++ )
{
    lpszName = pAttribs->GetString( i ) ;
    lpszValue = pRequest->GetAttribute( lpszName ) ;
    pRequest->Write( lpszName ) ;
    pRequest->Write( " = " ) ;
    pRequest->Write( lpszValue ) ;
    pRequest->Write( "<BR>" ) ;
}
```
CCFXRequest::GetCustomData

Syntax

LPVOID CCFXRequest::GetCustomData(void)

Description

Gets the custom (tag specific) data for the request. This method is typically used from within subroutines of a tag implementation to extract tag data from a request.

Returns

Returns a pointer to the custom data, or NULL if no custom data has been set during this request using CCFXRequest::SetCustomData.

Example

The following example retrieves a pointer to a request specific data structure of hypothetical type MYTAGDATA:

```c
void DoSomeGruntWork( CCFXRequest* pRequest )
{
    MYTAGDATA* pTagData =
        (MYTAGDATA*)pRequest->GetCustomData() ;

    ... remainder of procedure ... 
}
```

CCFXRequest::GetQuery

Syntax

CCFXQuery* CCFXRequest::GetQuery(void)

Description

Retrieves a query that was passed to a tag. To pass a query to a custom tag, you use the QUERY attribute. This attribute should be set to the name of a query (created using the cfquery tag or another custom tag). The QUERY attribute is optional and should be used only by tags that process an existing data set.

Returns

Returns an object of the CCFXQuery class that represents the query passed to the tag. If no query was passed to the tag, NULL is returned. The memory allocated for the returned query is freed automatically by ColdFusion after the request is completed.
Example

The following example retrieves the query that was passed to the tag. If no query was passed, an exception is thrown:

```cpp
CCFXQuery* pQuery = pRequest->GetQuery() ;
if ( pQuery == NULL )
{
    pRequest->ThrowException(
        "Missing QUERY parameter",
        "You must pass a QUERY parameter in "
        "order for this tag to work correctly." ) ;
}
```

CCFXRequest::ReThrowException

Syntax

```cpp
void CCFXRequest::ReThrowException(CCFXException* e)
```

Description

Re-throws an exception that has been caught within an extension procedure. This function is used to avoid having C++ exceptions that are thrown by DLL extension code propagate back into ColdFusion. Catch ALL C++ exceptions that occur in extension code, and either re-throw them (if they are of the CCFXException class) or create and throw a new exception pointer using CCFXRequest::ThrowException.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>e</td>
<td>A CCFXException that has been caught</td>
</tr>
</tbody>
</table>

Example

The following code demonstrates how to handle exceptions in ColdFusion Extension DLL procedures:

```cpp
try
{
    // Code that could throw an exception...
}
catch( CCFXException* e )
{
    // Do appropriate resource cleanup here...
    // Re-throw the exception
    pRequest->ReThrowException( e ) ;
}
catch( ... )
{
    // Something nasty happened
    pRequest->ThrowException( 
        "Unexpected error occurred in CFX tag", "" ) ;
}
```
CCFXRequest::SetCustomData

Syntax

void CCFXRequest::SetCustomData(LPVOID lpvData)

Description

Sets custom (tag specific) data to carry with the request. Use this function to store request specific
data to pass to procedures within your custom tag implementation.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lpvData</td>
<td>Pointer to custom data</td>
</tr>
</tbody>
</table>

Example

The following example creates a request-specific data structure of hypothetical type
MYTAGDATA and stores a pointer to the structure in the request for future use:

```cpp
void ProcessTagRequest( CCFXRequest* pRequest )
{
    try
    {
        MYTAGDATA tagData ;
        pRequest->SetCustomData( (LPVOID)&tagData ) ;
        ... remainder of procedure ...
    }
}
```

CCFXRequest::SetVariable

Syntax

void CCFXRequest::SetVariable(LPCSTR lpszName, LPCSTR lpszValue)

Description

Sets a variable in the calling template. If the variable name already exists in the template, its value
is replaced. If it does not exist, a variable is created. The values of variables created using
SetVariable can be accessed in the same manner as other template variables (for example,
#MessageSent#).

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lpszName</td>
<td>Name of variable</td>
</tr>
<tr>
<td>lpszValue</td>
<td>Value of variable</td>
</tr>
</tbody>
</table>
Example

The following example sets the value of a variable named ‘MessageSent’ based on the success of an operation performed by the custom tag:

```cpp
BOOL bMessageSent;
...attempt to send the message...
if ( bMessageSent == TRUE )
{
    pRequest->SetVariable( "MessageSent", "Yes" ) ;
}
else
{
    pRequest->SetVariable( "MessageSent", "No" ) ;
}
```

CCFXRequest::ThrowException

Syntax

```cpp
void CCFXRequest::ThrowException(LPCSTR lpszError,
LPCSTR lpszDiagnostics)
```

Description

Throws an exception and ends processing of a request. Call this function when you encounter an error that does not allow you to continue processing the request. This function is almost always combined with the CCFXRequest::ReThrowException to protect against resource leaks in extension code.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lpszError</td>
<td>Short identifier for error</td>
</tr>
<tr>
<td>lpszDiagnostics</td>
<td>Error diagnostic information</td>
</tr>
</tbody>
</table>

Example

The following example throws an exception indicating that an unexpected error occurred while processing a request:

```cpp
char buffError[512] ;
wsprintf( buffError,
    "Unexpected Windows NT error number %ld 
    "occurred while processing request.". GetLastError() ) ;

pRequest->ThrowException( "Error occurred", buffError ) ;
```
**CCFXRequest::Write**

**Syntax**

```cpp
void CCFXRequest::Write(LPCSTR lpszOutput)
```

**Description**

Writes text output back to the user.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lpszOutput</td>
<td>Text to output</td>
</tr>
</tbody>
</table>

**Example**

The following example creates a buffer to hold an output string, fills the buffer with data, and writes the output back to the user:

```cpp
CHAR buffOutput[1024] ;
wsprintf( buffOutput, "The destination is: %s", pRequest->GetAttribute("DESTINATION") );
pRequest->Write( buffOutput ) ;
```

**CCFXRequest::WriteDebug**

**Syntax**

```cpp
void CCFXRequest::WriteDebug(LPCSTR lpszOutput)
```

**Description**

Writes text output into the debug stream. The text is only displayed to the end-user if the tag contains the `Debug` attribute. (For more information, see `CCFXRequest::Debug`.)

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lpszOutput</td>
<td>Text to output</td>
</tr>
</tbody>
</table>

**Example**

The following example checks whether the `Debug` attribute is present; if so, it writes a brief debug message:

