CFML Reference
## CONTENTS

### INTRODUCTION
- About Macromedia ColdFusion MX documentation ........................... 17
- Documentation set .................................................................. 17
- Viewing online documentation .............................................. 18

### CHAPTER 1: Reserved Words and Variables
- Reserved words in forms .................................................. 20
- Reserved words in queries ............................................... 20
- Scope-specific built-in variables ....................................... 22
  - Variable scope ................................................................ 22
  - Caller scope ................................................................ 22
  - Client variables .......................................................... 22
  - Server variables ......................................................... 22
  - Application and session variables ................................. 23
  - Custom tag variables .................................................. 24
  - Request variable ........................................................ 24
  - Form variable .............................................................. 24
- ColdFusion tag-specific variables ...................................... 24
  - ColdFusion query variables ........................................ 25
  - CFCATCH variables .................................................... 25
  - CFDIRECTORY variables ............................................. 25
  - CFERROR variables .................................................... 25
  - CFILE ACTION=Upload variables ............................... 26
  - CFHTTP error variables .............................................. 26
  - CFFTP ReturnValue variable ........................................ 26
  - CFFTP query object columns ...................................... 27
  - CFHTTP variables ....................................................... 27
  - CFLDAP variables ....................................................... 27
  - CFPOP variables ........................................................ 27
  - CFQUERY and CFSTOREDPROC variables .................... 28
  - CFREGISTRY variables ............................................... 28
  - CFSEARCH variables .................................................. 28
- Standard CGI variables .................................................... 28
  - Request .................................................................. 28
  - Server .................................................................. 29
  - Client ................................................................. 29
CHAPTER 2: ColdFusion Tags .............................................. 33

Tags by function .................................................... 39
   Application framework tags .................................. 39
   Database manipulation tags ................................ 39
   Data output tags ............................................. 39
   Debugging tags ............................................... 39
   Exception handling tags ................................... 40
   Extensibility tags ........................................... 40
   File management tags ...................................... 40
   Flow-control tags ........................................... 40
   Forms tags .................................................. 40
   Internet Protocol tags ..................................... 41
   Page processing tags ...................................... 41
   Variable manipulation tags ................................ 41
   Other tags .................................................. 41

Tag changes since ColdFusion 5 ................................. 42
   New tags, attributes, and values ............................ 42
   Deprecated tags, attributes, and values .................. 44
   Obsolete tags, attributes, and values ..................... 46

cfabort ...................................................... 47
cfapplet ..................................................... 49
cfapplication .............................................. 51
cfargument .................................................. 54
cfassociate .................................................. 56
cfauthenticate .............................................. 57
cfbreak ...................................................... 58
cfcache ...................................................... 59
cfcase ....................................................... 62
cfcatch ...................................................... 64
cfchart ....................................................... 70
cfchartdata .................................................. 75
cfchartseries ................................................ 76
cfcollection .................................................. 78
cfcomponent ................................................ 80
cfcontent ................................................... 85
cfcookie ..................................................... 91
cfdefaultcase ............................................. 94
cfdirectory ................................................... 95
cfdump ....................................................... 98
cfelse ......................................................... 100
cfelseif .................................................... 101
cferror ....................................................... 102
cexecute ................................................................. 106
cexit ................................................................. 108
cfile ................................................................. 110
cfile action = "append" ........................................ 113
cfile action = "copy" ............................................ 115
cfile action = "delete" .......................................... 116
cfile action = "move" ............................................ 117
cfile action = "read" ............................................ 119
cfile action = "readBinary" .................................... 121
cfile action = "rename" ......................................... 122
cfile action = "upload" ......................................... 124
cfile action = "write" .......................................... 127
cflush ............................................................... 130
cform ............................................................... 132
cfttp ............................................................... 137
cfttp: Opening and closing FTP server connections .......... 138
cfttp: Connection: File and directory operations ........ 141
cfttp action = "listDir" ......................................... 145
cffunction .......................................................... 146
cfgraph ............................................................. 149
cfgrid ............................................................... 151
cfgriddata .......................................................... 152
   How data is returned from cfgrid ........................... 156
   Using the href attribute ...................................... 158
cfgridcolumn ...................................................... 161
cfgridrow .......................................................... 166
cfgridupdate ...................................................... 169
cfheader ............................................................. 171
cfhtmlhead .......................................................... 173
cfhttp ............................................................... 174
cfhttpparam ........................................................ 183
cfif ................................................................. 187
cfimpersonate ...................................................... 189
cfimport ............................................................ 190
cfimportcolumn .................................................... 192
cfindex ............................................................... 193
cfinput .............................................................. 201
cfinherit ............................................................ 205
cfinvoke ............................................................. 208
cfinvokeargument .................................................. 213
cfdap ................................................................. 215
cflocation ........................................................... 220
cflock ............................................................... 221
cflog ................................................................. 226
cflogon ............................................................... 228
cflogin ............................................................... 230
cfloginuser .......................................................... 230
cflogout ............................................................. 231
cfloop ............................................................... 232
cfloop: index loop .................................................. 233
cfloop: conditional loop ......................................... 235
<table>
<thead>
<tr>
<th>Function Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>cffreeitem</td>
<td>349</td>
</tr>
<tr>
<td>cftry</td>
<td>353</td>
</tr>
<tr>
<td>cfupdate</td>
<td>355</td>
</tr>
<tr>
<td>cfwddx</td>
<td>358</td>
</tr>
<tr>
<td>cxml</td>
<td>361</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>CHAPTER 3:</strong> ColdFusion Functions</td>
<td>363</td>
</tr>
<tr>
<td>Functions by category</td>
<td></td>
</tr>
<tr>
<td>Array functions</td>
<td>367</td>
</tr>
<tr>
<td>Authentication functions</td>
<td>367</td>
</tr>
<tr>
<td>Conversion functions</td>
<td>367</td>
</tr>
<tr>
<td>Date and time functions</td>
<td>368</td>
</tr>
<tr>
<td>Decision functions</td>
<td>368</td>
</tr>
<tr>
<td>Display and formatting functions</td>
<td>368</td>
</tr>
<tr>
<td>Dynamic evaluation functions</td>
<td>369</td>
</tr>
<tr>
<td>Extensibility functions</td>
<td>369</td>
</tr>
<tr>
<td>Full-text search functions</td>
<td>369</td>
</tr>
<tr>
<td>International functions</td>
<td>369</td>
</tr>
<tr>
<td>List functions</td>
<td>369</td>
</tr>
<tr>
<td>Mathematical functions</td>
<td>370</td>
</tr>
<tr>
<td>Other functions</td>
<td>370</td>
</tr>
<tr>
<td>Query functions</td>
<td>370</td>
</tr>
<tr>
<td>String functions</td>
<td>370</td>
</tr>
<tr>
<td>Structure functions</td>
<td>371</td>
</tr>
<tr>
<td>System functions</td>
<td>371</td>
</tr>
<tr>
<td>XML functions</td>
<td>371</td>
</tr>
<tr>
<td>Function changes since ColdFusion 5</td>
<td>372</td>
</tr>
<tr>
<td>New functions, parameters, and values</td>
<td>372</td>
</tr>
<tr>
<td>Deprecated functions, parameters, and values</td>
<td>373</td>
</tr>
<tr>
<td>Obsolete functions, parameters, and values</td>
<td>373</td>
</tr>
<tr>
<td>Abs</td>
<td>374</td>
</tr>
<tr>
<td>ACos</td>
<td>375</td>
</tr>
<tr>
<td>ArrayAppend</td>
<td>376</td>
</tr>
<tr>
<td>ArrayAvg</td>
<td>377</td>
</tr>
<tr>
<td>ArrayClear</td>
<td>379</td>
</tr>
<tr>
<td>ArrayDeleteAt</td>
<td>380</td>
</tr>
<tr>
<td>ArrayInsertAt</td>
<td>381</td>
</tr>
<tr>
<td>ArrayIsEmpty</td>
<td>382</td>
</tr>
<tr>
<td>ArrayLen</td>
<td>383</td>
</tr>
<tr>
<td>ArrayMax</td>
<td>384</td>
</tr>
<tr>
<td>ArrayMin</td>
<td>385</td>
</tr>
<tr>
<td>ArrayNew</td>
<td>386</td>
</tr>
<tr>
<td>ArrayPrepend</td>
<td>387</td>
</tr>
<tr>
<td>ArrayResize</td>
<td>388</td>
</tr>
<tr>
<td>ArraySet</td>
<td>389</td>
</tr>
<tr>
<td>ArraySort</td>
<td>390</td>
</tr>
<tr>
<td>ArraySum</td>
<td>392</td>
</tr>
<tr>
<td>ArraySwap</td>
<td>393</td>
</tr>
<tr>
<td>ArrayToList</td>
<td>394</td>
</tr>
</tbody>
</table>
Asc ............................................................... 395
ASin ............................................................... 396
Atn ............................................................... 398
AuthenticatedContext ........................................ 399
AuthenticatedUser ........................................... 400
BitAnd ........................................................... 401
BitMaskClear .................................................. 402
BitMaskRead .................................................... 403
BitMaskSet ....................................................... 404
BitNot ............................................................. 405
BitOr ............................................................... 406
BitSHLN .......................................................... 407
BitSHRN .......................................................... 408
BitXor ............................................................. 409
Ceiling ............................................................. 410
Chr ................................................................. 411
CJustify ........................................................... 412
Compare .......................................................... 413
CompareNoCase ................................................ 415
Cos ................................................................. 417
CreateDate ........................................................ 419
CreateDateTime .................................................. 421
CreateObject ..................................................... 423
  CreateObject object types ................................ 423
  CreateObject: COM object ................................ 424
  CreateObject: component object ......................... 425
  CreateObject: CORBA object ............................. 426
  CreateObject: Java or EJB object ....................... 428
  CreateObject: web service object ....................... 429
CreateODBCDate ................................................ 430
CreateODBCDateTime ........................................... 432
CreateODBCTime ............................................... 434
CreateTime ...................................................... 435
CreateTimeSpan ............................................... 436
CreateUUID ...................................................... 438
DateAdd .......................................................... 439
DateCompare .................................................... 441
DateConvert ..................................................... 443
DateDiff .......................................................... 445
DateFormat ....................................................... 448
DatePart .......................................................... 450
Day ............................................................... 452
DayOfWeek ......................................................... 453
DayOfWeekAsString ............................................. 454
DayOfYear ......................................................... 455
DaysInMonth ..................................................... 456
DaysInYear ....................................................... 457
DE ................................................................. 458
DecimalFormat .................................................... 460
DecrementValue .................................................. 461
<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Decrypt</td>
<td>462</td>
</tr>
<tr>
<td>DeleteClientVariable</td>
<td>463</td>
</tr>
<tr>
<td>DirectoryExists</td>
<td>464</td>
</tr>
<tr>
<td>DollarFormat</td>
<td>465</td>
</tr>
<tr>
<td>Duplicate</td>
<td>466</td>
</tr>
<tr>
<td>Encrypt</td>
<td>467</td>
</tr>
<tr>
<td>Evaluate</td>
<td>468</td>
</tr>
<tr>
<td>Exp</td>
<td>469</td>
</tr>
<tr>
<td>ExpandPath</td>
<td>470</td>
</tr>
<tr>
<td>FileExists</td>
<td>472</td>
</tr>
<tr>
<td>Find</td>
<td>473</td>
</tr>
<tr>
<td>FindNoCase</td>
<td>474</td>
</tr>
<tr>
<td>FindOneOf</td>
<td>475</td>
</tr>
<tr>
<td>FirstDayOfMonth</td>
<td>476</td>
</tr>
<tr>
<td>Fix</td>
<td>477</td>
</tr>
<tr>
<td>FormatBaseN</td>
<td>478</td>
</tr>
<tr>
<td>GetAuthUser</td>
<td>479</td>
</tr>
<tr>
<td>GetBaseTagData</td>
<td>480</td>
</tr>
<tr>
<td>GetBaseTagList</td>
<td>481</td>
</tr>
<tr>
<td>GetBaseTemplatePath</td>
<td>482</td>
</tr>
<tr>
<td>GetClientVariablesList</td>
<td>483</td>
</tr>
<tr>
<td>GetCurrentTemplatePath</td>
<td>484</td>
</tr>
<tr>
<td>GetDirectoryFromPath</td>
<td>485</td>
</tr>
<tr>
<td>GetEncoding</td>
<td>487</td>
</tr>
<tr>
<td>GetException</td>
<td>488</td>
</tr>
<tr>
<td>GetFileFromPath</td>
<td>489</td>
</tr>
<tr>
<td>GetFunctionList</td>
<td>490</td>
</tr>
<tr>
<td>GetHttpRequestData</td>
<td>491</td>
</tr>
<tr>
<td>GetHttpTimeString</td>
<td>493</td>
</tr>
<tr>
<td>GetK2ServerDocCount</td>
<td>494</td>
</tr>
<tr>
<td>GetK2ServerDocCountLimit</td>
<td>495</td>
</tr>
<tr>
<td>GetLocale</td>
<td>496</td>
</tr>
<tr>
<td>GetMetaData</td>
<td>497</td>
</tr>
<tr>
<td>GetMetricData</td>
<td>499</td>
</tr>
<tr>
<td>GetPageContext</td>
<td>501</td>
</tr>
<tr>
<td>GetProfileSections</td>
<td>502</td>
</tr>
<tr>
<td>GetProfileString</td>
<td>503</td>
</tr>
<tr>
<td>GetTempDirectory</td>
<td>505</td>
</tr>
<tr>
<td>GetTempFile</td>
<td>506</td>
</tr>
<tr>
<td>GetTemplatePath</td>
<td>507</td>
</tr>
<tr>
<td>GetTickCount</td>
<td>508</td>
</tr>
<tr>
<td>GetTimeZoneInfo</td>
<td>509</td>
</tr>
<tr>
<td>GetToken</td>
<td>510</td>
</tr>
<tr>
<td>Hash</td>
<td>513</td>
</tr>
<tr>
<td>Hour</td>
<td>514</td>
</tr>
<tr>
<td>HTMLCodeFormat</td>
<td>515</td>
</tr>
<tr>
<td>HTMLEditFormat</td>
<td>516</td>
</tr>
<tr>
<td>If</td>
<td>517</td>
</tr>
<tr>
<td>IncrementValue</td>
<td>520</td>
</tr>
<tr>
<td>InputBaseN</td>
<td>521</td>
</tr>
</tbody>
</table>
ListValueCount .................................................. 590
ListValueCountNoCase ............................................ 592
LJustify ......................................................... 594
Log ........................................................... 595
Log10 .......................................................... 596
LSCurrencyFormat ................................................ 597
LDatFormats ..................................................... 600
LSEuroCurrencyFormat ........................................... 602
LSIsCurrency .................................................... 605
LSIsDate ......................................................... 607
LSIsNumeric ..................................................... 609
LNumberFormat .................................................. 610
LSParseCurrency .................................................. 613
LSParseDateTime .................................................. 615
LSParseEuroCurrency ............................................. 617
LSParseNumber ................................................... 619
LSTimeFormat .................................................... 621
LTrim ............................................................ 623
Max ........................................................... 624
Mid ............................................................ 625
Min ............................................................ 626
Minute ........................................................ 627
Month .......................................................... 628
MonthAsString ................................................... 629
Now ............................................................ 630
NumberFormat .................................................... 631
ParagraphFormat .................................................. 634
ParameterExists ................................................... 635
ParseDateTime ..................................................... 636
Pi ............................................................. 638
PreserveSingleQuotes .............................................. 639
Quarter ........................................................ 641
QueryAddColumn ................................................... 642
QueryAddRow ....................................................... 644
QueryNew ......................................................... 645
QuerySetCell ...................................................... 646
QuotedValueList ................................................... 647
Rand ........................................................... 648
Randomize ......................................................... 649
RandRange ......................................................... 650
REFind ........................................................ 651
REFindNoCase ...................................................... 654
ReleaseComObject ................................................ 657
RemoveChars ....................................................... 658
RepeatString ....................................................... 659
Replace ........................................................ 660
ReplaceList ......................................................... 661
ReplaceNoCase ..................................................... 663
REReplace ........................................................ 664
REReplaceNoCase ................................................... 666
Reverse .......................................................... 668
Right ............................................................ 669
RJustify .......................................................... 670
Round ............................................................. 671
RTrim ............................................................. 672
Second ........................................................... 673
SetEncoding ..................................................... 674
SetLocale ........................................................ 676
SetProfileString .............................................. 678
SetVariable ..................................................... 680
Sgn ................................................................. 682
Sin ................................................................. 683
SpanExcluding .................................................. 685
SpanIncluding ................................................... 686
Sqr ................................................................. 687
StripCR ........................................................... 688
StructAppend ................................................... 689
StructClear ..................................................... 691
StructCopy ...................................................... 693
StructCount ..................................................... 697
StructDelete .................................................... 698
StructFind ....................................................... 700
StructFindKey .................................................. 701
StructFindValue ............................................... 702
StructGet ....................................................... 703
StructInsert ..................................................... 705
StructIsEmpty ................................................... 707
StructKeyArray ................................................ 708
StructKeyExists ............................................... 710
StructKeyList .................................................. 711
StructNew ....................................................... 713
StructSort ....................................................... 714
StructUpdate ................................................... 716
Tan ................................................................. 717
TimeFormat ...................................................... 719
ToBase64 .......................................................... 721
ToBinary .......................................................... 723
ToString .......................................................... 725
Trim ............................................................... 727
UCase ............................................................. 728
URLEncode ........................................................ 729
URLEncodedFormat ........................................... 731
URLRequestFormat ............................................ 733
Val ................................................................. 734
ValueList ........................................................ 735
Week .............................................................. 736
Wrap .............................................................. 737
WriteOutput ..................................................... 738
XmlNodePos ..................................................... 739
XmlElementNew ................................................ 740
CHAPTER 5: ColdFusion Java CFX Reference ................................................. 769
  Overview class libraries ........................................................................... 770
  CustomTag interface .................................................................................. 770
    Methods .................................................................................................. 770
    processRequest ...................................................................................... 770
  Query interface ........................................................................................ 772
    Methods .................................................................................................. 772
    addRow ................................................................................................. 772
    getColumnIndex ..................................................................................... 773
    getColumns ......................................................................................... 773
    getData ................................................................................................. 774
    getName ............................................................................................... 775
    getRowCount .......................................................................................... 775
    setData ................................................................................................. 775
  Request interface ...................................................................................... 777
    Methods .................................................................................................. 777
    attributeExists ..................................................................................... 777
    debug ..................................................................................................... 778
    getAttribute .......................................................................................... 778
    getAttributeList .................................................................................... 779
    getIntAttribute ..................................................................................... 779
    getQuery .............................................................................................. 780
    getSetting ............................................................................................ 780
  Response interface .................................................................................... 782
    Methods .................................................................................................. 782
    addQuery .............................................................................................. 782
    setVariable ........................................................................................... 783
    write ...................................................................................................... 784
    writeDebug ............................................................................................ 784
  Debugging classes reference ...................................................................... 785
    DebugRequest ....................................................................................... 785
    DebugResponse ...................................................................................... 785
    DebugQuery .......................................................................................... 785

CHAPTER 6: WDDX JavaScript Objects ......................................................... 787
  JavaScript object overview ........................................................................ 788
  WddxSerializer object ............................................................................. 789
    Functions ............................................................................................... 789
    serialize ............................................................................................... 789
    serializeVariable .................................................................................. 790
    serializeValue ...................................................................................... 790
    write ...................................................................................................... 791
  WddxRecordset object .............................................................................. 793
    Functions ............................................................................................... 793
    addColumn ............................................................................................ 793
    addRows ............................................................................................... 794
    getField ............................................................................................... 795
    getRowCount ........................................................................................ 796
setField ............................................................... 796
wddxSerialize ..................................................... 797

CHAPTER 7: ColdFusion ActionScript Functions .......................... 799
CFquery ............................................................... 800
CFhttp ............................................................... 801
INTRODUCTION

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

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>Book</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,</td>
</tr>
<tr>
<td></td>
<td>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</td>
</tr>
<tr>
<td></td>
<td>connecting to your data sources and configuring security for your</td>
</tr>
<tr>
<td></td>
<td>applications. Part II describes Verity search tools and utilities that you</td>
</tr>
<tr>
<td></td>
<td>can use for configuring the Verity K2 Server search engine, as well as</td>
</tr>
<tr>
<td></td>
<td>creating, managing, and troubleshooting Verity collections.</td>
</tr>
<tr>
<td>Developing ColdFusion MX Applications</td>
<td>Describes how to develop your dynamic web applications, including</td>
</tr>
<tr>
<td></td>
<td>retrieving and updating your data, using structures, and forms.</td>
</tr>
<tr>
<td>Getting Started Building ColdFusion MX</td>
<td>Contains an overview of ColdFusion features and application development</td>
</tr>
<tr>
<td>Applications</td>
<td>procedures. Includes a tutorial that guides you through the process of</td>
</tr>
<tr>
<td></td>
<td>developing an example ColdFusion application.</td>
</tr>
<tr>
<td>CFML Reference</td>
<td>Provides descriptions, syntax, usage, and code examples for all ColdFusion</td>
</tr>
<tr>
<td></td>
<td>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</td>
</tr>
<tr>
<td></td>
<td>variables.</td>
</tr>
</tbody>
</table>

17
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 ColdFusion reserved words and lists scope variables.