```cpp
if ( pRequest->Debug() )
{
    pRequest->WriteDebug( "Top secret debug info" ) ;
}
```
CCFXStringSet class

Abstract class that represents a set of ordered strings. Strings can be added to a set and can be retrieved by a numeric index (index values for strings are 1-based). To create a string set, use CCFXRequest::CreateStringSet.

Class methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>virtual int AddString( LPCSTR lpszString )</td>
<td>CCFXStringSet::AddString adds a string to the end of a list.</td>
</tr>
<tr>
<td>virtual int GetCount()</td>
<td>CCFXStringSet::GetCount gets the number of strings contained in a list.</td>
</tr>
<tr>
<td>virtual LPCSTR GetString( int iIndex )</td>
<td>CCFXStringSet::GetString gets the string located at the passed index.</td>
</tr>
<tr>
<td>virtual int GetIndexForString ( LPCSTR lpszString )</td>
<td>CCFXStringSet::GetIndexForString gets the index for the passed string.</td>
</tr>
</tbody>
</table>

CCFXStringSet::AddString

Syntax

int CCFXStringSet::AddString(LPCSTR lpszString)

Description

Adds a string to the end of the list.

Returns

The index of the string that was added.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lpszString</td>
<td>String to add to the list</td>
</tr>
</tbody>
</table>

Example

The following example demonstrates adding three strings to a string set and saving the indexes of the items that are added:

CCFXStringSet* pSet = pRequest->CreateStringSet();
int iRed = pSet->AddString( "Red" );
int iGreen = pSet->AddString( "Green" );
int iBlue = pSet->AddString( "Blue" );

CCFXStringSet::GetCount

Syntax

int CCFXStringSet::GetCount(void)
Description

Gets the number of strings in a string set. The value can be used with `CCFXStringSet::GetString` to iterate over the strings in the set (recall that the index values for strings in the list begin at 1).

Returns

Returns the number of strings contained in the string set.

Example

The following example demonstrates using `GetCount` with `CCFXStringSet::GetString` to iterate over a string set and write the contents of the list back to the user:

```cpp
int nNumItems = pStringSet->GetCount() ;
for ( int i=1; i<=nNumItems; i++ )
{
    pRequest->Write(pStringSet->GetString(i)) ;
pRequest->Write("<BR>" ) ;
}
```

CCFXStringSet::GetIndexForString

Syntax

```cpp
int CCFXStringSet::GetIndexForString(LPCSTR lpszString)
```

Description

Searches for a passed string. The search is case-insensitive.

Returns

If the string is found, its index within the string set is returned. If it is not found, the constant `CFX_STRING_NOT_FOUND` is returned.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lpszString</td>
<td>String to search for</td>
</tr>
</tbody>
</table>

Example

The following example demonstrates a search for a string and throwing an exception if it is not found:

```cpp
CCFXStringSet* pAttribs = pRequest->GetAttributeList() ;

int iDestination =
    pAttribs->GetIndexForString("DESTINATION") ;
if ( iDestination == CFX_STRING_NOT_FOUND )
{
    pRequest->ThrowException("DESTINATION attribute not found.
   "The DESTINATION attribute is required 
   "by this tag." ) ;
}
```
CCFXStringSet::GetString

Syntax

LPCSTR CCFXStringSet::GetString(int iIndex)

Description

Retrieves the string located at the passed index (index values are 1-based).

Returns

Returns the string located at the passed index.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Index</td>
<td>Index of string to retrieve</td>
</tr>
</tbody>
</table>

Example

The following example demonstrates GetString with CCFXStringSet::GetCount to iterate over a string set and write the contents of a list back to the user:

```c++
int nNumItems = pStringSet->GetCount() ;
for ( int i=1; i<=nNumItems; i++ )
{
    pRequest->Write( pStringSet->GetString( i ) ) ;
    pRequest->Write( "<BR>" ) ;
}
```
This chapter describes the Java interfaces available for building Macromedia ColdFusion MX 7 custom CFXs in Java.

Contents

Class libraries overview ................................................. 1075
Custom tag interface ....................................................... 1076
Query interface ............................................................ 1077
Request interface .......................................................... 1082
Response interface ......................................................... 1087
Debugging classes reference ............................................. 1090

Class libraries overview

The following Java interfaces are available for building ColdFusion custom CFXs in Java:

<table>
<thead>
<tr>
<th>Interface</th>
<th>Methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>Custom tag interface</td>
<td>processRequest</td>
</tr>
<tr>
<td>Query interface</td>
<td>addRow, getColumnIndex, getColumns, getData, getName, getRowCount, setData</td>
</tr>
</tbody>
</table>
Custom tag interface

public abstract interface CustomTag

Interface for implementing custom tags.

Classes that implement this interface can be specified in the CLASS attribute of the Java CFX tag. For example, in a class MyCustomTag, which implements this interface, the following CFML code calls the MyCustomTag.processRequest method:

```cfml
<CFX_MyCustomTag>
```

Other attributes can be passed to the Java CFX tag. Their values are available using the Request object passed to the processRequest method.

Methods

<table>
<thead>
<tr>
<th>Returns</th>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>void</td>
<td>processRequest(Request request, Response response)</td>
<td>Processes a request originating from the CFX_mycustomtag tag</td>
</tr>
</tbody>
</table>

processRequest

Description

Processes a request originating from the Java CFX tag.

Category

Custom tag interface

Syntax