Contents
Reserved words ......................................................... 20
Scope-specific built-in variables .................................... 22
ColdFusion tag-specific variables .................................... 24
CGI environment variables ........................................... 29
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

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

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. See your DBMS documentation for detailed list.

<table>
<thead>
<tr>
<th>ABSOLUTE</th>
<th>ACTION</th>
<th>ADD</th>
<th>ALL</th>
<th>ALLOCATE</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALTER</td>
<td>AND</td>
<td>ANY</td>
<td>ARE</td>
<td>AS</td>
</tr>
<tr>
<td>ASC</td>
<td>ASSERTION</td>
<td>AT</td>
<td>AUTHORIZATION</td>
<td>AVG</td>
</tr>
<tr>
<td>BEGIN</td>
<td>BETWEEN</td>
<td>BIT</td>
<td>BIT_LENGTH</td>
<td>BOTH</td>
</tr>
<tr>
<td>BY</td>
<td>CASCADE</td>
<td>CASE</td>
<td>CAST</td>
<td></td>
</tr>
<tr>
<td>CATALOG</td>
<td>CHAR</td>
<td>CHARACTER</td>
<td>CHARACTER_LENGTH</td>
<td></td>
</tr>
<tr>
<td>CHECK</td>
<td>CLOSE</td>
<td>COALESCE</td>
<td>COLLATE</td>
<td>COLLATION</td>
</tr>
<tr>
<td>COLUMN</td>
<td>COMMIT</td>
<td>CONNECT</td>
<td>CONNECTION</td>
<td>CONSTRAINT</td>
</tr>
<tr>
<td>--------------</td>
<td>--------</td>
<td>---------</td>
<td>------------</td>
<td>------------</td>
</tr>
<tr>
<td>CONSTRAINTS</td>
<td>CONTINUE</td>
<td>CONVERT</td>
<td>CORRESPONDING</td>
<td>COUNT</td>
</tr>
<tr>
<td>CREATE</td>
<td>CROSS</td>
<td>CURRENT</td>
<td>CURRENT_DATE</td>
<td>CURRENT_TIME</td>
</tr>
<tr>
<td>CURRENT_TIME</td>
<td>STAMP</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>
<td>DEFAULT</td>
</tr>
<tr>
<td>DEFERRABLE</td>
<td>DEFERRED</td>
<td>DELETE</td>
<td>DESC</td>
<td>DESCRIBE</td>
</tr>
<tr>
<td>DESCRIPTOR</td>
<td>DIAGNOSTICS</td>
<td>DISCONNECT</td>
<td>DISTINCT</td>
<td>DOMAIN</td>
</tr>
<tr>
<td>DOUBLE</td>
<td>DROP</td>
<td>ELSE</td>
<td>END</td>
<td>END-EXEC</td>
</tr>
<tr>
<td>ESCAPE</td>
<td>EXCEPT</td>
<td>EXCEPTION</td>
<td>EXEC</td>
<td>EXECUTE</td>
</tr>
<tr>
<td>EXISTS</td>
<td>EXTERNAL</td>
<td>EXTRACT</td>
<td>FALSE</td>
<td>FETCH</td>
</tr>
<tr>
<td>FIRST</td>
<td>FLOAT</td>
<td>FOR</td>
<td>FOREIGN</td>
<td>FOUND</td>
</tr>
<tr>
<td>FROM</td>
<td>FULL</td>
<td>GET</td>
<td>GLOBAL</td>
<td>GO</td>
</tr>
<tr>
<td>GOTO</td>
<td>GRANT</td>
<td>GROUP</td>
<td>HAVING</td>
<td>HOUR</td>
</tr>
<tr>
<td>IDENTITY</td>
<td>IMMEDIATE</td>
<td>IN</td>
<td>INDICATOR</td>
<td>INITIALLY</td>
</tr>
<tr>
<td>INNER</td>
<td>INPUT</td>
<td>INSENSITIVE</td>
<td>INSERT</td>
<td>INT</td>
</tr>
<tr>
<td>INTEGER</td>
<td>INTERSECT</td>
<td>INTERVAL</td>
<td>INTO</td>
<td>IS</td>
</tr>
<tr>
<td>ISOLATION</td>
<td>JOIN</td>
<td>KEY</td>
<td>LANGUAGE</td>
<td>LAST</td>
</tr>
<tr>
<td>LEADING</td>
<td>LEFT</td>
<td>LEVEL</td>
<td>LIKE</td>
<td>LOCAL</td>
</tr>
<tr>
<td>LOWER</td>
<td>MATCH</td>
<td>MAX</td>
<td>MIN</td>
<td>MINUTE</td>
</tr>
<tr>
<td>MODULE</td>
<td>MONTH</td>
<td>NAMES</td>
<td>NATIONAL</td>
<td>NATURAL</td>
</tr>
<tr>
<td>NCHAR</td>
<td>NEXT</td>
<td>NO</td>
<td>NOT</td>
<td>NULL</td>
</tr>
<tr>
<td>NULLIF</td>
<td>NUMERIC</td>
<td>OCTET_LENGTH</td>
<td>OF</td>
<td>ON</td>
</tr>
<tr>
<td>ONLY</td>
<td>OPEN</td>
<td>OPTION</td>
<td>OR</td>
<td>ORDER</td>
</tr>
<tr>
<td>OUTER</td>
<td>OUTPUT</td>
<td>OVERLAPS</td>
<td>PAD</td>
<td>PARTIAL</td>
</tr>
<tr>
<td>POSITION</td>
<td>PRECISION</td>
<td>PREPARE</td>
<td>PRESERVE</td>
<td>PRIMARY</td>
</tr>
<tr>
<td>PRIOR</td>
<td>PRIVILEGES</td>
<td>PROCEDURE</td>
<td>PUBLIC</td>
<td>READ</td>
</tr>
<tr>
<td>REAL</td>
<td>REFERENCES</td>
<td>RELATIVE</td>
<td>RESTRICT</td>
<td>REVOKE</td>
</tr>
<tr>
<td>RIGHT</td>
<td>ROLLBACK</td>
<td>ROWS</td>
<td>SCHEMA</td>
<td>SCROLL</td>
</tr>
<tr>
<td>SECOND</td>
<td>SECTION</td>
<td>SELECT</td>
<td>SESSION</td>
<td>SESSION_USER</td>
</tr>
<tr>
<td>SET</td>
<td>SIZE</td>
<td>SMALLINT</td>
<td>SOME</td>
<td>SPACE</td>
</tr>
<tr>
<td>SQL</td>
<td>SQLCODE</td>
<td>SQLERROR</td>
<td>SQLSTATE</td>
<td>SUBSTRING</td>
</tr>
<tr>
<td>SUM</td>
<td>SYSTEM_USER</td>
<td>TABLE</td>
<td>TEMPORARY</td>
<td>THEN</td>
</tr>
</tbody>
</table>

Reserved words  21
<table>
<thead>
<tr>
<th>Time</th>
<th>Timestamp</th>
<th>Timezone_hour</th>
<th>Timezone_minute</th>
<th>To</th>
</tr>
</thead>
<tbody>
<tr>
<td>Trailing</td>
<td>Transaction</td>
<td>Translate</td>
<td>Translation</td>
<td>Trim</td>
</tr>
<tr>
<td>True</td>
<td>Union</td>
<td>Unique</td>
<td>Unknown</td>
<td>Update</td>
</tr>
<tr>
<td>Upper</td>
<td>Usage</td>
<td>User</td>
<td>Using</td>
<td>Value</td>
</tr>
<tr>
<td>Values</td>
<td>Varchar</td>
<td>Varying</td>
<td>View</td>
<td>When</td>
</tr>
<tr>
<td>Whenever</td>
<td>Where</td>
<td>With</td>
<td>Work</td>
<td>Write</td>
</tr>
<tr>
<td>Year</td>
<td>Zone</td>
<td></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 Developing ColdFusion MX Applications.

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 221, and Chapter 15, "Using Persistent Data and Locking," in Developing ColdFusion MX Applications.

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 read-only:

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

Server variables

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

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

Application and session variables

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

- `Application.myvariable`
- `Session.myvariable`
To ensure that modifications to shared data occur in the intended sequence, use the `cflock` tag. For more information, see `cflock` on page 221.

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:

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

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

```html
cfoutput>Cfoutput>Caller.variable_name</cfoutput>
```

### Request variable

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

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

```html
<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:

```html
<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:

```html
Form.FieldNames
```

### ColdFusion tag-specific variables

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

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

- `cfcatch`
- `cfdirectory`
- `cferror`
- `cffile`
- `cfftp`
coldFusion tag-specific variables 25

cfhttp
cfindex
cfindap
cfmail
cfpop
cfquery
cfregistry
cfsearch
cfstoredproc

ColdFusion query variables

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

\begin{itemize}
\item \texttt{queryname.CurrentRow}
\item \texttt{queryname.RecordCount}
\item \texttt{queryname.ColumnList}
\end{itemize}

CFCATCH variables

Within a \texttt{cfcatch} block, the active exception properties can be accessed as the following variables:

\begin{itemize}
\item \texttt{CFCATCH.Type}
\item \texttt{CFCATCH.Message}
\item \texttt{CFCATCH.Detail}
\item \texttt{CFCATCH.ErrNumber}
\item \texttt{CFCATCH.NativeErrorCode}
\item \texttt{CFCATCH.SQLState}
\item \texttt{CFCATCH.LockName}
\item \texttt{CFCATCH.LockOperation}
\item \texttt{CFCATCH.MissingFileName}
\item \texttt{CFCATCH.TagContext}
\item \texttt{CFCATCH.ErrorCode}
\item \texttt{CFCATCH.ExtendedInfo}
\end{itemize}

CFDIRECTORY variables

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

\begin{itemize}
\item \texttt{queryname.Name}
\item \texttt{queryname.Size}
\item \texttt{queryname.Type}
\item \texttt{queryname.DateLastModified}
\item \texttt{queryname.Attributes}
\item \texttt{queryname.Mode}
\end{itemize}

CFERROR variables

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

\begin{itemize}
\item \texttt{Error.Diagnostics}
\item \texttt{Error.MailTo}
\item \texttt{Error.DateTime}
\item \texttt{Error.Browser}
\item \texttt{Error.GeneratedContent}
\item \texttt{Error.RemoteAddress}
\end{itemize}
Error.HTTPReferer
Error.Template
Error.QueryString

The following error variables are available if type="validation".
Error.ValidationHeader
Error.InvalidFields
Error.ValidationFooter

Any cfcatch variable that applies to exception type can be accessed within the Error scope, as follows:
Error.Type
Error.Message
Error.Detail
Error.ErrNumber
Error.NativeErrorCode
Error.SQLState
Error.LockName
Error.LockOperation
Error.MissingFileName
Error.TagContext
Error.ErrorCode
Error.ExtendedInfo

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

CFFILE ACTION=Upload variables

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

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

CFFTP error variables

When you use the cfftp stoponerror attribute, these variables are populated:
CFFTP.Succeeded
CFFTP.ErrorCode
CFFTP.ErrorText

**CFFTP ReturnValue variable**

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

- GetCurrentDir
- GetCurrentURL
- ExistsDir
- ExistsFile
- Exists

**CFFTP query object columns**

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

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

**CFHTTP variables**

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

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

**CFLDAP variables**

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

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

**CFPOP variables**

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

- queryname.Date
- queryname.From
- queryname.Body
- queryname.Header
- queryname.MessageNumber
- queryname.ReplyTo
- queryname.Subject
CFQUERY and CFSTOREDPROC variables

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

CFQUERY.ExecutionTime

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

queryname.CurrentRow
queryname.RecordCount
queryname.ColumnList

The cfstoredproc tag returns the following variables:

CFSTOREDPROC.ExecutionTime
CFSTOREDPROC.StatusCode

CFREGISTRY variables

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

queryname.Entry
queryname.Type
queryname.Value

CFSEARCH variables

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

searchname.URL
searchname.Key
searchname.Title
searchname.Score
searchname.Custom1 and Custom2
searchname.Summary
searchname.RecordCount
searchname.CurrentRow
searchname.RecordsSearched
searchname.ColumnList
Standard CGI variables

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

Request

- CGI.AUTH_TYPE
- CGI.CONTENT_LENGTH
- CGI.CONTENT_TYPE
- CGI.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.CLIENT_CERT_ISSUER
- CGI.CLIENT_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:

```coldfusion
<cfif CGI.varname IS NOT "">
  CGI variable exists
</cfif>
<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 path name, 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>
</tbody>
</table>
The following table describes common CGI environment variables the browser creates and passes in the request header:

<table>
<thead>
<tr>
<th>CGI server variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<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. Format: software/version library/version.</td>
</tr>
<tr>
<td>HTTP_IF_MODIFIED_SINCE</td>
<td>The last time the page was modified. The browser determines whether to set this variable, usually in response to the server having sent the LAST_MODIFIED HTTP header. It can be used to take advantage of browser-side caching.</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 includes the client’s name, e-mail address, etc. For example: \n\n\n\n\n0 = &quot;VeriSign, Inc.&quot;, OU = VeriSign Trust Network, OU = &quot;www.verisign.com/repository/RPA Incorp. By Ref.,LiAB,LTD(c)98&quot;, OU = Persona Not Validated, OU = Digital ID Class I - Microsoft, CN = Matthew 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 example: \n\n\n\n\n0 = &quot;VeriSign, Inc.&quot;, OU = VeriSign Trust Network, OU = &quot;www.verisign.com/repository/RPA Incorp. By Ref.,LiAB,LTD(c)98&quot;, CN = VeriSign Class I 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 of interest to developers, so they can integrate with other software that uses client certificates.</td>
</tr>
</tbody>
</table>
CHAPTER 2
ColdFusion Tags

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

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

Contents
Tag summary ................................................................. 34
Tags by function ......................................................... 39
Tag changes since ColdFusion 5 ................................. 42
Tag descriptions ......................................................... 47
## 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>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>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>CFML tag</td>
<td>Category</td>
<td>Description</td>
</tr>
<tr>
<td>-------------</td>
<td>---------------------------------------------</td>
<td>-----------------------------------------------------------------------------</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>cfflush</td>
<td>Data output tags, Page processing tags</td>
<td>Flushes currently available data to client</td>
</tr>
<tr>
<td>cfform</td>
<td>Forms tags</td>
<td>Builds input form; performs client-side input validation</td>
</tr>
<tr>
<td>cfftp</td>
<td>Forms tags, Extensibility tags, Internet Protocol tags</td>
<td>Permits FTP file operations</td>
</tr>
<tr>
<td>cffunction</td>
<td>Extensibility tags</td>
<td>Defines function that you build in CFML</td>
</tr>
<tr>
<td>cfgrid</td>
<td>Forms tags</td>
<td>Displays tabular grid control, in cfform tag</td>
</tr>
<tr>
<td>cfgridcolumn</td>
<td>Forms tags</td>
<td>Used in cfform; defines columns in a cfgrid</td>
</tr>
<tr>
<td>cfgridrow</td>
<td>Forms tags</td>
<td>Defines a grid row; used with cfgrid</td>
</tr>
<tr>
<td>cfgridupdate</td>
<td>Forms tags</td>
<td>Directly updates ODBC data source from edited grid data</td>
</tr>
<tr>
<td>cfheader</td>
<td>Data output tags, Page processing tags</td>
<td>Generates HTTP headers</td>
</tr>
<tr>
<td>cfhtmlhead</td>
<td>Forms tags, Page processing tags</td>
<td>Writes text and HTML to HEAD section of page</td>
</tr>
<tr>
<td>cfhttp</td>
<td>Forms tags, 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>cfhttpparam</td>
<td>Forms tags, Internet Protocol tags</td>
<td>Specifies parameters required for a cfhttp POST operation; used with cfhttp</td>
</tr>
<tr>
<td>cfif</td>
<td>Flow-control tags</td>
<td>Creates IF-THEN-ELSE constructs</td>
</tr>
<tr>
<td>cfimport</td>
<td>Application framework tags</td>
<td>Imports JSP tag libraries into a CFML page</td>
</tr>
<tr>
<td>cfinclude</td>
<td>Flow-control tags</td>
<td>Embeds references to ColdFusion pages</td>
</tr>
<tr>
<td>cfindex</td>
<td>Extensibility tags</td>
<td>Creates Verity search indexes</td>
</tr>
<tr>
<td>cfinput</td>
<td>Forms tags</td>
<td>Creates an input element (radio button, check box, text entry box); used in cfform</td>
</tr>
<tr>
<td>cfinsert</td>
<td>Database manipulation tags</td>
<td>Inserts records in a data source</td>
</tr>
<tr>
<td>cfinvoke</td>
<td>Extensibility tags</td>
<td>Invokes component methods from a ColdFusion page or component</td>
</tr>
<tr>
<td>CFML tag</td>
<td>Category</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------</td>
<td>------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>cfinvokeargument</td>
<td>Extensibility tags</td>
<td>Passes a parameter to a component method or a web service</td>
</tr>
<tr>
<td>cfldap</td>
<td>Forms tags, Internet Protocol tags</td>
<td>Provides access to LDAP directory servers</td>
</tr>
<tr>
<td>cflocation</td>
<td>Flow-control tags</td>
<td>Controls execution of a page</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>Extensibility tags</td>
<td>Defines a container for user login and authentication code</td>
</tr>
<tr>
<td>cfloginuser</td>
<td>Extensibility tags</td>
<td>Identifies an authenticated user to ColdFusion</td>
</tr>
<tr>
<td>cflogout</td>
<td>Extensibility 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>Forms tags, Internet Protocol tags</td>
<td>Assembles and posts an e-mail message</td>
</tr>
<tr>
<td>cfmailparam</td>
<td>Forms tags, Internet Protocol tags</td>
<td>Attaches a file or adds a header to an e-mail message</td>
</tr>
<tr>
<td>cfmailpart</td>
<td>Forms tags, 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>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>Forms tags, 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>CFML tag</td>
<td>Category</td>
<td>Description</td>
</tr>
<tr>
<td>------------</td>
<td>---------------------------------------</td>
<td>-----------------------------------------------------------------------------</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>cfregistry</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 Crystal Reports report</td>
</tr>
<tr>
<td>cfrethrow</td>
<td>Exception handling tags</td>
<td>Rethrows currently active exception</td>
</tr>
<tr>
<td>cfrethrow</td>
<td></td>
<td></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>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>Other tags, Variable manipulation tags</td>
<td>Defines and controls ColdFusion settings</td>
</tr>
<tr>
<td>cfsilent</td>
<td>Data output tags, Page processing 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>cfstoredproc</td>
<td>Database manipulation tags</td>
<td>Holds database connection information; identifies a stored procedure to execute</td>
</tr>
<tr>
<td>cfswitch</td>
<td>Flow-control tags</td>
<td>Evaluates passed expression; passes control to matching cfcase tag</td>
</tr>
<tr>
<td>cftable</td>
<td>Data output tags</td>
<td>Builds a table in a ColdFusion page</td>
</tr>
<tr>
<td>cftextarea</td>
<td>Forms tags</td>
<td>Puts a one-line text entry box in a form</td>
</tr>
<tr>
<td>cfthrow</td>
<td>Exception handling tags, Flow-control 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>cftree</td>
<td>Forms tags</td>
<td>Creates tree control element; used in cfform</td>
</tr>
<tr>
<td>CFML tag</td>
<td>Category</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>----------------------------------</td>
<td>-----------------------------------------------------------------------------</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,</td>
<td>Catches exceptions in ColdFusion pages</td>
</tr>
<tr>
<td></td>
<td>Flow-control tags</td>
<td></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>
Tags by function

The following tables list Tags by their function or purpose.

Application framework tags
- cfapplication
- cfimport
- cfscript
- cfassociate
- cflock
- cferror
- cfmodule

Database manipulation tags
- cfinsert
- cfprocresult
- cfstoredproc
- cfobjectcache
- cfquery
- cftx
- cfproccparam
- cfqueryparam
- cfupdate

Data output tags
- cfchart
- cfcontent
- cfoutput
- cfchartdata
- cfflush
- cfprocessingdirective
- cfchartseries
- cfheader
- cfsilent
- cfcol
- cflog
- cftable

Debugging tags
- cftrace
- cfmodule

Extensibility tags
- cfinsert
- cfprocresult
- cfstoredproc
- cfobjectcache
- cfquery
- cftx
- cfproccparam
- cfqueryparam
- cfupdate

Other tags
- cfchart
- cfcontent
- cfoutput
- cfchartdata
- cfflush
- cfprocessingdirective
- cfchartseries
- cfheader
- cfsilent
- cfcol
- cflog
- cftable
Exception handling tags

cfcatch
cfcatch
cferror
cferror
cfthrow
cfthrow
cftry
cftry

Extensibility tags

cfchart
cfchart
cfchartdata
cfchartdata
cfchartseries
cfchartseries
cfchartseries
cfchartseries
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
collection
Internet Protocol tags

cfftp
cfhttp
cfhttpparam
cfldap
cfmail
 cfmailparam
 cfmailparam
 cfpop
\Page processing tags

cfcache
cfcontent
cfflush
cfheader
cfhtmlhead
cfinclude
 cfsetting
cfsilent

Variable manipulation tags

cfcookie
cfdump
cfparam
cfregistry
cfset
cfsetting
cfschedule
\nOther 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 .......................................................... 42
Deprecated tags, attributes, and values ............................................... 44
Obsolete tags, attributes, and values .................................................. 46

New tags, attributes, and values

This table lists tags, attributes, and attribute values that have been added in ColdFusion MX releases:

<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>loginStorage</td>
<td>ColdFusion MX 6.1</td>
</tr>
<tr>
<td>cfargument</td>
<td>All</td>
<td>ColdFusion MX</td>
</tr>
<tr>
<td>cfcache</td>
<td>timespan attribute</td>
<td>ColdFusion MX</td>
</tr>
<tr>
<td></td>
<td>cachedirectory attribute</td>
<td>ColdFusion MX</td>
</tr>
<tr>
<td>cfchart</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>All</td>
<td>ColdFusion MX</td>
</tr>
<tr>
<td>cfcollection</td>
<td>list value of action attribute</td>
<td>ColdFusion MX</td>
</tr>
<tr>
<td></td>
<td>name attribute</td>
<td>ColdFusion MX</td>
</tr>
<tr>
<td>cfcomponent</td>
<td>All</td>
<td>ColdFusion MX</td>
</tr>
<tr>
<td>cfexecute</td>
<td>variable attribute</td>
<td>ColdFusion MX 6.1</td>
</tr>
<tr>
<td>cffunction</td>
<td>All</td>
<td>ColdFusion MX</td>
</tr>
<tr>
<td>cfhttp</td>
<td>attribute</td>
<td>ColdFusion MX</td>
</tr>
<tr>
<td></td>
<td>charset attribute</td>
<td>ColdFusion MX</td>
</tr>
<tr>
<td></td>
<td>HEAD, PUT, DELETE, OPTIONS, and TRACE methods of method attribute</td>
<td>ColdFusion MX 6.1</td>
</tr>
<tr>
<td></td>
<td>multipart attribute</td>
<td>ColdFusion MX 6.1</td>
</tr>
<tr>
<td></td>
<td>getasbinary attribute</td>
<td>ColdFusion MX 6.1</td>
</tr>
<tr>
<td></td>
<td>proxyUser</td>
<td>ColdFusion MX 6.1</td>
</tr>
<tr>
<td></td>
<td>proxyPassword</td>
<td>ColdFusion MX 6.1</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 attribute</td>
<td>ColdFusion MX 6.1</td>
</tr>
<tr>
<td></td>
<td>mimeType attribute</td>
<td>ColdFusion MX 6.1</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>cfimport</td>
<td>All</td>
<td>ColdFusion MX</td>
</tr>
<tr>
<td>cfinvoke</td>
<td>All</td>
<td>ColdFusion MX</td>
</tr>
<tr>
<td>cfinvokeargument</td>
<td>All</td>
<td>ColdFusion MX</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 attribute</td>
<td>ColdFusion MX 6.1</td>
</tr>
<tr>
<td></td>
<td>failto attribute</td>
<td>ColdFusion MX 6.1</td>
</tr>
<tr>
<td></td>
<td>replyTo attribute</td>
<td>ColdFusion MX 6.1</td>
</tr>
<tr>
<td></td>
<td>userName attribute</td>
<td>ColdFusion MX 6.1</td>
</tr>
<tr>
<td></td>
<td>password attribute</td>
<td>ColdFusion MX 6.1</td>
</tr>
<tr>
<td></td>
<td>wrapText attribute</td>
<td>ColdFusion MX 6.1</td>
</tr>
<tr>
<td>cfmailparam</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>cfojbject</td>
<td>All</td>
<td>ColdFusion MX</td>
</tr>
<tr>
<td>cfobjectcache</td>
<td>All</td>
<td>ColdFusion MX</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>creturn</td>
<td>All</td>
<td>ColdFusion MX</td>
</tr>
<tr>
<td>cfsetting</td>
<td>requestTimeOut attribute</td>
<td>ColdFusion MX</td>
</tr>
<tr>
<td>cfthrow</td>
<td>object attribute</td>
<td>ColdFusion MX</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>timeout attribute</td>
<td>ColdFusion MX</td>
</tr>
<tr>
<td>cffile</td>
<td>attributes attribute value archive</td>
<td>ColdFusion MX</td>
</tr>
<tr>
<td></td>
<td>attributes attribute value system</td>
<td>ColdFusion MX</td>
</tr>
<tr>
<td></td>
<td>attributes attribute value temporary</td>
<td>ColdFusion MX</td>
</tr>
<tr>
<td>cfgraph</td>
<td>All</td>
<td>ColdFusion MX</td>
</tr>
<tr>
<td>cfggraphdata</td>
<td>All</td>
<td>ColdFusion MX</td>
</tr>
<tr>
<td>cfgridupdate</td>
<td>connectString attribute</td>
<td>ColdFusion MX</td>
</tr>
<tr>
<td></td>
<td>dbName attribute</td>
<td>ColdFusion MX</td>
</tr>
<tr>
<td></td>
<td>dbServer attribute</td>
<td>ColdFusion MX</td>
</tr>
<tr>
<td></td>
<td>dbType attribute</td>
<td>ColdFusion MX</td>
</tr>
<tr>
<td></td>
<td>provider attribute</td>
<td>ColdFusion MX</td>
</tr>
<tr>
<td></td>
<td>providerDSN attribute</td>
<td>ColdFusion MX</td>
</tr>
<tr>
<td>cfindex</td>
<td>external attribute</td>
<td>ColdFusion MX</td>
</tr>
<tr>
<td>cfinsert</td>
<td>connectString attribute</td>
<td>ColdFusion MX</td>
</tr>
<tr>
<td></td>
<td>dbName attribute</td>
<td>ColdFusion MX</td>
</tr>
<tr>
<td></td>
<td>dbServer attribute</td>
<td>ColdFusion MX</td>
</tr>
<tr>
<td></td>
<td>dbType attribute</td>
<td>ColdFusion MX</td>
</tr>
<tr>
<td></td>
<td>provider attribute</td>
<td>ColdFusion MX</td>
</tr>
<tr>
<td></td>
<td>providerDSN attribute</td>
<td>ColdFusion MX</td>
</tr>
<tr>
<td>cfldap</td>
<td>filterConfig attribute</td>
<td>ColdFusion MX</td>
</tr>
<tr>
<td></td>
<td>filterFile attribute</td>
<td>ColdFusion MX</td>
</tr>
<tr>
<td>cflog</td>
<td>date attribute value &quot;No&quot;</td>
<td>ColdFusion MX</td>
</tr>
<tr>
<td></td>
<td>thread attribute value &quot;No&quot;</td>
<td>ColdFusion MX</td>
</tr>
<tr>
<td></td>
<td>time attribute value &quot;No&quot;</td>
<td>ColdFusion MX</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>cfquery</td>
<td>connectString</td>
<td>ColdFusion MX</td>
</tr>
<tr>
<td></td>
<td>dbName</td>
<td>ColdFusion MX</td>
</tr>
<tr>
<td></td>
<td>dbServer</td>
<td>ColdFusion MX</td>
</tr>
<tr>
<td></td>
<td>The following dbType attribute values:</td>
<td>ColdFusion MX</td>
</tr>
<tr>
<td></td>
<td>• dynamic</td>
<td>(The value query is valid.)</td>
</tr>
<tr>
<td></td>
<td>• ODBC</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Oracle73</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Oracle80</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Sybase11</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• OLEDB</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• DB2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>provider</td>
<td>ColdFusion MX</td>
</tr>
<tr>
<td></td>
<td>providerDSN</td>
<td>ColdFusion MX</td>
</tr>
<tr>
<td></td>
<td>sql</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 attribute</td>
<td>ColdFusion MX</td>
</tr>
<tr>
<td>cfscvrlent</td>
<td>All</td>
<td>ColdFusion MX</td>
</tr>
<tr>
<td>cfscvrlentparam</td>
<td>All</td>
<td>ColdFusion MX</td>
</tr>
<tr>
<td>cfslider</td>
<td>img</td>
<td>ColdFusion MX</td>
</tr>
<tr>
<td></td>
<td>imgStyle</td>
<td>ColdFusion MX</td>
</tr>
<tr>
<td></td>
<td>grooveColor</td>
<td>ColdFusion MX</td>
</tr>
<tr>
<td>cfscvredproc</td>
<td>connectString</td>
<td>ColdFusion MX</td>
</tr>
<tr>
<td></td>
<td>dbName</td>
<td>ColdFusion MX</td>
</tr>
<tr>
<td></td>
<td>dbServer</td>
<td>ColdFusion MX</td>
</tr>
<tr>
<td></td>
<td>provider</td>
<td>ColdFusion MX</td>
</tr>
<tr>
<td></td>
<td>providerDSN</td>
<td>ColdFusion MX</td>
</tr>
<tr>
<td>cfscvred</td>
<td>connectString</td>
<td>ColdFusion MX</td>
</tr>
<tr>
<td></td>
<td>dbName</td>
<td>ColdFusion MX</td>
</tr>
<tr>
<td></td>
<td>dbServer</td>
<td>ColdFusion MX</td>
</tr>
<tr>
<td></td>
<td>provider</td>
<td>ColdFusion MX</td>
</tr>
<tr>
<td></td>
<td>providerDSN</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>cfimpersonate</td>
<td>All</td>
<td>ColdFusion MX</td>
</tr>
<tr>
<td>cfindex action</td>
<td>attribute value</td>
<td>ColdFusion MX</td>
</tr>
<tr>
<td>cfinternaladminsecurity</td>
<td>All</td>
<td>ColdFusion MX</td>
</tr>
<tr>
<td>cfnewinternaladminsecurity</td>
<td>All</td>
<td>ColdFusion MX</td>
</tr>
<tr>
<td>cfsetting catchExceptionsByPattern</td>
<td>attribute</td>
<td>ColdFusion MX</td>
</tr>
</tbody>
</table>

This tag did not appear in CFML Reference.
**cfabort**

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

**Category**
Flow-control tags

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

**See also**
cfbreak, cfexecute, cfexit, cff, cflocation, 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>showError</td>
<td>Optional</td>
<td></td>
<td>Error to display, in a standard ColdFusion error page, when tag executes.</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>
    The value of myVariable after incrementing through the loop #Counter# times is:
    #myVariable#
</cfoutput>

<h3>Example B: Use cfabort to halt the instruction</h3>
```
<!--- reset the variable and show the use of cfabort --->
<cfset myVariable = 3>
<!--- now, perform a loop that increments this value --->
<cfloop from = "1" to = "4" index = "Counter">
<!--- on the second time through the loop, cfabort --->
<cfif Counter is 2>
    <cfabort>
<!--- processing is stopped, and subsequent operations are not carried out --->
</cfif>
<cfelse>
    <cfset myVariable = myVariable + 1>
</cfif>
</cfloop>

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

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

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

Category
Forms tags

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

See also
cfform, cfobject, cfapplet

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 pixel</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.

Example

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

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

**Description**

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

**Category**

Application framework tags

**Syntax**

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

**See also**

cfassociate, cferror, cflock, cfmodule

**History**

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.)

**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>See Description</td>
<td></td>
<td>Name of application. Up to 64 characters. For Application and Session variables: Required. For Client variables: Optional</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 |

---

**cfapplication** 51
Usage

This tag is typically used in the Application.cfm file, to set defaults for a ColdFusion application. This tag enables application variables, unless they are disabled in the ColdFusion Administrator. The Administrator setting also overrides the `sessionManagement` attribute. For more information, see Configuring and Administering ColdFusion MX.

**Server, application, and session variables**

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

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

For information about locking scopes, see `cflock` on page 221.

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

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Req/Opt</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>clientStorage</td>
<td>Optional</td>
<td>registry</td>
<td>How client variables are stored:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• datasource_name: in ODBC or native data source. You must create storage repository in the Administrator.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• registry: in the system registry.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• cookie: on client computer in a cookie. Scalable. If client disables cookies in the browser, client variables do not work.</td>
</tr>
<tr>
<td>setClientCookies</td>
<td>Optional</td>
<td>Yes</td>
<td>• Yes: enables client cookies</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• 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.</td>
</tr>
<tr>
<td>sessionManagement</td>
<td>Optional</td>
<td>No</td>
<td>• Yes: enables session variables</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• No</td>
</tr>
<tr>
<td>sessionTimeout</td>
<td>Optional</td>
<td>Specified in Variables page of ColdFusion Administrator</td>
<td>Lifespan of session variables. CreateTimeSpan function and values in days, hours, minutes, and seconds, separated by commas.</td>
</tr>
<tr>
<td>applicationTimeout</td>
<td>Optional</td>
<td>Specified in Variables page of ColdFusion Administrator</td>
<td>Lifespan of application variables. CreateTimeSpan function and values in days, hours, minutes, and seconds, separated by commas.</td>
</tr>
<tr>
<td>setDomainCookies</td>
<td>Optional</td>
<td>No</td>
<td>• Yes: Sets CFID and CFTOKEN cookies for a domain (not a host). Required, for applications running on clusters.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• No</td>
</tr>
</tbody>
</table>
If you use this tag to activate the Application and Client scopes, ColdFusion saves the application name as a key, whose maximum length is 64 characters. If an application name is longer than this, the client store fails during database processing.

**Note:** The CFTOKEN variable is 8 bytes in length. Its range is 10000000 — 99999999. If you specify **ClientStorage=cookie**, any Client scope variables set following a **cfflush** tag are not saved in the Client browser.

**Example**

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

<!--- This example shows how to use cflock to guarantee consistent data updates to variables in Application, Server, and Session scopes. --->

<h3>cfapplication Example</h3>

<cflock scope = "Session" timeout = "30" type = "Exclusive">
    <cfif NOT IsDefined("session.size")>
        <cfset session.size = ">
    </cfif>
    <cfif NOT IsDefined("session.color")>
        <cfset session.color = ">
    </cfif>
</cflock>

<!--- Use the application scope for the application variable. This variable keeps track of total number of turtlenecks sold. --->
<cflock scope = "Application" timeout = "30" type = "Exclusive">
    <cfif NOT IsDefined("application.number")>
        <cfset application.number = 1>
    </cfif>
</cflock>

<cflock scope = "Application" timeout = "30" type = "readOnly">
    <cfoutput>
        E-Turtleneck is proud to say that we have sold #application.number# turtlenecks to date.
    </cfoutput>
</cflock>
```

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

```
**cfargument**

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

**Category**
Extensibility tags

**Syntax**
```html
<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>
<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></td>
<td>• guid – The argument must be a UUID or GUID of the form <code>xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx</code> where each <code>x</code> is a character representing a hexadecimal number (0-9A-F).</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• numeric</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• query</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• string</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• struct</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• uuid – The argument must be a ColdFusion UUID of the form <code>xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxxxxxx</code> where each <code>x</code> is a character representing a hexadecimal number (0-9A-F).</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>• 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.</td>
</tr>
</tbody>
</table>
This tag must be in a \texttt{cffunction} tag, before any other tags in the \texttt{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: \texttt{#myargument#}
  
  (This example accesses the argument \texttt{myargument}.)

- Using the arguments scope as an array: \texttt{#arguments[1]#}
  
  (This example accesses the first defined argument in the \texttt{cffunction})

- Using the arguments scope as a struct: \texttt{#arguments.myargument#}
  
  (This example accesses the argument \texttt{myargument} in the array)

\textbf{Example}

\begin{verbatim}
<!--- This example defines a function that takes a course number parameter and returns the course description. --->
<cffunction name="getDescript">
  <!--- 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="cfsnippets">
    SELECT Descript
    FROM Courses
    WHERE Number = '#Course_Number#'
  </cfquery>
  <!--- Specify the variable that the function returns --->
  <cfreturn Description.Descript>
</cffunction>
\end{verbatim}
**cfassociate**

**Description**

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

**Category**

Application framework tags

**Syntax**

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

**See also**

cfapplication, cferror, cflock, cfmodule

**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>Base tag name</td>
<td></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**

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

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

Description
This tag is obsolete. Use the newer security tools; see “Authentication functions” on page 367 and Chapter 16, “Securing Applications,” in Developing ColdFusion MX Applications.

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

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

**Category**
Flow-control tags

**Syntax**
```xml
<cfbreak>
```

**See also**
`cfabort`, `cfexecute`, `cfif`, `cflocation`, `cfloop`, `cfswitch`, `cfthrow`, `cftry`

**Example**
```xml
<!--- This shows the use of cfbreak to exit a loop when a condition is met --->
<!--- select courses; use cfloop to find a condition; then break the loop --->
<!--- check that number is numeric --->
<cfif IsDefined("form.number")>
  <cfif Not IsNumeric(form.number)>
    <cfabort>
  </cfif>
</cfif>

<cfquery name="GetCourses" datasource="cfsnippets">
  SELECT *
  FROM Courses
  ORDER by Course_Number
</cfquery>