```java
public void processRequest(Request request, Response response)
```

Throws

Exception If an unexpected error occurs while processing the request.
Query interface

public abstract interface Query

Interface to a query used or created by a custom tag. A query contains tabular data organized by named columns and rows.

Methods

<table>
<thead>
<tr>
<th>Returns</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>int</td>
<td>addRow()</td>
<td>Adds a row to the query</td>
</tr>
<tr>
<td>int</td>
<td>getColumnIndex(String name)</td>
<td>Gets the index of a column given its name</td>
</tr>
<tr>
<td>String[]</td>
<td>getColumns()</td>
<td>Gets a list of the column names in a query</td>
</tr>
<tr>
<td>String</td>
<td>getData(int iRow, int iCol)</td>
<td>Gets a data element from a row and column of a query</td>
</tr>
<tr>
<td>String</td>
<td>getName()</td>
<td>Gets the name of a query</td>
</tr>
<tr>
<td>int</td>
<td>getRowCount()</td>
<td>Gets the number of rows in a query</td>
</tr>
<tr>
<td>void</td>
<td>setData(int iRow, int iCol, String data)</td>
<td>Sets a data element in a row and column of a query</td>
</tr>
</tbody>
</table>

addRow

Description

Adds a row to a query. Call this method to append a row to a query.

Returns the index of the row that was appended to the query.

Category

Query interface

Syntax

public int addRow()

See also

setData, getData

Example

The following example demonstrates the addition of two rows to a query that has three columns, City, State, and Zip:

// Define column indexes
int iCity = 1, iState = 2, iZip = 3;

// First row
int iRow = query.addRow();
query.setData(iRow, iCity, "Minneapolis");
query.setData(iRow, iState, "MN");
query.setData(iRow, iZip, "55345");
// Second row
iRow = query.addRow();
query.setData(iRow, iCity, "St. Paul");
query.setData(iRow, iState, "MN");
query.setData(iRow, iZip, "55105");

getColumnIndex

Description
Returns the index of the column, or 0 if no such column exists.

Category
Query interface

Syntax
public int getColumnIndex(String name)

See also
getColumns, getData

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Name of column to get index of (lookup is case-insensitive)</td>
</tr>
</tbody>
</table>

Example

The following example retrieves the index of the EMAIL column and uses it to output a list of the addresses contained in the column:

// Get the index of the EMAIL column
int iEMail = query.getColumnIndex( "EMAIL" ) ;

// Iterate over the query and output list of addresses
int nRows = query.getRowCount() ;
for( int iRow = 1; iRow <= nRows; iRow++ )
{
    response.write( query.getData( iRow, iEMail ) + "<BR>" ) ;
}
**getColumns**

**Description**

Returns an array of strings containing the names of the columns in the query.

**Category**

Query interface

**Syntax**

```java
public String[] getColumns()
```

**Example**

The following example retrieves the array of columns, then iterates over the list, writing each column name back to the user:

```java
// Get the list of columns from the query
String[] columns = query.getColumns();
int nNumColumns = columns.length;

// Print the list of columns to the user
response.write( "Columns in query: " );
for( int i=0; i<nNumColumns; i++ )
{
    response.write( columns[i] + " " );
}
```

**getData**

**Description**

Retrieves a data element from a row and column of a query. Row and column indexes begin with 1. You can find the number of rows in a query by calling `getRowCount`. You can find the number of columns in a query by calling `getColumns`.

Returns the value of the requested data element.

**Category**

Query interface

**Syntax**

```java
public String getData(int iRow, int iCol)
```

**Throws**

`IndexOutOfBoundsException` If an invalid index is passed to the method.

**See also**

`setData, addRow`
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>iRow</td>
<td>Row to retrieve data from (1-based)</td>
</tr>
<tr>
<td>iCol</td>
<td>Column to retrieve data from (1-based)</td>
</tr>
</tbody>
</table>

Example

The following example iterates over the rows of a query and writes the data back to the user in a simple, space-delimited format:

```java
int iRow, iCol;
int nNumCols = query.getColumns().length;
int nNumRows = query.getRowCount();
for ( iRow = 1; iRow <= nNumRows; iRow++ )
{
    for ( iCol = 1; iCol <= nNumCols; iCol++ )
    {
        response.write( query.getData( iRow, iCol ) + " ");
    }
    response.write( "<BR>" );
}
```

getName

Description

Returns the name of a query.

Category

Query interface

Syntax

```java
public String getName()
```

Example

The following example retrieves the name of a query and writes it back to the user:

```java
Query query = request.getQuery();
response.write( "The query name is: " + query.getName() );
```

getRowCount

Description

Retrieves the number of rows in a query.

Returns the number of rows contained in a query.

Category

Query interface

Syntax

```java
public int getRowCount()
```
Example

The following example retrieves the number of rows in a query and writes it back to the user:

```java
Query query = request.getQuery();
int rows = query.getRowCount();
response.write( "The number of rows in the query is " + Integer.ToString(rows) );
```

**setData**

**Description**

Sets a data element in a row and column of a query. Row and column indexes begin with 1. Before calling `setData` for a given row, call `addRow` and use the return value as the row index for your call to `setData`.

**Category**

Query interface

**Syntax**

```java
public void setData(int iRow, int iCol, String data)
```

**Throws**

`IndexOutOfBoundsException` If an invalid index is passed to the method.

**See also**

gedata, addRow

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>iRow</td>
<td>Row of data element to set (1-based)</td>
</tr>
<tr>
<td>iCol</td>
<td>Column of data element to set (1-based)</td>
</tr>
<tr>
<td>data</td>
<td>New value for data element</td>
</tr>
</tbody>
</table>

**Example**

The following example demonstrates the addition of two rows to a query that has three columns, *City*, *State*, and *Zip*:

```java
// Define column indexes
int iCity = 1, iState = 2, iZip = 3;

// First row
int iRow = query.addRow();
query.setData( iRow, iCity, "Minneapolis" );
query.setData( iRow, iState, "MN" );
query.setData( iRow, iZip, "55345" );

// Second row
iRow = query.addRow();
```
query.setData(iRow, iCity, "St. Paul")
query.setData(iRow, iState, "MN")
query.setData(iRow, iZip, "55105")

Request interface

public abstract interface Request

Interface to a request made to a CustomTag. The interface includes methods for retrieving attributes passed to the tag (including queries) and reading global tag settings.

Methods

<table>
<thead>
<tr>
<th>Returns</th>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>boolean</td>
<td>attributeExists(String name)</td>
<td>Checks whether the attribute was passed to this tag.</td>
</tr>
<tr>
<td>boolean</td>
<td>debug()</td>
<td>Checks whether the tag contains the debug attribute.</td>
</tr>
<tr>
<td>String</td>
<td>getAttribute(String name)</td>
<td>Retrieves the value of the passed attribute.</td>
</tr>
<tr>
<td>String[]</td>
<td>.getAttributeList()</td>
<td>Retrieves a list of attributes passed to the tag.</td>
</tr>
<tr>
<td>int</td>
<td>getIntAttribute(String name)</td>
<td>Retrieves the value of the passed attribute as an integer.</td>
</tr>
<tr>
<td>int</td>
<td>getIntAttribute(String name, int def)</td>
<td>Retrieves the value of the passed attribute as an integer (returns default if the attribute does not exist or is not a valid number).</td>
</tr>
<tr>
<td>Query</td>
<td>getQuery()</td>
<td>Retrieves the query that was passed to this tag.</td>
</tr>
</tbody>
</table>

attributeExists

Description

Checks whether the attribute was passed to this tag.

Returns True if the attribute is available; otherwise returns False.

Category

Request interface

Syntax

public boolean attributeExists(String name)

See also

ggetAttribute, getAttributeList

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Name of the attribute to check (case-insensitive)</td>
</tr>
</tbody>
</table>
Example

The following example checks whether the user passed an attribute named DESTINATION to the tag; if not, it throws an exception:

```java
if ( ! request.attributeExists("DESTINATION") )
{
    throw new Exception(
        "Missing DESTINATION parameter",
        "You must pass a DESTINATION parameter in order for this tag to work correctly." ) ;
}
```

description

Checks whether the tag contains the debug attribute. Use this method to determine whether to write debug information for this request. For more information, see writeDebug.

Returns True if the tag contains the debug attribute; False, otherwise.

category

Request interface

Syntax

```java
public boolean debug()
```

See also

writeDebug

Example

The following example checks whether the debug attribute is present, and if so, it writes a brief debug message:

```java
if ( request.debug() )
{
    response.writeDebug( "debug info" ) ;
}
```

getAttribute

Description

Retrieves the value of a passed attribute. Returns an empty string if the attribute does not exist (use attributeExists to test whether an attribute was passed to the tag). Use getAttribute(String,String) to return a default value rather than an empty string.

Returns the value of the attribute passed to the tag. If no attribute of that name was passed to the tag, an empty string is returned.

category

Request interface
Syntax

public String getAttribute(String name)

See also

attributeExists, getAttributeList, getIntAttribute, getAttribute

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>The attribute to retrieve (case-insensitive)</td>
</tr>
</tbody>
</table>

Example

The following example retrieves an attribute named DESTINATION and writes its value back to the user:

```java
String strDestination = request.getAttribute("DESTINATION") ;
response.write( "The destination is: " + strDestination ) ;
```

getAttributeList

Description

Retrieves a list of attributes passed to the tag. To retrieve the value of one attribute, use the getAttribute method.

Returns an array of strings containing the names of the attributes passed to the tag.

Category

Request interface

Syntax

public String[] getAttributeList()

See also

attributeExists, getAttributeList

Example

The following example retrieves the list of attributes, then iterates over the list, writing each attribute and its value back to the user:

```java
String[] attribs = request.getAttributeList() ;
int nNumAttribs = attribs.length ;
for( int i = 0; i < nNumAttribs; i++ )
{
    String strName = attribs[i] ;
    String strValue = request.getAttribute( strName ) ;
    response.write( strName + "=" + strValue + "<BR>" ) ;
}
```
**getIntAttribute**

**Description**
Retrieves the value of the passed attribute as an integer. Returns -1 if the attribute does not exist. Use `attributeExists` to test whether an attribute was passed to the tag. Use `getIntAttribute(String, int)` to return a default value rather than throwing an exception or returning -1.

Returns the value of the attribute passed to the tag. If no attribute of that name was passed to the tag, -1 is returned.

**Category**
Request interface

**Syntax**
```java
public int getIntAttribute(String name)
```

**Throws**
- `NumberFormatException` If the attribute is not a valid number.

**See also**
`attributeExists, getAttributeList, getIntAttribute`

**Parameters**
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>The attribute to retrieve (case-insensitive)</td>
</tr>
</tbody>
</table>

**Example**
The following example retrieves an attribute named `PORT` and writes its value back to the user:
```java
int nPort = request.getIntAttribute("PORT") ;
if ( nPort != -1 )
    response.write( "The port is: " + String.valueOf(nPort) ) ;
```

**getQuery**

**Description**
Retrieves the query that was passed to this tag.

To pass a query to a custom tag, you use the query attribute. It should be set to the name of a query (created using the `cfquery` tag). The query attribute is optional and should be used only by tags that process an existing dataset.

Returns the Query that was passed to the tag. If no query was passed, returns null.

**Category**
Request interface
Syntax

```java
public Query getQuery()
```

Example

The following example retrieves a query that was passed to a tag. If no query was passed, an exception is thrown:

```java
Query query = request.getQuery();
if (query == null)
{
    throw new Exception("Missing QUERY parameter. " +
            "You must pass a QUERY parameter in 
            "order for this tag to work correctly."");
}
```

**getSetting**

Description

Retrieves the value of a global custom tag setting. Custom tag settings are stored in the CustomTags section of the ColdFusion Registry key.

Returns the value of the custom tag setting. If no setting of that name exists, an empty string is returned.

Category

Request interface

Syntax

```java
public String getSetting(String name)
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>The name of the setting to retrieve (case-insensitive)</td>
</tr>
</tbody>
</table>

Usage

All custom tags implemented in Java share a registry key for storing settings. To avoid name conflicts, preface the names of settings with the name of your custom tag class. For example, the code below retrieves the value of a setting named `VerifyAddress` for a custom tag class named `MyCustomTag`:

```java
String strVerify = request.getSetting("MyCustomTag.VerifyAddress");
if (Boolean.valueOf(strVerify))
{
    // Do address verification...
}
```
Response interface

public abstract interface Response

Interface to response generated from a custom tag. This interface includes methods for writing output, generating queries, and setting variables in the calling page.

Methods

<table>
<thead>
<tr>
<th>Returns</th>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Query</td>
<td>addQuery(String name, String[] columns)</td>
<td>Adds a query to the calling template.</td>
</tr>
<tr>
<td>void</td>
<td>setVariable(String name, String value)</td>
<td>Sets a variable in the calling template.</td>
</tr>
<tr>
<td>void</td>
<td>write(String output)</td>
<td>Outputs text back to the user.</td>
</tr>
<tr>
<td>void</td>
<td>writeDebug(String output)</td>
<td>Writes text output into the debug stream.</td>
</tr>
</tbody>
</table>

addQuery

Description

Adds a query to the calling template. The query can be accessed by CFML tags in the template. After calling addQuery, the query is empty (it has 0 rows). To populate the query with data, call the Query methods addRow and setData.

Returns the Query that was added to the template.

Category

Response interface

Syntax

public Query addQuery(String name, String[] columns)

Throws

IllegalArgumentException  If the name parameter is not a valid CFML variable name.