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

<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>
    <cfelse>
      <br> Searching...
    </cfif>
  </cfloop>
</cfif>
```

<!--- 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>
    <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
<cfcatche
  action = "action"
  directory = "directory_name"
  timespan = "value"
  expireURL = "wildcarded_URL_reference"
  username = "username"
  password = "password"
  port = "port_number"
  protocol = "protocol">

See also
cfflush, cfheader, cfhtmlhead, cfsetting, cfsilent

History
ColdFusion MX:
• Deprecated the timeout and cachedirectory 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>
</table>
| action    | Optional| cache   | • cache: server-side and client-side caching.  
• flush: refresh cached page(s).  
• clientcache: browser-side caching only. To cache a personalized page, use this option.  
• servercache: server-side caching only. Not recommended.  
• optimal: same as "cache". |
| directory | Optional| cf_root/cache | Absolute path of cache directory. |
| timespan  | Optional| Page is flushed only when cfcache action = "flush" is executed | The interval until the page is flushed from the cache.  
• A decimal number of days. For example:  
  - ".25", for one-fourth day (6 hours)  
  - "1", for one day  
  - "1.5", for one and one half days  
• A return value from the CreateTimeSpan function. For example:  
  "#CreateTimeSpan(0, 6, 0, 0)#" |
| expireURL | Optional| Flush all cached pages | Used with action = "flush". A URL reference. ColdFusion matches it against the mappings in the specified cache directory. Can include wildcards. For example: "*/view.cfm?id=*" |
| username  | Optional| A username. Provide this if the page requires authentication at the web server level. |
| password  | Optional| A password. Provide this if the page requires authentication at the web server level. |
| port      | Optional| The current page port | Port number of the web server from which the URL is requested. In the internal call from cfcache to cfhttp, ColdFusion resolves each URL variable in the page; this ensures that links in the page remain functional. |
| protocol  | Optional| The current page protocol | Protocol that is used to create URL from cache.  
• http://  
• https:// |

Usage

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

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

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

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

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

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

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

**Example**

<!--- This example produces as many cached files as there are URL parameter permutations. --->

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

---

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

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

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

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

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

**Example**

<!--- This example produces as many cached files as there are URL parameter permutations. --->

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

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
cfdefaultcase, cfswitch

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 cfcase 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 cfcase tag body can include HTML and text, and CFML tags, functions, variables, and expressions. You do not have to explicitly break out of the cfcase tag, as you do in some languages.

One cfcase 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.--->
<cfset score="7">
<cfswitch expression="#score#">
  <cfcase value="10">
    <cfset grade="A">
  <cfcase value="10">
  <cfset grade="A"
<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
<cfcatch type = "exceptiontype">
    Exception processing code here
</cfcatch>

See also
cftry, cferror, cfrethrow, cfthrow, Chapter 14, “Handling Errors,” in Developing ColdFusion MX Applications

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>
</table>
| type      | Optional | Any     | - application: catches application exceptions  
- database: catches database exceptions  
- template: catches ColdFusion page exceptions  
- security: catches security exceptions  
- object: catches object exceptions  
- missingInclude: catches missing include file exceptions  
- expression: catches expression exceptions  
- lock: catches lock exceptions  
- custom_type: catches 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:

```cfc
#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:

```cfc
<cfthrow type = "MyApp.BusinessRuleException.InvalidAccount">
```

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

```cfc
<cfcatch type = "MyApp.BusinessRuleException.InvalidAccount">
```

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

```cfc
<cfcatch type = "MyApp.BusinessRuleException">
```

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

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

<!--- cftry example, using TagContext to display the tag stack. --->
<h3>cftry Example</h3>
<!--- open a cftry block --->
<cftry>

<!--- note misspelled tablename "employees" as "employeeas" --->
<cfquery name = "TestQuery" dataSource = "cfsnippets">
SELECT *
FROM EMPLOYEEAS
</cfquery>

<!--- specify the type of error for which we search --->
<cfcatch type = "Database">

<!--- the message to display --->
<h3>You've Thrown a Database Error</h3>
<cfoutput>

<!--- and the diagnostic message from the ColdFusion server --->
<p>#cfcatch.message#</p>
<p>Caught an exception, type = #CFCATCH.TYPE#</p>
</cfoutput>

</cfcatch>

<!--- other processing goes here --->
<p>... other processing goes here --->
</p>
</cftry>
The contents of the tag stack are:

```
<cfloop index = i from = 1 to = #ArrayLen(CFCATCH.TAGCONTEXT)#>
    <cfset sCurrent = #CFCATCH.TAGCONTEXT[i]#>
    <br>#i# #sCurrent["ID"]#
    (#sCurrent["LINE"]#, #sCurrent["COLUMN"]#)
    #sCurrent["TEMPLATE"]#
</cfloop>
```

</cfoutput>
</cfcatch>
</cftry>
cfchart

Description
Generates and displays a chart.

Category
Data output tags, Extensibility tags

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

See also
cfchartdata, cfchartseries

History
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>format</td>
<td>flash</td>
<td>flash</td>
<td>File format in which to save graph.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>jpg</td>
<td>• flash</td>
</tr>
<tr>
<td></td>
<td></td>
<td>png</td>
<td>• jpg</td>
</tr>
<tr>
<td>chartHeight</td>
<td>240</td>
<td></td>
<td>Chart height; integer number of pixels</td>
</tr>
<tr>
<td>chartWidth</td>
<td>320</td>
<td></td>
<td>Chart width; integer number of pixels</td>
</tr>
<tr>
<td>scaleFrom</td>
<td>Determined by data</td>
<td></td>
<td>Y-axis minimum value; integer.</td>
</tr>
<tr>
<td>scaleTo</td>
<td>Determined by data</td>
<td></td>
<td>Y-axis maximum value; integer</td>
</tr>
<tr>
<td>showXGridlines</td>
<td>no</td>
<td>yes</td>
<td>• yes: display X-axis gridlines</td>
</tr>
<tr>
<td></td>
<td></td>
<td>no</td>
<td>• no: hide X-axis gridlines</td>
</tr>
<tr>
<td>showYGridlines</td>
<td>yes</td>
<td></td>
<td>• yes: display Y-axis gridlines</td>
</tr>
<tr>
<td></td>
<td></td>
<td>no</td>
<td>• no: hide Y-axis gridlines</td>
</tr>
<tr>
<td>gridlines</td>
<td>10, including top and bottom</td>
<td></td>
<td>Number of grid lines to display on value axis, including axis; positive integer,</td>
</tr>
<tr>
<td>seriesPlacement</td>
<td>default</td>
<td></td>
<td>Applies to charts that have more than one data series. Relative positions of 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>foregroundColor</td>
<td>black</td>
<td></td>
<td>Color of text, gridlines, and labels.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Hex value or supported named color; see name list in the Usage section.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>For a hex value, use the form &quot;##xxxxxx&quot;, where x = 0-9 or A-F; use two pound signs or none.</td>
</tr>
<tr>
<td>dataBackgroundColor</td>
<td>white</td>
<td></td>
<td>Color of area around chart data.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Hex value or supported named color; see name list in the Usage section.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>For a hex value, use the form &quot;##xxxxxx&quot;, where x = 0-9 or A-F; use two pound signs or none.</td>
</tr>
<tr>
<td>showBorder</td>
<td>no</td>
<td></td>
<td>• yes</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• no</td>
</tr>
<tr>
<td>font</td>
<td>arial</td>
<td></td>
<td>Name of text font</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• arial</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• times</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• courier</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• arialunicodeMS. This option is required, if you are using a double-byte character set on UNIX, or using a double-byte character set on Windows with a file type of Flash.</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>fontSize</td>
<td>11</td>
<td>Font size; integer</td>
<td></td>
</tr>
</tbody>
</table>
| fontBold          | no      | • yes  
               | • no                                                                   |
| fontItalic        | no      | • Yes  
               | • No                                                                     |
| labelFormat       | number  | Format for Y-axis labels.                                                   |
|                   |         | • number  
               | • currency  
               | • percent  
               | • date                                                       |
| xAxisTitle        | text    | X-axis title                                                                 |
| yAxisTitle        | text    | Y-axis title                                                                 |
| xAxisType         | category | The axis indicates the data category. Data is sorted according to the sortXAxis attribute. |
|                   |         | • scale  
               | The axis is numeric. All cfchartdata item attribute values must numeric. The X axis is automatically sorted numerically. |
| yAxisType         | category | Currently has no effect, as Y axis is always used for data values. Valid values are category and scale |
| sortXAxis         | no      | • yes: display column labels in alphabetic order along X-axis  
               | • no                                                                   |
|                   |         | Ignored if xAxisType attribute is scale.                                    |
| show3D            | no      | • yes: display chart with three-dimensional appearance  
               | • no                                                                    |
| xOffset            | 0.1     | Applies if show3D="yes". Number of units by which to display the chart as angled, horizontally.  
               | • A number in the range -1 - 1, where "-1" specifies 90 degrees left and "1" specifies 90 degrees right |
| yOffset            | 0.1     | Applies if show3D="yes". Number of units by which to display the chart as angled, vertically.  
               | • A number in the range -1 - 1, where "-1" specifies 90 degrees down, and "1" specifies 90 degrees up |
| rotated            | no      | • yes: rotate chart 90 degrees. For a horizontal bar chart, use this option.  
               | • no                                                                    |
| showLegend        | yes     | • yes: if chart contains more than one data series, display legend  
               | • no                                                                    |
The **cfchart** tag defines a **container** in which a graph displays: its height, width, background color, labels, and so on. The **cfchartseries** tag defines the style in which data displays: bar, line, pie, and so on. The **cfchartdata** tag defines a data point.

---

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

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

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

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

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

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

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

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

Flash does not conform fully to UNIX X11 naming conventions.

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

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

Category
Data output tags, Extensibility tags

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

See also
cfchart, cfchartseries

History
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></td>
<td>string; data point name</td>
<td></td>
</tr>
<tr>
<td>value</td>
<td></td>
<td>number or expression; data point value</td>
<td></td>
</tr>
</tbody>
</table>
cfchartseries

Description

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

Category

Data output tags, Extensibility tags

Syntax

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

See also

cfchart, cfchartdata

History

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>type</td>
<td>Required</td>
<td></td>
<td>Sets the chart display style:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• bar</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• line</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• pyramid</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• area</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• cone</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• curve</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• cylinder</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• step</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• scatter</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• pie</td>
</tr>
<tr>
<td>query</td>
<td>Optional</td>
<td></td>
<td>Name of ColdFusion query from which to get data.</td>
</tr>
<tr>
<td>itemColumn</td>
<td>Required if query attribute is specified</td>
<td></td>
<td>Name of a column in the query specified in the query attribute; contains the item label for a data point to graph.</td>
</tr>
<tr>
<td>valueColumn</td>
<td>Required if query attribute is specified</td>
<td></td>
<td>Name of a column in the query specified in the query attribute; contains data values to graph.</td>
</tr>
</tbody>
</table>
Usage

If a chart has multiple line series, ColdFusion interpolates missing data points in the common X axis range. For example, if one series has data points at 0, 10, and 20 on the X axis and a second series has data points at 0, 20, and 30, ColdFusion calculates and displays a data point for X=10 on the second series line, but it does not calculate an X=30 value for the first series.

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

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

ColdFusion Generates an error if you use more than 16 chartseries tags in one cfchart tag.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Req/Opt</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>seriesLabel</td>
<td>Optional</td>
<td></td>
<td>Text of data series label</td>
</tr>
<tr>
<td>seriesColor</td>
<td>Optional</td>
<td></td>
<td>Color of the main element (such as the bars) of a chart. For a pie chart, the color of the first slice. Hex value or supported named color; see name list in the cfchart Usage section. For a hex value, use the form &quot;##xxxxxx&quot;, where x = 0-9 or A-F; use two pound signs or none.</td>
</tr>
<tr>
<td>paintStyle</td>
<td>Optional</td>
<td>plain</td>
<td>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</td>
</tr>
<tr>
<td>markerStyle</td>
<td>Optional</td>
<td>rectangle</td>
<td>Applies to chartseries type attribute values line, curve and scatter, and show3D attribute value no. Sets the icon that marks a data point: • rectangle • triangle • diamond • circle • letter • mcross • snow • rcross</td>
</tr>
<tr>
<td>colorList</td>
<td>Optional</td>
<td></td>
<td>Applies if chartseries type attribute = &quot;pie&quot;. Sets pie slice colors. Comma-delimited list of hex values or supported, named web colors; see name list in the cfchart Usage section. For a hex value, use the form &quot;##xxxxxx&quot;, where x = 0-9 or A-F; use two pound signs or none.</td>
</tr>
</tbody>
</table>
**cfcol**

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

**Category**
Data output tags

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

**See also**
cfcontent, cfoutput, cftable

**History**
ColdFusion MX: Added the ability to construct dynamic *cfcol* 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 <em>cftable</em> 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 <em>table</em> tag. If the surrounding <em>cftable</em> tag includes the htmltable attribute, <em>width</em> specifies the percent of the table width and it does not truncate text; otherwise, <em>width</em> specifies the number of characters.</td>
</tr>
</tbody>
</table>
| align     | Optional| Left    | Column alignment
• Left
• Right
• Center |
| text      | Required|         | Double-quotiation mark-delimited text; determines what to display. Rules same as for cfoutput sections. You can embed hyperlinks, image references, and input controls. |

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

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

**Example**
```
<!--- This example shows the use of cfcol and cftable to align
     information returned from a query --->
<!--- query selects information from cfsnippets data source --->
<cfquery name = "GetEmployees" dataSource = "cfsnippets">
    SELECT Emp_ID, FirstName, LastName, EMail, Phone, Department
</cfquery>
```
FROM Employees
</cfquery>
<html>
<body>
<h3>cfcoll 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 cfcoll tag sets the width of a column in the table, the header information and the text/CFML for the cell --->
<cfcoll header = "<b>ID</b>"
    align = "Left"
    width = 2
    text = "#Emp_ID#">
    <cfcoll header = "<b>Name/Email</b>"
        align = "Left"
        width = 15
        text = "<a href = 'mailto:#Email#'>#FirstName# #LastName#</a>">
        <cfcoll header = "<b>Phone Number</b>"
            align = "Center"
            width = 15
            text = "#Phone#">
    </cfcoll>
    <cfcoll header = "<b>Email</b>"
        align = "Left"
        width = 15
        text = "#Email#">
    </cfcoll>
</cftable>
</body>
</html>
**cfcollection**

**Description**

Creates, registers, and administers Verity search engine collections.

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

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

An internal collection can be created in these ways:

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

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

**Category**

**Extensibility tags**

**Syntax**

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

**See also**

cfexecute, cfindex, cfobject, cfreport, cfsearch, cfwddx

**History**

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

Usage

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

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

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

If more than instance of the cfcollection tag might access or modify a collection simultaneously, use the cflock tag to protect the collection from attempts at simultaneous operations.

To register a collection with K2Server, you update the k2server.ini file.
Before you attempt to delete or purge a collection that is also opened by the K2Server, you must stop the K2Server. If you do not, some files may be open, and ColdFusion might not complete the action.

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

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

The ColdFusion MX Administrator Verity > Collections page displays the information that is returned when you use the list attribute.

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

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

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

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

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

To add content to a collection, use cfindex. To search a collection, use cfsearch.
With the European Verity Locales language pack installed, the `language` attribute of this tag supports the following options:

<table>
<thead>
<tr>
<th>bokmal</th>
<th>french</th>
<th>norweg</th>
</tr>
</thead>
<tbody>
<tr>
<td>danish</td>
<td>german</td>
<td>portug</td>
</tr>
<tr>
<td>dutch</td>
<td>italian</td>
<td>portuguese</td>
</tr>
<tr>
<td>english</td>
<td>nynorsk</td>
<td>spanish</td>
</tr>
<tr>
<td>finnish</td>
<td>norwegian</td>
<td>swedish</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>arabic</th>
<th>hungarian</th>
<th>russian</th>
</tr>
</thead>
<tbody>
<tr>
<td>czech</td>
<td>japanese</td>
<td>simplified_chinese</td>
</tr>
<tr>
<td>greek</td>
<td>korean</td>
<td>traditional_chinese</td>
</tr>
<tr>
<td>hebrew</td>
<td>polish</td>
<td>turkish</td>
</tr>
</tbody>
</table>

The default location of Verity collections is as follows:

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

**Example**

```cfml
<!------------------------------------------------------------------------->
Check for server platform and use it's default Verity Collection directory. You may need to change the path if you did not install ColdFusion MX in the default directory.
-------------------------------------------------------------------------
<cfif Find("Windows", Server.OS.Name)>
    <cfset collPath = "C:\CFusionMX\Verity\Collections\">
<cfelse>
    <cfset collPath = "/opt/coldfusionmx/verity/collections/">
</cfif>

<!-------------------------------------------------------------------->
Process form input and do the requested cfcollection operation.
--------------------------------------------------------------------->
<cfif IsDefined("form.CollectionName") AND IsDefined("form.CollectionAction")>
    <cfswitch expression="#FORM.CollectionAction#">
    <cfcase value="Create">
        <cfcollection action="CREATE" collection="#FORM.CollectionName#"
        path="#collPath#">
        <h3>Collection created.<br>
        Use CFINDEX to populate it.</h3>
    </cfcase>
    <cfcase value="Repair">
        <cfcollection action="REPAIR" collection="#FORM.CollectionName#">
        <h3>Collection repaired.</h3>
    </cfcase>
    </cfswitch>
</cfif>
```

---

(cfcollection 83)
<cfcase value="Optimize">
    <cfcollection action="OPTIMIZE" collection="#FORM.CollectionName#">
        <h3>Collection optimized.</h3>
    </cfcase>
<cfcase value="Delete">
    <cfcollection action="DELETE" collection="#FORM.CollectionName#">
        <h3>Collection deleted.</h3>
    </cfcase>
<cfcase>
</cfswitch>
<cfelse>
    <h3>Please enter a name for your collection</h3>
</cfif>

<!--------------------------------------------------------------------
Form to specify the collection name and action
--------------------------------------------------------------------->

<form action="#CGI.SCRIPT_NAME" 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>
    Use the default value or enter your own Collection name</p>
    <input type="Text" name="CollectionName" value="snippets" />
    <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 instatiated.

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**

```xml
<cfcomponent
  extends = "anotherComponent"
  output = "yes" or "no"
  displayname = "text string">
    hint = "text string">
  variable declarations
  <cffunction ...
    ...
  </cffunction>
  ...
</cfcomponent>
```

**See also**

cfargument, cffunction, cfinvoke, cfinvokeargument, cfobject, cfproperty, cfreturn. Chapter 11, “Building and Using ColdFusion Components,” in Developing ColdFusion MX Applications
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>Name of parent component from which to inherit methods and properties.</td>
<td></td>
</tr>
</tbody>
</table>
| output    | Optional | Component body displayable text is processed as standard CFML. Specifies whether constructor code in the component can generate HTML output; does not affect output in the body of cffunction tags in the component.  
- 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.  
- no: Constructor code is processed as if it were within a cfsilent tag.  
- If you do not specify this attribute, constructor code is processed as standard CFML. Any variables must be in cfoutput tags. |
| displayname | Optional | name attribute value | A string to be displayed when using introspection to show information about the CFC. The information appears on the heading, following the component name. |
| hint      | Optional | TEXT to be displayed when using introspection to show information about the CFC. The hint attribute value appears below the component name heading. This attribute can be useful for describing the purpose of the parameter. |

Usage

If you specify the extends attribute, the data and methods of the parent component are available to any as if they were parts of the current component. If the managerCFC component extends the employeeCFC component, and the employeeCFC component has a getEmployeeName method, you can call this method using the managerCFC, as follows:

```cfm
<cfinvoke component="managerCFC" method="getEmployeeName"  
  returnVariable="managerName" EmployeeID=#EmpID#>
```

This tag requires an end tag.

Example

```cfm
<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>
```
cfccontent

Description
Does either or both of the following:
• Sets the MIME content encoding header for the current page
• Sends the contents of a file from the server as the page output

Note: This tag executes only if it is enabled in the ColdFusion Administrator. For more information, see Configuring and Administering ColdFusion MX.

Category
Data output tags

Syntax
<cfcontent
type = "file_type"
deleteFile = "Yes" or "No"
file = "filename"
reset = "Yes" or "No">starting with a drive letter and a colon, or a forward or backward slash

See also
cfcol, cfheader, cfhttp, cfoutput, cftable
## 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 include:  
  - 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. 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 is sent to the client. |
| reset     | Optional | Yes | The reset and file attributes are mutually exclusive. If you specify a file, this attribute has no effect.  
  - Yes: discards output that precedes call to cfcontent  
  - No: preserves output that precedes call to cfcontent. In this case all output is sent with the specified type.
Usage

To set the character encoding (character set) of generated output, 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 of the data returned in the HTTP response. By default, ColdFusion returns character data using the Unicode UTF-8 format (regardless of the value of an HTML meta tag in the page). You can use the `cfcontent` tag to override the default character encoding of the response. For example, to specify the character encoding of the page output as Japanese EUC, 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.

If you use this tag after the `cfflush` tag on a page, ColdFusion throws an error. The following tag can force most browsers to display a dialog that asks users whether they want to save the contents of the file specified by the `cfcontent` tag as a with the filename specified by the `filename` value.

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

For many file types, such as Excel documents, that Internet Explorer can display directly in the browser, the browser displays the file without asking users whether to save it if you use a `cfheader` tag similar to the following:

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

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.

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

<!--- 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:\CFusionMX\wwwroot\cfdocs\dochome.htm"
deleteFile = "No">```

<!--- This example shows how reset attribute changes text output. --->

```<!--- This example shows how reset attribute changes text output. --->
```
This example shows how reset attribute changes output for text.

reset = "Yes": 123
<cfcontent type = "text/html" reset = "Yes">456</cfcontent>

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

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

Category
Forms tags, Variable manipulation tags

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

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

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>
</tbody>
</table>
| expires   | Optional|         | Expiration of cookie variable.  
- The default: the cookie expires when the user closes the browser, that is, the cookie is "session only".  
- A date or date/time object (for example, 10/09/97)  
- A number of days (for example, 10, or 100)  
- now: deletes cookie from client cookie.txt file (but does not delete the corresponding variable the Cookie scope of the active page).  
- never: The cookie expires in 30 years from the time it was created (effectively never in web years). |
| secure    | Optional|         | If browser does not support Secure Sockets Layer (SSL) security, the cookie is not sent. To use the cookie, the page must be accessed using the https protocol.  
- Yes: variable must be transmitted securely  
- No |
Usage

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 `<cif Cookie.mycookie="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 those 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 sample database ---><cfquery name = "GetAoIUser" dataSource = "cfsnippets">  
SELECT EMail, FromUser, Subject, Posted  
FROM Comments  
</cfquery>  
<html>  
<body>  
<h3>cfcookie Example</h3>  
```
<cfif IsDefined(url.delcookie) is True>
  <cfcookie name = "TimeVisited"
  value = "#Now()#
  expires = "NOW">
</cfif><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>
</cfif>
<!--- If the timeVisited cookie is not set, set a value --->
<cfif IsDefined(Cookie.TimeVisited) is False>
  <cfcookie name = "TimeVisited"
  value = "#Now()#
  expires = "10">
</cfif>
<!--- show the most recent cookie set --->
<cfif IsDefined(Cookie.LastAOLVisitor) is "True">
  <p>The last AOL visitor to view this site was $Cookie.LastAOLVisitor$, on $DateFormat(COOKIE.TimeVisited)$</p>
</cfif><cfelse>
  <p>No AOL Visitors have viewed the site lately.</p>
</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
<cfdefaultcase>

See also
cfcase, cfswitch

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 executed 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 -->
<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>
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"/>
```

See also
cffile

History
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. |
| 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. |
### Usage

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

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

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

- **name**: directory entry name. The entries "." and ".." are not returned.
- **size**: directory entry size
- **type**: file type: File, for a file; Dir, for a directory
- **dateLastModified**: the date that an entry was last modified
- **attributes**: file attributes, if applicable
- **mode**: 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:

```
#mydirectory.name#
#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 2 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 i.e. foo.html, foo.cfm, 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 2 character names.</td>
</tr>
</tbody>
</table>

**Example**

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

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

Category
Debugging tags, Variable manipulation tags

Syntax
```<cfdump
   var = "#variable#"
   expand = "Yes or No"
   label = "text">
See also
   cfcookie, cfparam, cfsavecontent, cschedule, cfset, cfwddx

History
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>Variable to display. Enclose a variable name in pound signs. These kinds of variables yield meaningful cfdump displays: array, CFC, COM object, Java object, simple, query, structure, UDF, wddx, xml</td>
<td></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>
</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:
```<cfif IsXmlDoc(mydoc) is "True"><cfdump var="#mydoc#"></cfif>
```
The tag output is color-coded according to data type.
If a table cell is empty, this tag displays "[empty string"].

**Example**

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

```<cfdump var="$URL">"
**cfelse**

**Description**
Used 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**
```html
<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, cfthrow, cftry**

**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 187**.
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
<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, cfthrow, cftry

Usage
If the value of the expression in this tag is true, and the values of the expressions in the containing cfif 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 cfif tag block. It does not require an end tag.

For more information and an example, see cfif on page 187.
**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**
cfcatch, cfthrow, cftry

**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>
</table>
| type      | Required|         | Type of error that the custom error page handles. The type also determines how ColdFusion handles the error page. For more information, see Chapter 14, “Specifying a custom error page,” in Developing ColdFusion MX Applications.  
  • exception: a exception of the type specified by the exception attribute.  
  • validation: errors recognized by sever-side type validation.  
  • request: any encountered error. |
| template  | Required|         | Relative path to the custom error page. (A ColdFusion page was formerly called a template.) |
| mailTo    | Optional|         | 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. |
| exception | Optional| Any     | 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` on page 353 |
Usage
Use this tag to provide custom error messages for pages in an application. You generally embed this tag in the Application.cfm file. For more information, see Chapter 14, “Handling Errors,” in Developing ColdFusion MX Applications.

To ensure that error pages display successfully, avoid using the `cfencode` tag 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 <code>cfoutput</code> tags.</td>
<td>Can handle specific exception types or display general information for exceptions.</td>
</tr>
<tr>
<td>Request</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 #error.MailTo#.</td>
<td>Use as a backup error handler to other error handling methods, including exception type.</td>
</tr>
<tr>
<td>Validation</td>
<td>Handles data input validation errors that occur when submitting a form that uses hidden form-field validation. Cannot include CFML tags, but you can display values of the error variables by enclosing them in number signs (#), as in #Error.InvalidFields#. You must specify the validation error handler in the Application.cfm file.</td>
<td>Handles hidden form-field style validation errors only.</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>
</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></td>
<td>error.validationFooter</td>
<td>Validation message footer text.</td>
</tr>
</tbody>
</table>
In exception error handling pages, you can access the error variables that are also available to the \texttt{cfcatch} tag. See \texttt{cfcatch} for a description of these variables. To use these variables, prefix them with "\texttt{cferror}".

\textbf{Note:} If \texttt{type = "exception"} you can substitute the prefix \texttt{cferror} for \texttt{Error}; for example, \texttt{cferror.diagnostics}, \texttt{cferror.mailTo}, or \texttt{cferror.dateTime}.

\textbf{Example}

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

     <!--- Example of the page to handle this error --->
     <html>
     <head>
         <title>We're sorry -- An Error Occurred</title>
     </head>
     <body>

     Request and Exception error.diagnostics Detailed error diagnostics from ColdFusion MX.
     error.mailTo E-mail address (same as value in \texttt{cferror.MailTo}).
     error.dateTime Date and time when error occurred.
     error.browser Browser that was running when error occurred.
     error.remoteAddress IP address of remote client.
     error.HTTPReferer Page from which client accessed link to page where error occurred.
     error.template Page executing when error occurred.
     error.generatedContent The content generated by the page up to the point where the error occurred.
     error.queryString URL query string of client's request.

     Exception only error.message Error message associated with the exception.
     error.rootCause Java servlet exception reported by the JVM as the cause of the "root cause" of the exception. This variable is a Java object.
     error.tagContext Array of structures containing information for each tag in the tag stack. The tag stack consists of each tag that is currently open.
     error.type Exception type.

     In exception error handling pages, you can access the error variables that are also available to the \texttt{cfcatch} tag. See \texttt{cfcatch} for a description of these variables. To use these variables, prefix them with "\texttt{cferror}.

     \textbf{Note:} If \texttt{type = "exception"} you can substitute the prefix \texttt{cferror} for \texttt{Error}; for example, \texttt{cferror.diagnostics}, \texttt{cferror.mailTo}, or \texttt{cferror.dateTime}.
\end{verbatim}
<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><b>Your Location:</b> #error.remoteAddress#</li>
<li><b>Your Browser:</b> #error.browser#</li>
<li><b>Date and Time the Error Occurred:</b> #error.dateTime#</li>
<li><b>Page You Came From:</b> #error.HTTPReferer#</li>
<li><b>Message Content:</b></li>
<p>#error.diagnostics#</p></ul>
cfexecute

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

Category
Extensibility tags, Flow-control tags

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

See also
collection, 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>
<tr>
<td>arguments</td>
<td>Optional</td>
<td>Command-line variables passed to application. If specified as string, it is processed as follows:</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Windows: passed to process control subsystem for parsing.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 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.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>If passed as array, it is processed as follows:</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Windows: elements are concatenated into a string of tokens, separated by spaces. Passed to process control subsystem for parsing.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• UNIX: elements are copied into an array of exec() arguments.</td>
<td></td>
</tr>
<tr>
<td>outputFile</td>
<td>Optional</td>
<td>File to which to direct program output. If no outputFile or variable attribute is specified, output is displayed on the page from which it was called.</td>
<td></td>
</tr>
<tr>
<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>
<td></td>
</tr>
</tbody>
</table>
Usage

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

Exceptions

Throws the following exceptions:

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

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

Example

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

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

**Description**

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

**Category**

Debugging tags, Flow-control tags

**Syntax**

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

**See also**

cfabort, cfbreak, cfexecute, 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: reexecutes body of currently executing tag |

**Usage**

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

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

<table>
<thead>
<tr>
<th>Method value</th>
<th>Location of cfexit call</th>
<th>Behavior</th>
</tr>
</thead>
</table>
| exitTag      | Base page               | Terminate processing  
  Execution mode = Start  
  Execution mode = End  |
| exitTemplate | Base page               | Terminate processing  
  Execution mode = Start  
  Execution mode = End  |
| loop         | Base page               | Error  
  Execution mode = Start  
  Execution mode = End  |

**Example**

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

<!--- cfexit can be used within a CFML custom tag, as follows: --->

<!--- Place this code (uncomment the appropriate sections) within the CFUSION/customtags directory --->

<!--- MyCustomTag.cfm --->

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

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

<!--- End MyCustomTag.cfm --->

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

<!--- The call to the custom tag, and then the result: --->

<CF_myCustomTag myValue2 = 4> --->
<cfoutput>$result$</cfoutput> --->

<p>If cfexit is used outside a custom tag, it functions like a cfabort. For example, the text after this message is not processed:</p>

<cfexit> --->

<p>This text is not executed because of the cfexit tag above it.</p>
cfile

Description
Manages interactions with server files.

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

- `cfile action = "append"` on page 113
- `cfile action = "copy"` on page 115
- `cfile action = "delete"` on page 116
- `cfile action = "move"` on page 117
- `cfile action = "read"` on page 119
- `cfile action = "readBinary"` on page 121
- `cfile action = "rename"` on page 122
- `cfile action = "upload"` on page 124
- `cfile action = "write"` on page 127

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 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 an 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

<!--- This shows how to write, read, update, and delete a file using CFFILE --->
This is a view-only example.

<!--- If IsDefined("form.formsubmit") is "Yes" --->
<!--- form has been submitted, now do the action --->
<cfif form.action is "new">
<!--- make a new file --->
<cffile action="Write" 
    file="#GetTempDirectory()#foobar.txt" 
    output="#form.the_text#">
</cfif>

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

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

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

</cfif>

<!--- set some variables --->
<cfparam name="fileExists" default="no">
<cfparam name="readText" default="">
<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> A file exists (foobar.txt, in "#GetTempDirectory()#").
You may add to it, read from it, or delete it.
</p>
</form>
<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
<cffile
    action = "append"
    file = "full_path_name"
    output = "string"
    addNewLine = "Yes" or "No"
    attributes = "file_attributes_list"
    mode = "mode"
    charset = "charset_option" />
```

**See also**
cfdirectory

**History**
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. 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

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

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

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

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

**See also**
cfdirectory

**History**
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 copy.</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>Pathname of a directory or file on web server where the file will be copied. 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, 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>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. Each value must be specified explicitly. For example, if you specify attributes = &quot;readOnly&quot;, all other attributes are overwritten.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• readOnly</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• hidden</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• normal</td>
</tr>
</tbody>
</table>

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

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

**Description**
Deletes a file on the server.

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

**See also**
cfdirectory

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

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

**Syntax**
```cfile
<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 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 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 GetTempDirectory 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>
<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: • 644: Assigns read/write permission to owner; read permission to group and other • 777: Assigns read/write/execute permission to all</td>
</tr>
</tbody>
</table>
The following example moves the keymemo.doc file from the c:\files\upload\ directory to the c:\files\memo\ directory in Windows:

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

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

### Example

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

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

In this example, the destination directory is "memo."
**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. 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>
<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>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:

```cftags
<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:

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

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

Syntax
```xml
<cffile action = "readBinary"
    file = "full_path_name"
    variable = "var_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 a binary fine 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>
<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 creates a variable named aBinaryObj to contain the ColdFusion MX executable:
```xml
<cffile action = "readBinary"
    file = "c:\cfusion\bin\cfserver.exe"
    variable = "aBinaryObj">
```
cffile action = "rename"

**Description**

Renames or moves a file on the server.

**Syntax**

```csharp
<cffile
  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>Type of file manipulation that the tag performs.</td>
<td></td>
</tr>
<tr>
<td>source</td>
<td>Required</td>
<td>Pathname of file to rename. 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>
<td></td>
</tr>
<tr>
<td>destination</td>
<td>Required</td>
<td>Destination file or directory. If not an absolute path, it is relative to the source directory.</td>
<td></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. 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 `attributes = "readOnly"`, all other attributes are overwritten.
- hidden
- normal
- readOnly |

**Usage**

The `rename` action renames or moves 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**

The following example renames the file keymemo.doc to oldmemo.doc:
<cffile action="rename"
    source = "c:\files\memo\keymemo.doc"
    destination = "c:\files\memo\oldmemo.doc">
**cfile action = "upload"**

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

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

**See also**
`cfdirectory`

**History**
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>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 pound 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. 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>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 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 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 the file object variable <code>serverFile</code>.</td>
</tr>
<tr>
<td>accept</td>
<td>Optional</td>
<td></td>
<td>Limits the MIME types to accept. Comma-delimited list. For example, to permit JPG and Microsoft Word file uploads: accept = &quot;image/jpg, application/msword&quot; The browser uses file extension to determine file type.</td>
</tr>
</tbody>
</table>
Usage

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

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

**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>Attribute</th>
<th>Req/Opt</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>attemptedServerFile</td>
<td>Optional</td>
<td></td>
<td>Initial name ColdFusion used when attempting to save a file</td>
</tr>
<tr>
<td>clientDirectory</td>
<td>Directory location of the file uploaded from the client’s system</td>
<td></td>
<td></td>
</tr>
<tr>
<td>clientFile</td>
<td>Name of the file uploaded from the client’s system</td>
<td></td>
<td></td>
</tr>
<tr>
<td>clientFileExt</td>
<td>Extension of the uploaded file on the client system (without a period)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>clientFileName</td>
<td>Name of the uploaded file on the client system (without an extension)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>contentSubType</td>
<td>MIME content subtype of the saved file</td>
<td></td>
<td></td>
</tr>
<tr>
<td>contentType</td>
<td>MIME content type of the saved file</td>
<td></td>
<td></td>
</tr>
<tr>
<td>dateLast Accessed</td>
<td>Date and time the uploaded file was last accessed</td>
<td></td>
<td></td>
</tr>
<tr>
<td>fileExisted</td>
<td>Whether the file already existed with the same path (Yes or No)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>fileSize</td>
<td>Size of the uploaded file</td>
<td></td>
<td></td>
</tr>
<tr>
<td>fileWasAppended</td>
<td>Whether ColdFusion appended uploaded file to a file (Yes or No)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>fileWasOverwritten</td>
<td>Whether ColdFusion overwrote a file (Yes or No)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>fileWasRenamed</td>
<td>Whether uploaded file renamed to avoid a name conflict (Yes or No)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Tip: To refer to parameters, use the cffile prefix: for example, #cffile.fileExisted#.

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

Example

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

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

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

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

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

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

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

```xml
<cffile action = "upload"
    fileField = "fieldname"
    destination = "/tmp/program.exe"
    mode = 777>
```
cfile 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
<cfoutput>
action = "write"
file = "full_path_name"
output = "content"
mode = "permission"
addNewLine = "Yes" or "No"
attributes = "file_attributes_list"
ccharset = "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 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 GetTempDirectory function.</td>
</tr>
<tr>
<td>output</td>
<td>Required</td>
<td></td>
<td>Content of the file to be created.</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 |
| addNewLine| Optional| Yes     | Yes: appends newline character to text written to file |
|           |         |         | No |

See also
cfdirectory
Example

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

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

If the user submitted a form with the following:

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

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

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

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

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

This example appends to the file and sets permissions to read/write (rw) for all:
<cffile action = "append"
    destination = "/home/tomj/testing.txt"
    mode = 666
    output = "Is this a test?">

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

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

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

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

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

**See also**
cfcache, cfheader, cfinclude, cfsetting, cfsilent

**Attributes**

<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>Integer. Flushes output each time this number of bytes becomes available. HTML headers, and data that is already available when the tag is executed, are omitted from the count.</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 cfflush tags on the page send only the output that was generated after the previous flush.

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

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

**Caution:** Because the `cfflush` 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 `cfflush` tag can cause errors or unexpected results: `cfcontent`, `cfcookie`, `cfhtmlhead`, `cflocation`, and `SetLocale`. (These tags and functions normally modify the HTML header, but cannot do so after a `cfflush` tag, because the `cfflush` sends the header.) Using the `cfset` tag to set a cookie anywhere on a page that has a `cfflush` tag does not set the cookie in the browser. Using the `cfflush` 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 `cfflush` 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 `cfflush` tag discards the current buffer. As a result, the `Error.GeneratedContent` variable resulting from a `cferror` tag after a `cfflush` contains any contents of the output buffer that has not been flushed. This content is not sent to the client. The content of the error page displays to the client after the bytes that have been sent.

**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.