See also

addRow, setData

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>The name of the query to add to the template</td>
</tr>
<tr>
<td>columns</td>
<td>The column names to use in the query</td>
</tr>
</tbody>
</table>

Example

The following example adds a query named People to the calling template. The query has two columns (FirstName and LastName) and two rows:

```java
// Create string array with column names (also track columns indexes)
String[] columns = { "FirstName", "LastName" } ;
```
int iFirstName = 1, iLastName = 2;

// Create a query which contains these columns
Query query = response.addQuery( "People", columns );

// Add data to the query
int iRow = query.addRow();
query.setData(iRow, iFirstName, "John");
query.setData(iRow, iLastName, "Smith");
iRow = query.addRow();
query.setData(iRow, iFirstName, "Jane");
query.setData(iRow, iLastName, "Doe");

setVariable

Description
Sets a variable in the calling template. If the variable name specified exists in the template, its value is replaced. If it does not exist, a new variable is created.

Category
Response interface

Syntax
public void setVariable(String name, String value)

Throws
IllegalArgumentException If the name parameter is not a valid CFML variable name.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>The name of the variable to set</td>
</tr>
<tr>
<td>value</td>
<td>The value to set the variable to</td>
</tr>
</tbody>
</table>

Example
For example, this code sets the value of a variable named MessageSent based on the success of an operation performed by the custom tag:

boolean bMessageSent;

...attempt to send the message...

if ( bMessageSent == true )
{
    response.setVariable( "MessageSent", "Yes" );
}
else
{
    response.setVariable( "MessageSent", "No" );
}
### write

**Description**

Outputs text back to the user.

**Category**

*Response interface*

**Syntax**

```
public void write(String output)
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>output</td>
<td>Text to output</td>
</tr>
</tbody>
</table>

**Example**

The following example outputs the value of the `DESTINATION` attribute:

```java
response.write( "DESTINATION = " +
    request.getAttribute("DESTINATION") ) ;
```

### writeDebug

**Description**

Writes text output into the debug stream. This text is displayed to the end-user only if the tag contains the `debug` attribute (check for this attribute using the `Request.debug` method).

**Category**

*Response interface*

**Syntax**

```
public void writeDebug(String output)
```

**See also**

*debug*

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>output</td>
<td>The text to output</td>
</tr>
</tbody>
</table>

**Example**

The following example checks whether the `debug` attribute is present; if so, it writes a brief debug message:

```java
if ( request.debug() )
{
    response.writeDebug( "debug info" ) ;
}
```
Debugging classes reference

The constructors and methods supported by the DebugRequest, DebugResponse, and DebugQuery classes are as follows. These classes also support the other methods of the Request, Response, and Query interfaces, respectively.

**DebugRequest**

```java
// initialize a debug request with attributes
public DebugRequest( Hashtable attributes ) ;

// initialize a debug request with attributes and a query
public DebugRequest( Hashtable attributes, Query query ) ;

// initialize a debug request with attributes, a query, and settings
public DebugRequest( Hashtable attributes, Query query,
                     Hashtable settings ) ;
```

**DebugResponse**

```java
// initialize a debug response
public DebugResponse() ;

// print the results of processing
public void printResults() ;
```

**DebugQuery**

```java
// initialize a query with name and columns
public DebugQuery( String name, String[] columns )
                     throws IllegalArgumentException ;

// initialize a query with name, columns, and data
public DebugQuery( String name, String[] columns, String[][] data )
                     throws IllegalArgumentException ;
```
This chapter provides information about JavaScript objects and functions used to WDDX in a Macromedia ColdFusion MX 7 application.

Contents

JavaScript object overview .......................................................... 1091
WddxSerializer object ................................................................. 1092
WddxRecordset object ................................................................. 1095

JavaScript object overview

These are the JavaScript objects and functions:

<table>
<thead>
<tr>
<th>Class</th>
<th>Functions</th>
</tr>
</thead>
</table>
| WddxSerializer object | serialize  
                        | serializeVariable  
                        | serializeValue  
                        | write  |
| WddxRecordset object | addColumn  
                        | addRows  
                        | getField  
                        | getRowCount  
                        | setField  
                        | wddxSerialize  |

WDDX JavaScript objects are defined in the wddx.js file; this file is installed in the CFIDE/scripts directory.

To use these objects, you must put a JavaScript tag before the code that refers to the objects; for example:

```html
<script type="text/javascript" src="/CFIDE/scripts/wddx.js"></script>
```
The `WddxSerializer` object includes functions that serialize any JavaScript data structure. For more information on using this object, see “Using WDDX” in Chapter 35, “Using XML and WDDX” in *ColdFusion MX Developer’s Guide*.

### Functions

The only function that developers typically call is `serialize`.

<table>
<thead>
<tr>
<th>Function syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>object.serialize(rootobj)</code></td>
<td>Creates a WDDX packet for a passed <code>WddxRecordset</code> instance.</td>
</tr>
<tr>
<td><code>object.serializeVariable(name, obj)</code></td>
<td>Serializes a property of a structure. If an object is not a string, number, array, Boolean, or a date, <code>WddxSerializer</code> treats it as a structure.</td>
</tr>
<tr>
<td><code>object.serializeValue(obj)</code></td>
<td>Recursively serializes eligible data in a passed instance.</td>
</tr>
<tr>
<td><code>object.write(str)</code></td>
<td>Appends data to the serialized data stream.</td>
</tr>
</tbody>
</table>

### serialize

**Description**

Creates a WDDX packet for a passed `WddxRecordset` instance.

**Syntax**

`object.serialize( rootobj )`

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>object</code></td>
<td>Instance name of the <code>WddxSerializer</code> object</td>
</tr>
<tr>
<td><code>rootobj</code></td>
<td>JavaScript data structure to serialize</td>
</tr>
</tbody>
</table>

**Return value**

Returns a serialized WDDX packet as a string if the function succeeds, or a null value if an error occurs.

**Usage**

Call this function to serialize the data in a `WddxRecordset` instance.

**Example**

This example shows a JavaScript function that you can call to serialize a `WddxRecordset` instance. It copies serialized data to a form field for display:

```javascript
function serializeData(data, formField) {
    wddxSerializer = new WddxSerializer();
    wddxPacket = wddxSerializer.serialize(data);
    formField.value = wddxPacket;
}
```
if (wddxPacket != null)
{
    formField.value = wddxPacket;
}
else
{
    alert("Couldn’t serialize data");
}

**serializeVariable**

**Description**

Serializes a property of a structure. If an object is not a string, number, array, Boolean, or date, WddxSerializer treats it as a structure.

**Syntax**

```javascript
object.serializeVariable(name, obj)
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>Instance name of a WddxSerializer object</td>
</tr>
<tr>
<td>name</td>
<td>Property to serialize</td>
</tr>
<tr>
<td>obj</td>
<td>Instance name of the value to serialize</td>
</tr>
</tbody>
</table>

**Return value**

Returns a Boolean True if serialization was successful; False, otherwise.