```html
<h1>Your Magic numbers</h1>
```
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.

We are sure you will agree it was worth the short wait!
**cfmform**

**Description**

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

**Category**

Forms tags

**Syntax**

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

**See also**

cfapplet, cfgrid, cfinput, cfselect, cfslider, cftextinput, cftree, cftreeitem

**History**

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.

**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>Optional</td>
<td>CFForm_1 [...]</td>
<td>A name for the form.</td>
</tr>
<tr>
<td>action</td>
<td>Optional</td>
<td>Name of ColdFusion page to execute when the form is submitted for processing.</td>
<td></td>
</tr>
<tr>
<td>scriptSrc</td>
<td>Optional</td>
<td>/cfide/scripts/cfform.js</td>
<td>Lets the user control the URL of the script file; useful if you do not keep the file in the /cfide directory.</td>
</tr>
</tbody>
</table>
In addition to the listed attributes, you can use the following HTML attributes in the `cfform` tag. The tag does not use these attributes, but includes them in the `form` tag that it generates and returns to the browser:

- `class`
- `enctype`
- `id`
- `onReset`
- `style`
- `target`

For detailed information on these attributes, refer to the ColdFusion documentation.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Req/Opt</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>preserveData</td>
<td>Optional</td>
<td>No</td>
<td>When the <code>cfform</code> action attribute posts back to the same page as the form, this determines whether to override the control values with the submitted values.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- <code>false</code>: values specified in the control tag attributes are used</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- <code>true</code>: corresponding submitted values are used</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Applies to these controls:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- <code>cfform controls cfforminput, cfslider, cftextarea; overrides value attribute value</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- <code>cfselect controls that are populated from queries. Overrides the selected attribute. See </code>cfselect<code> on page 312.</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- <code>cftree controls: Overrides the cftreeitem expand attribute. If true, expands previously-selected elements. The cftree completePath attribute must be set to Yes.</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- <code>cfgrid controls: has no effect. (This avoids confusion as to whether data has been resubmitted to the database by the control.)</code></td>
</tr>
<tr>
<td>onSubmit</td>
<td>Optional</td>
<td></td>
<td>JavaScript function to execute after input validation. Use for preprocessing data before form is submitted. See Developing ColdFusion MX Applications.</td>
</tr>
<tr>
<td>passThrough</td>
<td>Optional</td>
<td></td>
<td>Passes arbitrary attribute-value pairs to the HTML code that is generated for the tag. You can use either of the following formats:</td>
</tr>
</tbody>
</table>
|             |         |         | passthrough="title="myTitle"
|             |         |         | passthrough='title="mytitle'` |
| codeBase    | Optional | See Description | URL of downloadable JRE plug-in (for Internet Explorer only). Default: /CFIDE/classes/cf-j2re-win.cab |
| archive     | Optional | See Description | URL of downloadable Java classes for ColdFusion controls. Default: /CFIDE/classes/CFJava2.jar |
Usage

This tag requires an end tag.

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

- \texttt{cfinput} Creates and validates an input element (radio button, text box, check box)
- \texttt{cfselect} Creates a drop-down list box
- \texttt{cfslider} Creates a slider control (Java support required)
- \texttt{cftextinput} Creates a text input box (Java support required)
- \texttt{cftree} Creates a tree control (Java support required)
- \texttt{cfgrid} Creates a grid control to display tabular data (Java support required)
- \texttt{cfapplet} Embeds a registered Java applet (Java support required)

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

If you use this tag after the \texttt{cfflush} tag on a page, an error is thrown.

The \texttt{method} attribute is automatically set to \texttt{post}; if you specify a value, it is ignored.

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

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

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

Do not use these suffixes for your field names.

For more information, see Chapter 26, “Retrieving and Formatting Data,” in Developing ColdFusion MX Applications.

Incorporating HTML form tags

The \texttt{cfform} tag lets you incorporate these standard HTML elements:

- Standard form tag attributes and values. The attributes and values are included in the \texttt{form} tag that \texttt{cfform} outputs in the page. For example, you can use \texttt{form} tag attributes like \texttt{target} with \texttt{cfform}. Other pass-through attributes include \texttt{CLASS}, \texttt{ENCTYPE}, \texttt{ID}, \texttt{ONLOAD}, \texttt{ONRESET}, and \texttt{STYLE}.  

134 Chapter 2: ColdFusion Tags
• HTML tags that can ordinarily be put within the HTML form tag. For example, you can use the HTML input tag to create a submit button in a cfform, without the other features of cfinput:
<cfform>
  <input type = "Submit" value = " update... ">
</cfform>

Example
<h3>cfform Example</h3>
<cfif IsDefined("form.oncethrough") is "Yes">
  <cfif IsDefined("form.testVal1") is True>
    <h3>Results of Radio Button Test</h3>
    <cfif form.testVal1 is "Yes">Your radio button answer was yes</cfif>
    <cfif form.testVal1 is "No">Your radio button answer was no</cfif>
  </cfif>
  <cfif IsDefined("form.chkTest2") is True>
    <h3>Results of Checkbox Test</h3>
    Your checkbox answer was yes
    <cfelse>
      Your checkbox answer was no
    </cfif>
  <cfif IsDefined("form.textSample") is True AND form.textSample is not ">">
    <h3>Results of Credit Card Input</h3>
    Your credit card number, #form.textSample#, was valid under the MOD 10 algorithm.
  </cfif>
  <cfif IsDefined("form.sampleSlider") is "True">
    You gave this page a rating of #form.sampleSlider#
  </cfif>
</cfif>
<hr noshade>
</cfif>

<!--- begin by calling the cfform tag --->
<cfform action = "cfform.cfm">
<table>
<tr>
  <td>
    <h4>This example displays radio button input type for cfinput.</h4>
    Yes <cfinput type = "Radio" name = "TestVal1" value = "Yes" checked>
    No <cfinput type = "Radio" name = "TestVal1" value = "No">
  </td>
</tr>
<tr>
  <td>
    <h4>This example displays checkbox input type for cfinput.</h4>
    <cfinput type = "Checkbox" name = "ChkTest2" value = "Yes"/>
  </td>
</tr>
<tr>
  <td>
    <h4>This shows client-side validation for cfinput text boxes.</h4>
    <cfinput type = "Text" name = "TextSample"
      message = "Please enter a Credit Card Number"
      validate = "creditcard" required = "No">
  </td>
</tr>
</table>
</cfform>
This example shows the use of the cfslider tag.

Rate your approval of this example from 1 to 10 by sliding control.

1 <cfslider name = "sampleSlider"
    label = "Sample Slider" range = "1,10"
    message = "Please enter a value from 1 to 10"
    scale = "1" bold = "No"
    italic = "No" refreshlabel = "No"> 10
</td>
</tr>
</table>

<input type = "submit" name = "submit" value = "show me the result">
<input type = "hidden" name = "oncethrough" value = "Yes">
</form>
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 138
- “cftp: Connection: File and directory operations” on page 141
- “cftp action = "listDir"” on page 145

See also
cfhttp, cfldap, cfmail, cfpop

History
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 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

```cfft
<cfftp
    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: create an FTP connection</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• close: terminate an FTP connection</td>
</tr>
<tr>
<td>username</td>
<td>Required if action = &quot;open&quot;</td>
<td></td>
<td>User name to pass in the FTP operation.</td>
</tr>
<tr>
<td>password</td>
<td>Required if action = &quot;open&quot;</td>
<td></td>
<td>Password to log in the user.</td>
</tr>
<tr>
<td>server</td>
<td>Required if action = &quot;open&quot;</td>
<td></td>
<td>FTP server to which to connect; for example, ftp.myserver.com</td>
</tr>
<tr>
<td>timeout</td>
<td>Optional</td>
<td>30</td>
<td>Value in seconds for the timeout of all operations, including 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 username, password, and server attributes, and if no connection exists for them, ColdFusion creates one. Calls to cftp 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>
<tr>
<td>retryCount</td>
<td>Optional</td>
<td>1</td>
<td>Number of retries until failure is reported.</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 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, you can use a Session or Application variable as the connection name. 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

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

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Req/Opt</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
</table>
| stopOnError  | Optional| No      | • Yes: halts processing, displays an appropriate error.
|              |         |         | • No: populates these variables:
|              |         |         | - cfftp.succeeded - Yes or No.
|              |         |         | - cfftp.errorText - Message text
| passive      | Optional| No      | • Yes: enable passive mode
|              |         |         | • No

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 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, you can use a Session or Application variable as the connection name. 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.
stopOnError = "Yes"
name = "ListFiles"
directory = "/"
connection = "my_query">
<cfoutput query = "ListFiles">
#name#</br>
</cfoutput>
<p>Close the connection:
<cfftp action = "close"
connection = "My_query"
stopOnError = "Yes">
<p>Did it succeed? <cfoutput>#cfftp.succeeded#</cfoutput>
cftp: Connection: File and directory operations

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

Syntax
```
<cftp
    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">
```

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 if connection is not cached</td>
<td>FTP operation to perform.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• changedir</td>
<td>• createDir</td>
</tr>
<tr>
<td>username</td>
<td>Required if connection is not cached</td>
<td>User name to pass in the FTP operation.</td>
<td></td>
</tr>
<tr>
<td>password</td>
<td>Required if action = &quot;open&quot;</td>
<td>Password to log in the user.</td>
<td></td>
</tr>
<tr>
<td>name</td>
<td>Required if action = &quot;listDir&quot;</td>
<td>Query name of directory listing.</td>
<td></td>
</tr>
<tr>
<td>server</td>
<td>Required if FTP connection is not cached</td>
<td>FTP server to which to connect; for example, ftp.myserver.com.</td>
<td></td>
</tr>
</tbody>
</table>
Usage

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

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

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

If action = "listDir", a mode column is returned. The column contains an octal string
representation of UNIX permissions; for example, "777."
The `cfftp.returnValue` variable provides the return value for these actions:

- `getCurrentDir`
- `getCurrentURL`
- `existsDir`
- `existsFile`
- `exists`

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

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

### Action (cfftp.ReturnValue variable)

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

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

**Example**

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

```plaintext
<p>Open a connection
<cfftp connection = "myConnection"
    username = "myUserName"
    password = "myUserName@allaire.com"
    server = "ftp.allaire.com"
    action = "open"
    stopOnError = "Yes">
<p>Did it succeed? <cfoutput>#cfftp.succeeded#</cfoutput>
<cfftp connection = "myConnection"
    action = "LISTDIR"
    stopOnError = "Yes"
    name = "ListDirs"
    directory = "/">
<p>FTP Directory Listing:
<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>Close the connection:
<cfftp connection = "myConnection"
    stopOnError = "Yes">
```

```
action = "close"
stopOnError = "Yes">
<p>Did it succeed? <cfoutput>$cftp.succeeded$</cfoutput>
**cftp 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>cftp 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 defined ColdFusion component methods.

Category
Extensibility tags

Syntax
<cffunction
name = "methodName"
returnType = "dataType"
roles = "securityRoles"
access = "methodAccess"
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 cfcomponent tag.</td>
<td></td>
</tr>
<tr>
<td>returnType</td>
<td>Required for a web service; any string formatted according to ColdFusion variable naming conventions.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Optional</td>
<td>any</td>
<td>String; a type name; data type of the function return value:</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></td>
<td>• guid – The argument must be a UUID or GUID of the form xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxxxxxx where each x is a character representing a hexadecimal number (0-9A-F).</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• numeric</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• query</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• string</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• struct</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• uuid – The argument must be a ColdFusion UUID of the form xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxxxxxx where each x is a character representing a hexadecimal number (0-9A-F).</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>• 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.</td>
</tr>
</tbody>
</table>


The **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 **cffunction** tag. For information on using the **cffunction** tag.

The following example shows **cffunction** tag attributes for a simple CFC method that returns a ColdFusion Query object.

```cfml
<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."

For information on using the cffunction tag for ColdFusion components, see Chapter 11, “Building and Using ColdFusion Components,” in Developing ColdFusion MX Applications.

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
<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>
</tbody>
</table>
| Colorlist             | • Pie chart: list of colors to use for each data point  
                        | • Other chart types: ignored |
| Valuelabelfont        | Sets value label text font. If the `Valuelabelfont`, `Itemlabelfont`, and `Legendfont` values differ, ColdFusion uses the last value that you specify in the tag. Aria is not supported; it is mapped to Dialog. |
| Itemlabelfont         | Sets item label text font. If the `Valuelabelfont`, `Itemlabelfont`, and `Legendfont` values differ, ColdFusion uses the last value that you specify in the tag. Aria is not supported; it is mapped to Dialog. |
| Legendfont            | Sets legend text font. If the `Valuelabelfont`, `Itemlabelfont`, and `Legendfont` values differ, ColdFusion uses the last value that you specify in the tag. Aria is not supported; it is mapped to Dialog. |
| ShowLegend            | • above, below, left, right: these options cause the legend to display, but have no effect on its location.  
                        | • none: prevents display of a legend |
| Valuelabelsize        | Sets value label text size. If the `Valuelabelsize` and `Itemlabelsize` values differ, ColdFusion uses the last value that you specify in the tag. |
| Itemlabelsize         | Sets item label text size. |
| Itemlabelorientation  | Ignored. ColdFusion calculates best orientation based on label and graph size. |
| Borderwidth           | • a non-zero number: default-width border, regardless of number value  
                        | • 0: no border |
| Depth                 | • 0: displays graph with two-dimensional appearance  
<pre><code>                    | • any other value: displays graph with three-dimensional appearance |
</code></pre>
<p>| Linewidth             | Ignored                     |</p>
<table>
<thead>
<tr>
<th><strong>cfgraph tag attribute</strong></th>
<th><strong>ColdFusion MX functionality</strong></th>
</tr>
</thead>
</table>
| 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
<cfgrid
  name = "name"
  height = "integer"
  width = "integer"
  autoWidth = "Yes" or "No"
  vSpace = "integer"
  hSpace = "integer"
  align = "value"
  query = "query_name"
  insert = "Yes" or "No"
  delete = "Yes" or "No"
  sort = "Yes" or "No"
  font = "column_font"
  fontSize = "size"
  italic = "Yes" or "No"
  bold = "Yes" or "No"
  textColor = "web color"
  href = "URL"
  hrefKey = "column_name"
  target = "URL_target"
  appendKey = "Yes" or "No"
  highlightHref = "Yes" or "No"
  onValidate = "javascript_function"
  onError = "text"
  gridDataAlign = "position"
  gridLines = "Yes" or "No"
  rowHeight = "pixels"
  rowHeaders = "Yes" or "No"
  rowHeaderAlign = "position"
  rowHeaderFont = "font_name"
  rowHeaderFontSize = "size"
  rowHeaderItalic = "Yes" or "No"
  rowHeaderBold = "Yes" or "No"
  rowHeaderTextColor = "web color"
  colHeaders = "Yes" or "No"
  colHeaderAlign = "position"
  colHeaderFont = "font_name"
  colHeaderFontSize = "size"
  colHeaderItalic = "Yes" or "No"
  colHeaderBold = "Yes" or "No"
  colHeaderTextColor = "web color"
  bgColor = "web color"
  selectColor = "web color"
  selectMode = "mode"
  maxRows = "number"
  notSupported = "text"
  pictureBar = "Yes" or "No"
  insertButton = "text"
deleteButton = "text"
sortAscendingButton = "text"
sortDescendingButton = "text">
</cfgrid>

See also
cfgridcolumn, cfgridrow, cfgridupdate, cfaplet, cfform, cfinput, cfselect,
cfslider, cftextinput, cftree, cftreeitem

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

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 grid element.</td>
</tr>
<tr>
<td>height</td>
<td>Optional</td>
<td>300</td>
<td>Height of grid control, in pixels.</td>
</tr>
<tr>
<td>width</td>
<td>Optional</td>
<td>300</td>
<td>Width of grid control, in pixels.</td>
</tr>
</tbody>
</table>
| autoWidth | Optional; see description | No | • Yes: sets column widths so that all columns display within grid width.  
• No: sets columns to equal widths. User can resize columns. Horizontal scroll bars are not available, because if you specify a column width and set autoWidth = "Yes", ColdFusion sets to this width, if possible. |
| vSpace    | Optional |         | Vertical space above and below grid control, in pixels. |
| hSpace    | Optional |         | Horizontal space to left and right of grid control, in pixels. |
| align     | Optional |         | Alignment of the grid cell contents:  
• Top  
• Left  
• Bottom  
• Baseline  
• Texttop  
• Absbottom  
• Middle  
• Absmiddle  
• Right |
| query     | Optional |         | Name of query associated with grid control. |
| insert    | Optional | No      | • Yes: user can insert row data in grid. Takes effect only if selectmode="edit"  
• No |
| delete    | Optional | No      | • Yes: user can delete row data from grid. Takes effect only if selectmode="edit"  
• No |
<table>
<thead>
<tr>
<th>Attribute</th>
<th>Req/Opt</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sort</td>
<td>Optional</td>
<td>No</td>
<td>The sort button performs simple text sort on column. User can sort columns by clicking column head or by clicking sort buttons. Not valid with selectmode=browse.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Yes: sort buttons display on grid control</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• No</td>
</tr>
<tr>
<td>font</td>
<td>Optional</td>
<td></td>
<td>Font of column data in the grid control.</td>
</tr>
<tr>
<td>fontSize</td>
<td>Optional</td>
<td></td>
<td>Size of text in the grid control, in points.</td>
</tr>
<tr>
<td>italic</td>
<td>Optional</td>
<td>No</td>
<td>• Yes: displays grid control text in italic</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• No</td>
</tr>
<tr>
<td>bold</td>
<td>Optional</td>
<td>No</td>
<td>• Yes: displays grid control text in bold</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• No</td>
</tr>
<tr>
<td>textColor</td>
<td>Optional</td>
<td>Black</td>
<td>Color of text in grid control; hex or text. Hex value or supported named color; see name list in the Usage section. For a hex value, use the form &quot;##xxxxxx&quot;, where x = 0-9 or A-F; use two pound signs or none. For a list of the supported named colors, see cfchart on page 70.</td>
</tr>
<tr>
<td>href</td>
<td>Optional</td>
<td></td>
<td>URL or query column name that contains a URL to hyperlink each grid cell with.</td>
</tr>
<tr>
<td>hrefKey</td>
<td>Optional</td>
<td></td>
<td>The query column to use for the value appended to the href URL of each cell, instead of the cell's value.</td>
</tr>
<tr>
<td>target</td>
<td>Optional</td>
<td></td>
<td>Target of href URL.</td>
</tr>
<tr>
<td>appendKey</td>
<td>Optional</td>
<td>Yes</td>
<td>• Yes: When used with href, appends &quot;GFGRIDKEY=&quot; and the value of the cell to each cell's URL.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• No</td>
</tr>
<tr>
<td>highlightHref</td>
<td>Optional</td>
<td>Yes</td>
<td>• Yes: highlights links associated with a cfgrid with an href attribute value</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• No</td>
</tr>
<tr>
<td>onValidate</td>
<td>Optional</td>
<td></td>
<td>A JavaScript function to validate user input. The form object, input object, and input object value are passed to routine, which returns True if validation succeeds; False otherwise.</td>
</tr>
<tr>
<td>onError</td>
<td>Optional</td>
<td></td>
<td>A JavaScript function to execute if validation fails.</td>
</tr>
<tr>
<td>gridDataAlign</td>
<td>Optional</td>
<td>Left</td>
<td>• Left: left-aligns data within column.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Right: right-aligns data within column.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Center: center-aligns data within column.</td>
</tr>
<tr>
<td>gridLines</td>
<td>Optional</td>
<td>Yes</td>
<td>• Yes: enables row and column rules in grid control</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• No</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>rowHeight</td>
<td>Optional</td>
<td></td>
<td>Minimum row height, in pixels, of grid control. Used with <code>cfgridcolumn type = &quot;Image&quot;</code>; defines space for graphics to display in row.</td>
</tr>
<tr>
<td>rowHeaders</td>
<td>Optional</td>
<td>Yes</td>
<td>• Yes: displays a column of numeric row labels in grid control</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• No</td>
</tr>
<tr>
<td>rowHeaderAlign</td>
<td>Optional</td>
<td>Left</td>
<td>• Left: left-aligns data within row header</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Right: right-aligns data within row header</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Center: center-aligns data within row header</td>
</tr>
<tr>
<td>rowHeaderFont</td>
<td>Optional</td>
<td></td>
<td>Row label font.</td>
</tr>
<tr>
<td>rowHeaderFontSize</td>
<td>Optional</td>
<td></td>
<td>Row label text size in grid control, in points.</td>
</tr>
<tr>
<td>rowHeaderItalic</td>
<td>Optional</td>
<td>No</td>
<td>• Yes: displays row label text in italic</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• No</td>
</tr>
<tr>
<td>rowHeaderBold</td>
<td>Optional</td>
<td>No</td>
<td>• Yes: displays row label text in bold</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• No</td>
</tr>
<tr>
<td>rowHeaderTextColor</td>
<td>Optional</td>
<td>Black</td>
<td>Text color of grid control row headers.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Options: same as for <code>textColor</code> attribute</td>
</tr>
<tr>
<td>colHeaders</td>
<td>Optional; see description</td>
<td>Yes</td>
<td>• Yes: displays column headers in grid control</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• No</td>
</tr>
<tr>
<td>colHeaderAlign</td>
<td>Optional</td>
<td>Left</td>
<td>• Left</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Right</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Center</td>
</tr>
<tr>
<td>colHeaderFont</td>
<td>Optional</td>
<td></td>
<td>Font of column header in grid control.</td>
</tr>
<tr>
<td>colHeaderFontSize</td>
<td>Optional</td>
<td></td>
<td>Size of column header text in grid control, in points.</td>
</tr>
<tr>
<td>colHeaderItalic</td>
<td>Optional</td>
<td>No</td>
<td>• Yes: displays column headers in italics</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• No</td>
</tr>
<tr>
<td>colHeaderBold</td>
<td>Optional</td>
<td>No</td>
<td>• Yes: displays column headers in bold</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• No</td>
</tr>
<tr>
<td>colHeaderTextColor</td>
<td>Optional</td>
<td></td>
<td>Color of grid control column headers.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Options: same as for <code>textColor</code> attribute</td>
</tr>
<tr>
<td>bgColor</td>
<td>Optional</td>
<td></td>
<td>Background color of grid control.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Options: same as for <code>textColor</code> attribute</td>
</tr>
<tr>
<td>selectColor</td>
<td>Optional</td>
<td></td>
<td>Background color for a selected item.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Options: same as for <code>textColor</code> attribute</td>
</tr>
</tbody>
</table>
Usage
You can populate a cfgrid with data from a cfquery. If you do not specify any cfgridcolumn entries, ColdFusion generates a default set of columns, which includes each column in the query. A default header for each column is created by replacing hyphen or underscore characters in the table column name with spaces. The first character, and any character after a space, are changed to uppercase; all other characters are lowercase.

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

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

This tag requires an end tag.

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

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

• Simple data, returned from simple select operations
• Complex data, returned from insert, update and delete operations
Simple selection data (SelectMode = Single, Column, or Row)
The data that form variables return to the cfform's action page contains information about which cells the user selected. In general, ColdFusion makes this data available in the action page, as ColdFusion variables in the Form scope, with the naming convention form.#GridName#.ColumnName#
Each SelectMode returns these form variable(s):
- SelectMode="single"
  form.#GridName#.ColumnName# = "SelectedCellValue"
- SelectMode="column"
  form.#GridName#.ColumnName# = "ValueOfCellRow1, ValueOfCellRow2, ValueOfCellRowN"
- SelectMode="row"
  form.#GridName#.ColumnName# = "ValueOfCellInSelectedRow"
  form.#GridName#.Column2Name# = "ValueOfCellInSelectedRow"
  form.#GridName#.ColumnNName# = "ValueOfCellInSelectedRow"

Complex update data (SelectMode = Edit)
The grid returns a large amount of data, to inform the action page of inserts, updates or deletes that the user made to the grid. In most cases, you can use the cfgridupdate tag to automatically gather the data from the form variables; the tag collects data, writes SQL calls, and updates the data source.
If you cannot use cfgridupdate (if, for example, you must distribute the returned data to more than one data source), you must write code to read form variables. In this mode, ColdFusion creates the following array variables in the Form scope for each cfgrid:
form.#GridName#.ColumnName#
form.#GridName#.original.ColumnName#
form.#GridName#.RowStatus.Action
Each table row that contains an update, insert, or deletion has a parallel entry in each of these arrays. To view all the information for all the changes, you can traverse the arrays, as in this example:
<cfloop index="ColName" list="#ColNameList#">
  <cfif IsDefined("form.#GridName#.ColName")>
    <cfoutput><br>form.#GridName#.ColName:<br></cfoutput>
    <cfset Array_New = evaluate("form.#GridName#.ColName")>
    <cfset Array_Orig = evaluate("form.#GridName#.original.ColName")>
    <cfset Array_Action = evaluate("form.#GridName#.RowStatus.Action")>
    <cfif NOT IsArray(Array_New)>
      <b>The form variable is not an array!</b><br>
    <cfelse>
      <cfset size = ArrayLen(Array_New)>
      <cfoutput>
        Result Array Size is #size#. Contents:<br>
      </cfoutput>
      <cfif size IS 0>
        <b>The array is empty.</b><br>
      <cfelse>
<table BORDER="yes">
<tr>
<th>Loop Index</th>
<th>Action</th>
<th>Old Value</th>
<th>New Value</th>
</tr>
<cfloop index="LoopCount" from="1" to=#size#>
<cfset Val_Orig = Array_Orig[LoopCount]>
<cfset Val_New = Array_New[LoopCount]>
<cfset Val_Action = Array_Action[LoopCount]>
<cfoutput>
<tr>
<td>#LoopCount#</td>
<td>#Val_Action#</td>
<td>#Val_Orig#</td>
<td>#Val_New#</td>
</tr>
</cfoutput>
</cfloop>
</table>
</cfif>

Using the href attribute

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

http://myserver.com?cfgridkey = selection

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

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

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

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

Example

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

<html>
<head>
<title>cfgrid Example</title>
</head>
<body>
<h3>cfgrid Example</h3>
<!--- call the cfform to allow us to use cfgrid controls --->
<cfform action = "cfgrid.cfm">
<!--- We include Course_ID in cfgrid, but do not allow selection or display --->
<!--- cfgridcolumn tags are used to change the parameters involved in displaying each data field in the table --->
<cfgrid name = "FirstGrid" width = "450"
query = "GetCourses" insert = "Yes" delete = "Yes" sort = "Yes"
font = "Tahoma" bold = "No" italic = "No" appendKey = "No" highlightHref = "No"
gridDataAlign = "LEFT" gridLines = "Yes" rowHeaders = "Yes"
rowHeaderAlign = "LEFT" rowHeaderItalic = "No" rowHeaderBold = "No"
colHeaders = "Yes" colHeaderAlign = "LEFT"
colHeaderItalic = "No" colHeaderBold = "No"
selectColor = "Red" selectMode = "EDIT" pictureBar = "No"
insertButton = "To insert" deleteButton = "To delete"
sortAscendingButton = "Sort ASC" sortDescendingButton = "Sort DESC">
<cfgridcolumn name = "Course_ID" dataAlign = "LEFT"
bold = "No" italic = "No" select = "No" display = "No"
headerBold = "No" headerItalic = "No"/>
<cfgridcolumn name = "Dept_ID" header = "Department"
headerAlign = "LEFT" dataAlign = "LEFT"
bold = "Yes" italic = "No" select = "Yes" display = "Yes"
headerBold = "No" headerItalic = "Yes"/>
<cfgridcolumn name = "CorNumber" header = "Course #"
headerAlign = "LEFT" dataAlign = "LEFT"
bold = "No" italic = "No" select = "Yes" display = "Yes"
headerBold = "No" headerItalic = "No"/>
<cfgridcolumn name = "CorName" header = "Name"
headerAlign = "LEFT" dataAlign = "LEFT"
font = "Times" bold = "No"
italic = "No" select = "Yes" display = "Yes" headerBold = "No"
headerItalic = "No"/>
<cfgridcolumn name = "CorLevel" header = "Level"
headerAlign = "LEFT" dataAlign = "LEFT"
bold = "No" italic = "No" select = "Yes" display = "Yes"
headerBold = "No" headerItalic = "No"/>
<cfgridcolumn name = "CorDesc" header = "Description"
headerAlign = "LEFT" dataAlign = "LEFT"
bold = "No" italic = "No" select = "Yes" display = "Yes"
headerBold = "No" headerItalic = "No"/>
</cfgrid>
</cfform>
</body>
</html>
...

cfgridcolumn

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

Category
Forms tags

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

See also
cfgrid, cfgridrow, cfgridupdate, cfapplet, cfform, cfinput, cfselect, cfslider, cftextinput, cftree

History
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

<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 grid column element. If grid uses a query, column name must specify name of a query column.</td>
</tr>
<tr>
<td>header</td>
<td>Optional</td>
<td>Yes</td>
<td>Column header text. Used only if cfgrid colHeaders = &quot;Yes&quot;.</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>width</td>
<td>Optional; see</td>
<td>Column head width</td>
<td>Column width, in pixels.</td>
</tr>
<tr>
<td></td>
<td>Req/Opt</td>
<td></td>
<td>Description</td>
</tr>
<tr>
<td>font</td>
<td>Optional</td>
<td>As specified by cfgrid</td>
<td>Font of data in column.</td>
</tr>
<tr>
<td>fontSize</td>
<td>Optional</td>
<td>As specified by cfgrid</td>
<td>Size of text in column.</td>
</tr>
<tr>
<td>italic</td>
<td>Optional</td>
<td>As specified by cfgrid</td>
<td>• Yes: displays grid control text in italics</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• No</td>
<td></td>
</tr>
<tr>
<td>bold</td>
<td>Optional</td>
<td>As specified by cfgrid</td>
<td>• Yes: displays grid control text in bold</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• No</td>
<td></td>
</tr>
<tr>
<td>textColor</td>
<td>Optional</td>
<td></td>
<td>Color of grid element text in column, or an expression to manipulate color; hex or text. To enter hex value, use the form &quot;##xxxxxx&quot;, where x = 0-9 or A-F; use two pound signs or none. You can enter an expression; for example: textColor= &quot;(C2 LT 0 ? red : pink)&quot; This means: If value in Column 2 is less than 0, display value in red; otherwise, display value in pink. See &quot;Using expressions in textColor and bgColor attributes&quot; on page 164. • Any color, in hex format • Black • Red • Blue • Magenta • Cyan • Orange • Darkgray • Pink • Gray • White • Lightgray • Yellow</td>
</tr>
<tr>
<td>bgColor</td>
<td>Optional</td>
<td></td>
<td>Color of background of grid column, or an expression to manipulate color. See &quot;Using expressions in textColor and bgColor attributes&quot; on page 164. • Options: same as for textColor attribute</td>
</tr>
<tr>
<td>href</td>
<td>Optional</td>
<td></td>
<td>URL or query column name that contains a URL to hyperlink each grid column with.</td>
</tr>
<tr>
<td>hrefKey</td>
<td>Optional</td>
<td></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>
</tr>
<tr>
<td>target</td>
<td>Optional</td>
<td></td>
<td>Frame in which to open link specified in href.</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>select</td>
<td>Optional</td>
<td></td>
<td>• Yes: user can select the column in grid control.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• No: user cannot edit column, regardless of cfgrid insert and delete values. If cfgrid selectMode = &quot;Row&quot; or &quot;Browse&quot;, this value is ignored.</td>
</tr>
<tr>
<td>display</td>
<td>Optional</td>
<td>Yes</td>
<td>• Yes</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• No: hides column</td>
</tr>
<tr>
<td>type</td>
<td>Optional</td>
<td></td>
<td>• image: grid displays image that corresponds to value in column (a built-in ColdFusion image name, or an image in cfide\classes directory or subdirectory referenced with relative URL). If image is larger than column cell, it is clipped to fit. Built-in image names are as follows.</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>• numeric: user can sort grid data numerically</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• boolean: column displays as check box; if cell is editable, user can change checkmark</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>headerFont</td>
<td>Optional</td>
<td>as specified by cfgrid</td>
<td>Column header font</td>
</tr>
<tr>
<td>headerFontSize</td>
<td>Optional</td>
<td>as specified by cfgrid</td>
<td>Column header text size, in pixels</td>
</tr>
<tr>
<td>headerItalic</td>
<td>Optional</td>
<td>as specified by cfgrid</td>
<td>• Yes: displays column header in italics</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• No</td>
</tr>
<tr>
<td>headerBold</td>
<td>Optional</td>
<td>as specified by cfgrid</td>
<td>• Yes: displays header in bold</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• No</td>
</tr>
<tr>
<td>headerTextColor</td>
<td>Optional</td>
<td></td>
<td>Color of grid control column header text.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Options: same as for textColor attribute</td>
</tr>
<tr>
<td>dataAlign</td>
<td>Optional</td>
<td>as specified by cfgrid</td>
<td>Column data alignment:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Left</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Right</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Center</td>
</tr>
<tr>
<td>headerAlign</td>
<td>Optional</td>
<td>as specified by cfgrid</td>
<td>Column header text alignment:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Left</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Right</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Center</td>
</tr>
<tr>
<td>numberFormat</td>
<td>Optional</td>
<td></td>
<td>Format for displaying numeric data in grid. See &quot;numberFormat mask characters&quot; on page 165.</td>
</tr>
</tbody>
</table>
Using expressions in `textColor` and `bgColor` attributes

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

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

The syntax for an expression is as follows:

\[(CX \text{ operator } \text{ string } \text{ ? } \text{ true\_condition } : \text{ false\_condition})\]

The symbol meanings are as follows:

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

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

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

\[
<\text{cfgridcolumn name = "FirstName" textColor = "(CX EQ Pam ? blue : black)"}>\]

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

\[
<\text{cfgridcolumn name = "FirstName" textColor = "(C1 GT 4 ? blue : black)"}>\]
numberFormat mask characters

You can use the following numberFormat attribute mask characters, which correspond to those in the NumberFormat function, to format output in U.S. numeric and currency styles. For more information, see NumberFormat on page 631. (This 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>

Example

For a code example, see cfgrid on page 152.
**cfgridrow**

**Description**

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

**Category**

Forms tags

**Syntax**

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

**See also**

cfgrid, cfgridcolumn, cfgridupdate, cfapplet, cfform, cfinput, cfselect, cfslider, cftextinput, 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>Delimited list of column values. If a value contains a comma, it must be escaped with another comma.</td>
</tr>
</tbody>
</table>

**Usage**

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

```cfc
<cfset MyList1 = "Rome,Athens,Perth,Brasilia">
<cfset MyList2 = "Italy,Greece,Australia,Brazil">
<cfform
    name = "someform" action = "cfform.cfm">
    <cfgrid
        name="GeoGrid" autowidth = "yes" vspace = "4"
        height = "120" font="tahoma" gridlines="yes"
        rowheaders="yes" rowheaderalign="left" colheaders="yes" >
        <cfgridcolumn NAME="City" header="City">
        <cfgridcolumn NAME="Country" header="Country">
        <cfloop index="Counter" from="1" to="#ListLen(MyList1)#">
            <cfgridrow data = "#ListGetAt(MyList1,Counter)#,#ListGetAt(MyList2,(Counter))#">
        </cfloop>
    </cfgrid>
</cfform>
```

**Example**

For a code example, see **cfgrid** on page 152.

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

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

```cfc
<!--- This example shows cfgrid, cfgridcolumn, cfgridrow, cfgridupdate tags --->
<!--- If the gridEntered form field has been tripped, perform the gridupdate on the table specified in the database. Using the default value keyonly=yes lets us change only the information that differs from the previous grid --->
```
<!-- query the database to fill up the grid --->
<cfquery name="GetCourses" datasource="cfsnippets">
   select Course_ID, Dept_ID, CorNumber, CorName, CorLevel, CorDesc
   FROM CourseList
   ORDER by Dept_ID ASC, CorNumber ASC
</cfquery>
</html>
<head>
<title>cfgridrow example</title>
</head>
<body>
<h3>cfgridrow Example</h3>
<I>Try adding a course to the database, and then deleting it.</I>
<!-- call the cfform to allow us to use cfgrid controls --->
<cfform action="cfgridrow.cfm">
<!-- When inserting rows while running under UNIX, you must also specify a value for Course_ID --->
<!-- cfgridcolumn tags are used to change the parameters involved in displaying each data field in the table --->
<cfgrid name="FirstGrid" width="600" query="GetCourses" insert="yes" delete="yes" sort="yes" font="tahoma" bold="no" italic="no" appendkey="no" highlightref="no" griddataalign="left" gridlines="yes" rowheaders="yes" rowheaderalign="left" rowheaderitalic="no" rowheaderbold="no" colheaders="yes" colheaderalign="left" colheaderitalic="no" colheaderbold="no" selectcolor="red" selectmode="edit" picturebar="no" insertbutton="to insert" deletebutton="to delete" sortascendingbutton="sort asc" sortdescendingbutton="sort desc">
<cfgridcolumn name="Dept_ID" header="Department" headeralign="left" dataalign="left" bold="yes" italic="no" select="yes" display="yes" headerbold="no" headeritalic="yes">
<cfgridcolumn name="CorNumber" header="Course ##" headeralign="left" dataalign="left" bold="no" italic="no" select="yes" display="yes" headerbold="no" headeritalic="no">
<cfgridcolumn name="CorName" header="Name" headeralign="left" dataalign="left" font="times" bold="no" italic="no" select="yes" display="yes" headerbold="no" headeritalic="no">
<cfgridcolumn name="CorLevel" header="Level" headeralign="left" dataalign="left" bold="no" italic="no" select="yes" display="yes" headerbold="no" headeritalic="no">
<cfgridcolumn name="CorDesc" header="Description" headeralign="left" dataalign="left" bold="no" italic="no" select="yes" display="yes" headerbold="no" headeritalic="no">
<cfgridcolumn name="Course_ID" header="Course ID (Do Not Specify on NT)" dataalign="left" bold="no" italic="no" select="yes" display="yes" headerbold="no" headeritalic="no">
</cfgrid>
</cfform>
<h3>Example Two</h3>
This grid shows how the same grid can be built using cfgridrow with cfloop (i.e., defining query external to cfgrid, rather than within cfgrid).