This is an internal function; you do not typically call it.

**Example**

This example is from the WddxSerializer `serializeValue` function:

```javascript
... // Some generic object; treat it as a structure
this.write("<struct>");
for (prop in obj)
{
    bSuccess = this.serializeVariable(prop, obj[prop]);
    if (! bSuccess)
    {
        break;
    }
} this.write("</struct>");
...```
serializeValue

Description
Recursively serializes eligible data in a passed instance. Eligible data includes:

• String
• Number
• Boolean
• Date
• Array
• Recordset
• Any JavaScript object

This function serializes null values as empty strings.

Syntax
object.serializeValue( obj )

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>Instance name of the WddxSerializer object</td>
</tr>
<tr>
<td>obj</td>
<td>Instance name of the WddxRecordset object to serialize</td>
</tr>
</tbody>
</table>

Return value

Returns a Boolean True if obj was serialized successfully; False, otherwise.

Usage

This is an internal function; you do not typically call it.

Example

This example is from the WddxSerializer serialize function:

```javascript
... 
this.wddxPacket = "";
this.write("<wddxPacket version='1.0'><header><data>");
bSuccess = this.serializeValue(rootObj);
this.write("</data></wddxPacket>");
if (bSuccess)
{
    return this.wddxPacket;
}
else
{
    return null;
}
... 
```
**write**

**Description**
Appends data to a serialized data stream.

**Syntax**

```javascript
object.write( str )
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>Instance name of the WddxSerializer object</td>
</tr>
<tr>
<td>str</td>
<td>String to be copied to the serialized data stream</td>
</tr>
</tbody>
</table>

**Return value**

Returns an updated serialized data stream as a String.

**Usage**

This is an internal function; you do not typically call it.

**Example**

This example is from the WddxSerializer `serializeValue` function:

```javascript
... 
else if (typeof(obj) == "number")
{
    // Number value
    this.write("<number>" + obj + "/number>");
} else if (typeof(obj) == "boolean")
{
    // Boolean value
    this.write("<boolean value='" + obj + "/value'>");
}
...
```

**WddxRecordset object**

Includes functions that you call as needed when constructing a WDDX record set. For more information on using this object, see “Using WDDX” in Chapter 35, “Using XML and WDDX” in *ColdFusion MX Developer's Guide*.

**Functions**

<table>
<thead>
<tr>
<th>Function syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>object.addColumn(name)</code></td>
<td>Adds a column to all rows in a WddxRecordset instance.</td>
</tr>
<tr>
<td><code>object.addRows(n)</code></td>
<td>Adds rows to all columns in a WddxRecordset instance.</td>
</tr>
<tr>
<td><code>object.dump(escapeStrings)</code></td>
<td>Displays WddxRecordset object data.</td>
</tr>
</tbody>
</table>
Returns

HTML table of the WddxRecordset object data.

Usage

Convenient for debugging and testing record sets. The boolean parameter escapeStrings determines whether &lt;&gt;&amp; characters in string values are escaped as &lt;&gt;&amp; in HTML.

Example

<!--- Create a simple query --->
<cfquery name = "q" datasource ="cfdocexamples">
SELECT Message_Id, Thread_id, Username, Posted
FROM messages
</cfquery>
<!--- Load the wddx.js file, which includes the dump function --->
<script type="text/javascript" src="/CFIDE/scripts/wddx.js"></script>
<script>
// Use WDDX to move from CFML data to JS
<cfwddx action="cfml2js" input="#q#" topLevelVariable="qj"> // Dump the record set
document.write(qj.dump(true));
</script>

addColumn

Description

Adds a column to all rows in a WddxRecordset instance.

Syntax

object.addColumn( name )

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>Instance name of the WddxRecordset object</td>
</tr>
<tr>
<td>name</td>
<td>Name of the column to add</td>
</tr>
</tbody>
</table>

Return value

None.
Usage

Adds a column to every row of the WDDX record set. Initially the new column’s values are set to NULL.

Example

This example calls the addColumn function:

```java
// Create a new record set
rs = new WddxRecordset();

// Add a new column
rs.addColumn("NewColumn");

// Extend the record set by 3 rows
rs.addRows(3);

// Set an element in the first row
// newValue is a previously defined variable
rs.setField(0, "NewColumn", newValue);
```

addRows

Description

Adds rows to all columns in a WddxRecordset instance.

Syntax

```java
object.addRows( n )
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>Instance name of the WddxRecordset object</td>
</tr>
<tr>
<td>n</td>
<td>Integer; number of rows to add</td>
</tr>
</tbody>
</table>

Return value

None.

Usage

This function adds the specified number of rows to every column of a WDDX record set. Initially, the row/column values are set to NULL.

Example

This example calls the addRows function:

```java
// Create a new record set
rs = new WddxRecordset();

// Add a new column
rs.addColumn("NewColumn");
```
// Extend the record set by 3 rows
rs.addRows(3);

// Set an element in the first row
// newValue is a previously defined variable
rs.setField(0, "NewColumn", newValue);

**getField**

**Description**

Returns the element in the specified row/column position.

**Syntax**

```
object.getField(row, col)
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>Instance name of the WddxRecordset object</td>
</tr>
<tr>
<td>row</td>
<td>Integer; zero-based row number of the value to return</td>
</tr>
<tr>
<td>col</td>
<td>Integer or string; column of the value to be returned.</td>
</tr>
</tbody>
</table>

**Return value**

Returns the value in the specified row/column position.

**Usage**

Call this function to access a value in a WDDX record set.

**Example**

This example calls the `getField` function (the variable `r` is a reference to a WddxRecordset instance):

```javascript
for (row = 0; row < nRows; ++row)
{
    o += "<tr>":
    for (i = 0; i < colNames.length; ++i)
    {
        o += "<td>" + r.getField(row, colNames[i]) + "</td>":
    }
    o += "</tr>":
}
```

**getRowCount**

**Description**

Indicates the number of rows in a WddxRecordset instance.

**Syntax**

```
object.getRowCount()
```
**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>Instance name of a WddxRecordset object</td>
</tr>
</tbody>
</table>

**Return value**

Integer. Returns the number of rows in the WddxRecordset instance.

**Usage**

Call this function before a looping construct to determine the number of rows in a record set.

**Example**

This example calls the `getRowCount` function:

```javascript
function dumpWddxRecordset(r) {
    // Get row count
    nRows = r.getRowCount();
    ...
    for (row = 0; row < nRows; ++row)
        ...
}
```

**setField**

**Description**

Sets the element in the specified row/column position.

**Syntax**

`object.setField(row, col, value)`

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>Instance name of a WddxRecordset object</td>
</tr>
<tr>
<td>row</td>
<td>Integer; row that contains the element to set</td>
</tr>
<tr>
<td>col</td>
<td>Integer or string; the column containing the element to set</td>
</tr>
<tr>
<td>value</td>
<td>Value to set</td>
</tr>
</tbody>
</table>

**Return value**

None.

**Usage**

Call this function to set a value in a WddxRecordset instance.

**Example**

This example calls the `setField` function:

```javascript
// Create a new recordset
rs = new WddxRecordset();
```
// Add a new column
rs.addColumn("NewColumn");

// Extend the record set by 3 rows
rs.addRows(3);

// Set an element in the first row
// newValue is a previously defined variable
rs.setField(0, "NewColumn", newValue);

### wddxSerialize

**Description**

Serializes a record set.

**Syntax**

```
object.wddxSerialize( serializer )
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>Instance name of the WddxRecordset object</td>
</tr>
<tr>
<td>serializer</td>
<td>WddxSerializer instance</td>
</tr>
</tbody>
</table>

**Return value**

Returns a Boolean `True` if serialization was successful; `False`, otherwise.

**Usage**

This is an internal function; you do not typically call it.

**Example**

This example is from the WddxSerializer `serializeValue` function:

```javascript
... else if (typeof(obj) == "object")
{
    if (obj == null)
    {
        // Null values become empty strings
        this.write("<string></string>");
    }
    else if (typeof(obj.wddxSerialize) == "function")
    {
        // Object knows how to serialize itself
        bSuccess = obj.wddxSerialize(this);
    }
...```

---

1100 Chapter 9: WDDX JavaScript Objects
CHAPTER 10
ColdFusion ActionScript Functions

This chapter explains the syntax and usage of the two Macromedia ColdFusion MX 7 server-side ActionScript functions, CF.query and CF.http.

Contents

CF.query .................................................. 1102
CF.http .................................................. 1104
**CF.query**

**Description**
Performs queries against ColdFusion data sources.

**Return value**
Returns a RecordSet object.

**Syntax**
```cfm
CF.query
({
  datasource:"data source name",
  sql:"SQL stmts",
  username:"username",
  password:"password",
  maxrows:number,
  timeout:milliseconds
})
```

**Arguments**

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Req/Opt</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>datasource</td>
<td>Required</td>
<td>Name of the data source from which the query retrieves data.</td>
</tr>
<tr>
<td>sql</td>
<td>Required</td>
<td>SQL statement.</td>
</tr>
<tr>
<td>username</td>
<td>Optional</td>
<td>Username. Overrides the username specified in the data source setup.</td>
</tr>
<tr>
<td>password</td>
<td>Optional</td>
<td>Password. Overrides the password specified in the data source setup.</td>
</tr>
<tr>
<td>maxrows</td>
<td>Optional</td>
<td>Maximum number of rows to return in the record set.</td>
</tr>
<tr>
<td>timeout</td>
<td>Optional</td>
<td>Maximum number of seconds for the query to execute before returning an error indicating that the query has timed out. Can only be used in named arguments.</td>
</tr>
</tbody>
</table>

**Usage**
You can code the **CF.query** function using named or positional arguments. You can invoke all supported arguments using the named argument style, as follows:
```cfm
CF.query({datasource:"datasource", sql:"sql stmt", username:"username", password:"password", maxrows:"maxrows", timeout:"timeout"});
```

**Note:** The named argument style uses curly braces {} to surround the function arguments.

Positional argument style, which is a shorthand coding style, does not support all arguments. Use the following syntax to code the **CF.query** function using positional arguments:
```cfm
CF.query(datasource, sql);
CF.query(datasource, sql, maxrows);
CF.query(datasource, sql, username, password);
CF.query(datasource, sql, username, password, maxrows);
```

**Note:** Do not use curly braces {} with positional arguments.
You can manipulate the record set returned by the `CF.query` function using methods in the RecordSet ActionScript class. The following are some of the methods available in the RecordSet class:

- `RecordSet.getColumnnames`
- `RecordSet.getLength`
- `RecordSet.getItemAt`
- `RecordSet.getItemId`
- `RecordSet.sortItemsBy`
- `RecordSet.getNumberAvailable`
- `RecordSet.filter`
- `RecordSet.sort`

For more information on using server-side ActionScript, see Chapter 34, “Using Server-Side ActionScript” in ColdFusion MX Developer’s Guide. For more detailed information about the RecordSet ActionScript class, see Using Flash Remoting.

Example

// Define a function to do a basic query
// Note use of positional arguments
function basicQuery()
{
  result = CF.query("myquery", "cust_data", "SELECT * from tblParks");
  return result;
}

// Example function declaration using named arguments
function basicQuery()
{
  result = CF.query({datasource:"cust_data", sql: "SELECT * from tblParks");
  return result;
}

// Example of the CF.query function using maxrows argument
function basicQueryWithMaxRows()
{
  result = CF.query("cust_data", "SELECT * from tblParks", 25);
  return result;
}

// Example of the CF.query function with username and password
function basicQueryWithUser()
{
  result = CF.query("cust_data", "SELECT * from tblParks", "wsburroughs", "migraine1");
  return result;
}
**CF.http**

**Description**

Executes HTTP POST and GET operations on files. (POST operations upload MIME file types to a server, or post cookie, formfield, URL, file, or CGI variables directly to a server.)

**Return value**

Returns an object containing properties that you reference to access data.

**Syntax**

```actionscript
CF.http(
    method: "get or post",
    url: "URL",
    username: "username",
    password: "password",
    resolveurl: "yes or no",
    params: arrayvar,
    path: "path",
    file: "filename"
)
```

**Arguments**

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Req/Opt</th>
<th>Description</th>
</tr>
</thead>
</table>
| method    | Required | One of two arguments:  
  - get: downloads a text or binary file or creates a query from the contents of a text file.  
  - post: sends information to the server page or CGI program for processing. Requires the params argument. |
| url       | Required | The absolute URL of the host name or IP address of the server on which the file resides. The URL must include the protocol (http or https) and host name. |
| username  | Optional | When required by a server, a username. |
| password  | Optional | When required by a server, a password. |
You can write the `CF.http` function using named arguments or positional arguments. You can invoke all supported arguments using the named argument style, as follows:

```javascript
```

**Note:** The named argument style uses curly braces `{}` to surround the function arguments.

Positional arguments let you use a shorthand coding style. However, not all arguments are supported for the positional argument style. Use the following syntax to code the `CF.http` function using positional arguments:

```javascript
CF.http(url);
CF.http(method, url);
```

### Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Req/Opt</th>
<th>Description</th>
</tr>
</thead>
</table>
| resolveurl | Optional| For GET and POST methods.  
  - Yes or No. Default is No.  
  For GET and POST operations, if Yes, the page reference that is returned into the Filecontent property has its internal URLs fully resolved, including port number, so that links remain intact. The following HTML tags, which can contain links, are resolved:  
  - `img src`  
  - `a href`  
  - `form action`  
  - `applet code`  
  - `script src`  
  - `embed src`  
  - `embed pluginspace`  
  - `body background`  
  - `frame src`  
  - `bsound src`  
  - `object data`  
  - `object classid`  
  - `object codebase`  
  - `object usemap`  
| params     | Optional| HTTP parameters passed as an array of objects. Supports the following parameter types:  
  - `name`  
  - `type`  
  - `value`  
  `CF.http` params are passed as an array of objects. The `params` argument is required for POST operations.  
| path       | Optional| The path to the directory in which to store files. When using the `path` argument, the `file` argument is required.  
| file       | Optional| Name of the file that is accessed. For GET operations, defaults to the name specified in the `url` argument. Enter path information in the `path` argument. This argument is required if you are using the `path` argument.  

### Usage

You can write the `CF.http` function using named arguments or positional arguments. You can invoke all supported arguments using the named argument style, as follows:
CF.http(method, url, username, password);
CF.http(method, url, params, username, password);

**Note:** Do not use curly braces {} with positional arguments.

The following parameters can only be passed as an array of objects in the `params` argument in the `CF.http` function:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>The variable name for data that is passed</td>
</tr>
<tr>
<td>type</td>
<td>The transaction type:</td>
</tr>
<tr>
<td></td>
<td>• URL</td>
</tr>
<tr>
<td></td>
<td>• FormField</td>
</tr>
<tr>
<td></td>
<td>• Cookie</td>
</tr>
<tr>
<td></td>
<td>• CGI</td>
</tr>
<tr>
<td></td>
<td>• File</td>
</tr>
<tr>
<td>value</td>
<td>Value of URL, FormField, Cookie, File, or CGI variables that are passed</td>
</tr>
</tbody>
</table>

The `CF.http` function returns data as a set of object properties, as described in the following table:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Text</td>
<td>A Boolean value that indicates whether the specified URL location contains text data.</td>
</tr>
<tr>
<td>Charset</td>
<td>The charset used by the document specified in the URL. HTTP servers normally provide this information, or the charset is specified in the charset parameter of the Content-Type header field of the HTTP protocol. For example, the following HTTP header announces that the character encoding is EUC-JP: Content-Type: text/html; charset=EUC-JP</td>
</tr>
<tr>
<td>Header</td>
<td>Raw response header. For example, macromedia.com returns the following header: HTTP/1.1 200 OK Date: Mon, 04 Mar 2002 17:27:44 GMT Server: Apache/1.3.22 (Unix) mod_perl/1.26 Set-Cookie: MM_cookie=207.22.48.162.4731015262864476; path=/; expires=Wed, 03-Mar-04 17:27:44 GMT; domain=.macromedia.com Connection: close Content-Type: text/html</td>
</tr>
<tr>
<td>Filecontent</td>
<td>File contents, for text and MIME files.</td>
</tr>
<tr>
<td>Mimetype</td>
<td>MIME type. Examples of MIME types include text/html, image/png, image/gif, video/mpeg, text/css, and audio/basic.</td>
</tr>
</tbody>
</table>
You access these attributes using the `get` function:

```actionscript
function basicGet()
{
    url = "http://localhost:8100/";

    // Invoke with just the url. This is an HTTP GET.
    result = CF.http(url);
    return result.get("Filecontent");
}
```

**Note:** For more information on using server-side ActionScript, see Chapter 34, "Server-Side ActionScript" in the ColdFusion MX Developer’s Guide.

**Example**

The following examples show a number of the ways to use the `CF.http` function:

```actionscript
function postWithNamedArgs()
{
    // Set up the array of Post parameters.
    params = new Array();
    params[1] = {name:"arg1", type:"FormField", value:"value1"};
    params[2] = {name:"arg2", type:"URL", value:"value2"};
    params[3] = {name:"arg3", type:"CGI", value:"value3"};

    url = "http://localhost:8100/";
    path = application.getContext("/").getRealPath("/");
    file = "foo.txt";


    if (result)
    {
        return result.get("Statuscode");
    }
    return null;
}
```

Example of a basic HTTP GET operation

// Shows that HTTP GET is the default
function basicGet()
url = "http://localhost:8100/";

// Invoke with just the url. This is an HTTP GET.
result = CF.http(url);
return result.get("Filecontent");

// Example showing simple array created to pass params arguments
function postWithParams()
{
    // Set up the array of Post parameters. These are just like cfftparam tags.
    params = new Array();
    params[1] = {name:arg2, type:"URL", value:value2};

    url = "http://localhost:8100/";

    // Invoke with the method, url, and params
    result = CF.http("post", url, params);
    return result.get("Filecontent");
}

// Example with username and params arguments
function postWithParamsAndUser()
{
    // Set up the array of Post parameters. These are just like cfftparam tags.
    params = new Array();
    params[1] = {name:arg2, type:"URL", value:value2};

    url = "http://localhost:8100/";

    // Invoke with the method, url, params, username, and password
    result = CF.http("post", url, params, "karl", "salsa");
    return result.get("Filecontent");
}