--- cfgridcolumn is used to define container columns. cfgridrow is used to define the data put into those containers ---

```cfgrid
<cfgrid name="SecondGrid" width="600" insert="no"
delete="no" sort="yes" bold="no" italic="no"
appendkey="no" highlightref="no" griddataalign="left" gridlines="yes"
rowheaders="no" rowheaderalign="left" rowheaderitalic="no"
rowheaderbold="no" colheaders="yes" colheaderalign="left"
colheaderitalic="no" colheaderbold="no" selectmode="browse"
picturebar="yes">
<cfgridcolumn name="Course_ID" dataalign="left" bold="no" italic="no"
select="no" display="no" headerbold="no" headeritalic="no">
<cfgridcolumn name="Dept_ID" header="Department" headeralign="left"
dataalign="left" bold="yes" italic="no" select="yes"
display="yes" headerbold="no" headeritalic="yes">
<cfgridcolumn name="CorNumber" header="Course ##" headeralign="left"
dataalign="left" bold="no" italic="no" select="yes"
display="yes" headerbold="no" headeritalic="no">
<cfgridcolumn name="CorName" header="Name" headeralign="left"
dataalign="left" font="times" bold="no" italic="no"
select="yes" display="yes" headerbold="no" headeritalic="no">
<cfgridcolumn name="CorLevel" header="level" headeralign="left"
dataalign="left" bold="no" italic="no" select="yes"
display="yes" headerbold="no" headeritalic="no">
<cfgridcolumn name="CorDesc" header="Description" headeralign="LEFT"
dataalign="left" bold="no" italic="no" select="yes" display="yes"
headerbold="no" headeritalic="no">
<cfloop query="GetCourses">
<cfgridrow data="#Course_ID#, #Dept_ID#, #CorNumber#, #CorName#, #CorLevel#, #CorDesc#"></cfloop>
</cfgrid>
</cfform>
</body>
</html>
```
**cfgridupdate**

**Description**

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

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

**Category**

Forms tags

**Syntax**

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

**See also**

`cfgrid`, `cfgridcolumn`, `cfgridrow`, `cfapplet`, `cfform`, `cfinput`, `cfselect`, `cfslider`, `cfinput`, `cfree`

**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 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 data source for the update action.</td>
</tr>
<tr>
<td>tableName</td>
<td>Required</td>
<td></td>
<td>Name of table to update. For ORACLE drivers, entry must be upper-case. For Sybase driver, entry is case-sensitive; must be same case as used when table was created.</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

<!--- This example shows the cfgridupdate tag--->

<!--- If the gridEntered form field has been tripped, perform
the gridupdate on the table specified in the database.
Using the default value keyonly = yes lets us change only the
information that differs from the previous grid --->

<cfif IsDefined("form.gridEntered") is True>
<cfgridupdate grid = "FirstGrid" dataSource = "cfsnippets"
tableName = "CourseList" keyOnly = "Yes">
</cfif>

...</cfif>
**cfheader**

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

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

**Syntax**
```html
<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>
<tr>
<td></td>
<td></td>
<td></td>
<td>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
If you use this tag after the `cfflush` tag on a page, an error is thrown.

### Example

```html
<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 = "#Now()#"/>
```
cfhtmlhead

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

Category
Page processing tags

Syntax
<cfhtmlhead
text = "text">

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

Attributes

<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>Text to add to the &lt;head&gt; area of an HTML page.</td>
<td></td>
</tr>
</tbody>
</table>

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

Example
<body>
<!--- This example shows the use of cfhtmlhead --->
<cfhtmlhead
text="<meta name="Description" content="This is an example of a generated header"">">
<p>cfhtmlhead writes the text specified in the text attribute to the &lt;HEAD&gt; section of a generated HTML page. cfhtmlhead can be useful for embedding JavaScript code, or placing other HTML tags such as META, LINK, TITLE, or BASE in an HTML header.
<p>View the source of this frame to see that the title of the page is generated by the cfhtmlhead tag.
</body>
cfhttp

Description
Generates an HTTP request and handles the response from the server.

Category
Forms tags, 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"
  cfhttpparam tags [optional for some methods]
/></cfhttp>```

See also
`cfhttpparam, GetHttpRequestData, cfftp, cfldap, cfmail, cfpop, SetEncoding`

History
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 `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 <em>port</em> attribute value. The <em>cfhttpparam</em> tag URL attribute appends query string attribute-value pairs to the URL.</td>
</tr>
<tr>
<td>port</td>
<td>Opt</td>
<td>80 for http 413 for https</td>
<td>Port number on the server to which to send the request. A port value in the <em>url</em> attribute overrides this value.</td>
</tr>
<tr>
<td>method</td>
<td>Opt</td>
<td>GET</td>
<td>• GET Requests information from the server. Any data that the server requires to identify the requested information must be in the URL or in <em>cfhttp type=&quot;URL&quot;</em> tags. • POST Sends information to the server for processing. Requires one or more <em>cfhttpparam</em> 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.</td>
</tr>
<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>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>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>A username. May be required by server.</td>
</tr>
<tr>
<td>password</td>
<td>Opt</td>
<td></td>
<td>A password. May be required by server.</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>
<tr>
<td>charset</td>
<td>Opt</td>
<td></td>
<td>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: <a href="http://www.w3.org/International/O-charset.html">www.w3.org/International/O-charset.html</a>.</td>
</tr>
<tr>
<td>resolveURL</td>
<td>Opt</td>
<td>No</td>
<td>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</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>
| throwOnError       | Opt     | No      | • 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. |
| redirect           | Opt     | Yes     | If the response header includes a Location field, determines 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"`. |
| timeout            | Opt     |         | Value, in seconds of 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, convert it to a ColdFusion object.  
• Auto  If ColdFusion does not recognize the response body type as text, convert it to ColdFusion Binary type data.  
• Yes  Always convert 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 (Sends as multipart only if request includes File type data.)</td>
<td>Tells ColdFusion to send all data specified by <code>cfhttpparam type=&quot;formField</code> 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-8859-1 or ASCII.) Some http parsers, including the one used by previous versions of ColdFusion, ignore the multipart form field character encoding description.</td>
</tr>
<tr>
<td>path</td>
<td>Required if file is specified</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 if path is specified and not a GET method</td>
<td>See Description</td>
<td>Name of file in which to store the response body. For a GET operation, defaults to 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 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 DELETE or OPTIONS.
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. Column names must start with a letter. The remaining characters can be letters, numbers, or underscores (_). 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></td>
<td></td>
<td></td>
<td>• Yes  processes the first row as column heads. If you specify a columns attribute, ColdFusion ignores the first row of the file.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• No   processes the first row as data. If you do not specify a columns attribute, ColdFusion generates column names by appending numbers to the word &quot;column&quot;; for example, &quot;column_1&quot;.</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.

### Variables returned by a `cfhttp` get operation

The `cfhttp` tag returns the following variables:

<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</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 <code>cfhttp.responseHeader</code> 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 <code>cfhttp.responseHeader</code> structure as a dynamic array; for example, <code>#cfhttp.responseHeader[fieldVariable]#</code>.</td>
</tr>
<tr>
<td>cfhttp.statusCode</td>
<td>The HTTP status_code header value followed by the HTTP Explanation header value, for example &quot;200 OK&quot;.</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 from 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:

```cfm
<cfhttp method="Get"
    url="127.0.0.1:8500/tests/escapetest.txt"
    name="onerow">
<cfdump var="#onerow#"><br>
</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 underscores (_). 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. However, Basic Authentication does not work if your web server has Windows NT Challenge/Response (Microsoft IIS) enabled.
- 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 with proxy servers. If a proxy server requires authentication, it must allow Basic Authentication.

Example

```cfm
<!--- 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 --->

<!--- 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"/>

<!--- 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.xmlChildren>
<cfset numresources = ArrayLen(resources)>

<!--- Loop through all of the 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
Forms tags, 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>
<tbody>
<tr>
<td>type</td>
<td>Required</td>
<td></td>
<td>Information type:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Header: The parameter specifies an HTTP header. ColdFusion does not URL encode the header.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• CGI: Specifies an HTTP header. ColdFusion URL encodes the header by default.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Body: Specifies the body of the HTTP request. ColdFusion does not URL encode the body contents.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• 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.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• File: Tells ColdFusion to send the contents of the specified file. ColdFusion does not URL encode the file contents</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• URL: Specifies a URL query string name-value pair to append to the cfhttp url attribute. ColdFusion URL encodes the query string.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• FormField: Specifies a form field to send. ColdFusion URL encodes the Form field by default.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Cookie: Specifies a cookie to send as an HTTP header. ColdFusion URL encodes the cookie.</td>
</tr>
<tr>
<td>name</td>
<td>Required, Optional for Body and XML types</td>
<td></td>
<td>Variable name for data that is passed. Ignored for Body and XML types. For File type, specifies the filename to send in the request.</td>
</tr>
<tr>
<td>value</td>
<td>Required, Optional (and ignored) for File type</td>
<td></td>
<td>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.</td>
</tr>
<tr>
<td>file</td>
<td>Required only if type=&quot;file&quot;</td>
<td></td>
<td>Applies to File type; ignored for all other types. The absolute path to the file that is sent in the request body.</td>
</tr>
<tr>
<td>encoded</td>
<td>Optional</td>
<td>Yes</td>
<td>Applies to FormField and CGI types; ignored for all other types. Specifies whether to URLEncode the form field or header.</td>
</tr>
<tr>
<td>mimeType</td>
<td>Optional</td>
<td></td>
<td>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.</td>
</tr>
</tbody>
</table>

## 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 731.

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="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 cfhttpparamexample.cfm page that receives and processes the Post request. It's 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#
</cfoutput>
CGI variable: #CGI.cgi_test#

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

**See also**
`cfelse, cfelseif, cfabort, cfbreak, cfexecute, cfexit, 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:

```xml
<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:

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

This tag requires an end tag.

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

```xml
<!--- This example shows the interaction of cfif, cfelse, and cfelseif --->
<!--- first, perform a query to get some data ----->
<cfquery name="getCenters" datasource="cfsnippets">
    SELECT Center_ID, Name, Address1, Address2, City, State, Country, PostalCode, Phone, Contact
    FROM Centers
    ORDER by City, State, Name
</cfquery>
<p>CIF 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>
</p>--- 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#
</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 OR 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. Notice how a nested CFIF is used to specify the location of the featured site (San Diego or San Diego).
</p>
<!----- 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:
</p>
<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></cfoutput>
</cfoutput>
<hr>
<p>Finally, we can use CFELSEIF to cycle through a number of conditions and produce varying output. Note that you can use CFCASE and CFSWITCH for a more elegant representation of this behavior.
</p>
<!------- use CFIF in conjunction with CFELSEIF to specify more than one branch in a conditional situation --->
<cfoutput query="getCenters">
<cfif Trim(City) is "San Diego" OR Trim(City) is "Santa Ana">
<br><i>#Name#, #City#</i> (this one is in San Diego)
</cfelseif>
<cfelseif Trim(City) is "San Francisco">
<br><i>#Name#, #City#</i> (this one is in San Francisco)
</cfelseif>
<cfelseif Trim(City) is "Suisun">
<br><i>#Name#, #City#</i> (this one is in Suisun)
</cfelseif>
<cfelseif #Name#>
<br><b>Not in a city we track</b></cfelseif></cfoutput>
cfimpersonate

Description
This tag is obsolete. Use the newer security tools; see “Authentication functions” on page 367 and Chapter 16, “Securing Applications,” in Developing ColdFusion MX Applications.

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
<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, follow these steps:

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#@<br>`

   `</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 250. (A ColdFusion page was formerly sometimes called a ColdFusion template or a template.)

Category
Flow-control tags, Page processing tags

Syntax
```html
<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></td>
<td>A logical path to a ColdFusion page.</td>
</tr>
</tbody>
</table>

Usage
ColdFusion searches for included files in the following sequence:

1. In the directory of the current page
2. In directories mapped in the ColdFusion Administrator for the included file

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.

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

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

**Description**

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

A collection must exist before it can be populated.

A collection can be indexed in the following ways:

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

For more information, see Chapter 24, “Building a Search Interface,” in Developing ColdFusion MX Applications.

**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"  
    URLpath = "URL"  
    extensions = "file_extensions"  
    query = "query_name"  
    recurse = "Yes" or "No"  
    language = "language">
</cfindex>
```

**See also**

cfcollection, cfexecute, cfobject, cfreport, cfsearch, cfwddx

**History**

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>
</table>
| collection  | Required|         | • Name of a collection that is registered by ColdFusion; for example, "personnel"
|             |         |         | • Name and absolute path of a collection that is not registered by ColdFusion; for example: "e:\collections\personnel"
| action      | Depends on collection attribute value; see Usage section |         | • update: updates a collection and adds key to the index. Do not use the cflock tag with this option.
|             |         |         | • delete: deletes data in the entities specified by the type attribute
|             |         |         | • purge: deletes all keys from a collection
|             |         |         | • refresh: purges all keys from a collection, then updates it.
| type        | Optional| custom, if query attribute is specified. file, otherwise. | • file: using the key attribute value of the query result as input, applies action value to filenames or filepaths.
|             |         |         | • path: using the key attribute value of the query result as input, applies action to filenames or filepaths that pass the extensions filter
|             |         |         | • custom: If action = "update" or "delete": applies action to custom entities in query results.
| title       | Optional|         | • Title for collection
|             |         |         | • Query column name for type and a valid query name Permits searching collections by title or displaying a separate title from the key
| key         | Depends on action attribute value; see Usage section | (empty string) | • Absolute path and filename, if type = "file"
|             |         |         | • Absolute path, if type = "path"
|             |         |         | • A query column name (typically, the primary key column name), if type = "custom"
|             |         |         | • A query column name, if type = any other value
|             |         |         | This attribute is required for the actions listed, unless you intend for its value to be an empty string.
| body        | Required if type = "custom"; see Usage section |         | • ASCII text to index
|             |         |         | • Query column name(s), if name is specified in query
|             |         |         | You can specify columns in a delimited list. For example: "emp_name, dept_name, location"
| custom1     | Optional|         | Custom field in which you can store data during an indexing operation. Specify a query column name for type, and a query name.
| custom2     | Optional|         | Usage is the same as for custom1.
| URLpath     | Optional|         | If type = "file" or "path", specifies the URL path. When the collection is searched with cfsearch, this pathname is prefixed to filenames and returned as the url attribute.
| extensions  | Optional| HTM, HTML, CFM, DBM, DBML | Delimited list of file extensions that ColdFusion uses to index files, if type = "Path".
|             |         | "*.*" returns files with no extension.
|             |         | For example: the following code returns files with a listed extension or no extension: extensions = ".htm, .html, .cfm, .cfml, ".*."
This tag populates Verity search engine collections with metadata from the following sources:

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

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

<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 against which collection is generated</td>
</tr>
</tbody>
</table>
| recurse         | Optional | No      | • Yes: if type = "path", directories below the path specified in key are included in indexing operation
| language        | Optional | English | For options, see cfcollection on page 80. Requires the appropriate (European or Asian) Verity Locales language pack. |

**Usage**

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

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

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

<table>
<thead>
<tr>
<th>Specifying this attribute is required, optional or unnecessary (blank):</th>
<th>For this action attribute value:</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>collection</td>
<td>purge</td>
<td>delete</td>
<td>update or refresh</td>
<td></td>
</tr>
<tr>
<td>type</td>
<td>Required</td>
<td>Required</td>
<td>Required</td>
<td></td>
</tr>
<tr>
<td>title</td>
<td>Optional</td>
<td>If type = &quot;file&quot;,&quot;path&quot;, or &quot;custom&quot;: Optional.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>key</td>
<td>Required</td>
<td>If type = &quot;file&quot;,&quot;path&quot;, or &quot;custom&quot;: Required. Otherwise: unnecessary.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>body</td>
<td>Required</td>
<td>If type = &quot;custom&quot;: Required. Otherwise: unnecessary.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>custom1</td>
<td>If type = &quot;file&quot;,&quot;path&quot;, or &quot;custom&quot;: Optional. Otherwise: unnecessary.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>custom2</td>
<td>If type = &quot;file&quot; or &quot;path&quot;: Optional. Otherwise: unnecessary.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>URLPath</td>
<td>If type = &quot;file&quot; or &quot;path&quot;: Optional. Otherwise: unnecessary.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>extensions</td>
<td>If type = &quot;path&quot;: Optional. Otherwise: unnecessary.</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
For all action values of this tag except `update`, use the `cflock` tag to protect the collection during tag execution.

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

### Example

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

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

<!--- for ACTION=UPDATE, #3 (TYPE=PATH) (extensions .htm, .html,.cfm,.cfml) --->
<cfindex collection="snippets"
    action="update"
    type="path"
    key="c:\inetpub\wwwroot\cfdocs\snippets"
    recurse="true"
    language="en">
```

<table>
<thead>
<tr>
<th>Specifying this attribute is required, optional or unnecessary (blank):</th>
<th>For this action attribute value:</th>
</tr>
</thead>
<tbody>
<tr>
<td>query</td>
<td>If type = &quot;file&quot;, &quot;path&quot;: Optional. If type = &quot;custom&quot;: Required. If type = &quot;file&quot;, &quot;path&quot;: Optional. If type = &quot;custom&quot;: Required.</td>
</tr>
<tr>
<td>recurse</td>
<td>If type = &quot;path&quot;: Optional. Otherwise: unnecessary.</td>
</tr>
<tr>
<td>language</td>
<td>If type = &quot;file&quot;, &quot;path&quot;, or &quot;custom&quot;: Optional. If type = &quot;file&quot;, &quot;path&quot;, or &quot;custom&quot;: Optional.</td>
</tr>
</tbody>
</table>

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

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

### Example

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

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

<!--- for ACTION=UPDATE, #3 (TYPE=PATH) (extensions .htm, .html,.cfm,.cfml) --->
<cfindex collection="snippets"
    action="update"
    type="path"
    key="c:\inetpub\wwwroot\cfdocs\snippets"
    recurse="true"
    language="en">
```
<!-- for ACTION=UPDATE, #4 (TYPE=PATH) (extensions are files with no extension) --->
<cfindex
collection="snippets"
action="update"
type="path"
key="c:\inetpub\wwwroot\cfdocs\snippets"
urlpath="http://localhost/cfdocs/snippets"
custom1="custom1"
custom2="custom2"
recurse="no"
extensions=".htm, .html, .cfm, .cfml" >

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

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

<!-- for ACTION=UPDATE, #7 (TYPE=CUSTOM) --->
<cfquery name="book"
datasource="book"
select * from book
</cfquery>
<cfindex
collection="custom_book"
action="update"
type="custom"
body="description"
key="bookid"
<!--- for ACTION=REFRESH----------------------------------------------->
<!--- ACTION=REFRESH, #1 (TYPE=FILE) ---->
<cflock name="verity" timeout="60">
<cfindex
collection="snippets"
action="Refresh"
type="file"
key="c:\inetpub\wwwroot\cfdocs\snippets\abs.cfm"
urlpath="http://localhost/"
custom1="custom1"
custom2="custom2" />
</cflock>

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

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

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

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

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

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

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

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

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

Category
Forms tags

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

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

History
ColdFusion MX 6.1: Changed the requirements for the validate = "creditcard" option: it requires that the text entry have 13-16 digits.

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

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


<table>
<thead>
<tr>
<th>Attribute</th>
<th>Req/Opt</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
</table>
| validate    | Optional  |         | Verifies a value’s format:  
• date: US date mm/dd/yyyy  
• eurodate: European date dd/mm/yyyy  
• time: time hh:mm:ss  
• float: floating point entry  
• integer: integer entry  
• telephone: telephone ###-###-####. Separator: hyphen or blank. Area code and exchange must begin with a digit 1–9.  
• zipcode: (U.S. formats only) 5-digit #### or 9-digit ####-####. Separator: hyphen or blank.  
• creditcard: must be a 13 to 16 digit integer after stripping blanks and dashes; uses the mod10 algorithm.  
• social_security_number: ###-##-####. Separator: hyphen or blank.  
• regular_expression: matches input against regular expression specified by the pattern attribute. |
| onValidate   | Optional  |         | Custom JavaScript function to validate user input. The form object, input object, and input object values are passed to the routine, which should return True if validation succeeds, and False otherwise. If used, the validate attribute is ignored. |
| pattern      | Required if validate = "regular_expression" |         | JavaScript regular expression pattern to validate input. Omit leading and trailing slashes. For examples and syntax, see Chapter 27, "Building Dynamic Forms," in Developing ColdFusion MX Applications. |
| message      | Optional  |         | Message text to display if validation fails. |
| onError      | Optional  |         | Custom JavaScript function to execute if validation fails. |
| size         | Optional  |         | Size of input control. Ignored, if type = "radio" or "checkbox". |
| maxLength    | Optional  |         | Maximum length of text entered, if type = "Text" or "password". |
| checked      | Optional  |         | Selects a control. No value is required.  
Applies if type = "radio" or "checkbox".  
Optional: you can enter the following values:  
• true (equivalent to checked)  
• false (equivalent to omitting the attribute) |
| passThrough  | Optional  |         | Passes one or more arbitrary attribute-value pairs to the HTML code that is generated for the tag. You can use either of the following formats to include the quotation marks around the attribute value:  
passthrough="ID="myID""  
passthrough='ID="myID"'  
The second format, which surrounds all the attribute-value pairs to be passed through in single quotation marks is clearer, particularly when you pass multiple HTML attributes. |

In addition to the listed attributes, you can use the following HTML attributes in the cfform tag without using the passThrough attribute. The tag does not use these attributes, but includes them in the HTML of the form tag that it generates and returns to the browser:
Usage

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

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

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

If you specify a value in quotation marks, you must escape them; for example,

```
passThrough = "readonly = " "YES " "
```

For more information, see cfform on page 132. For information on using JavaScript regular expressions with this tag, see Chapter 27, “Building Dynamic Forms,” in Developing ColdFusion MX Applications.

Example

```xml
<!--- this example shows the use of cfinput within a cfform to ensure simple validation of text items --->
<cfform action = "cfinput.cfm">
<!--- phone number validation --->
Phone Number Validation (enter a properly formatted phone number): <br>
<cfinput
type = "Text" name = "MyPhone"
message = "Enter telephone number, formatted xxx-xxx-xxxx (e.g. 617-761-2000)"
validate = "telephone" required = "Yes">
<font size = -1 color = red>Required</font>
<!--- zip code validation --->
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 --->
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 --->
Date Validation (enter a properly formatted date):<br>
<cfinput
type = "Text" name = "MyDate"
```
message = "Enter a correctly formatted date (dd/mm/yy)"
validate = "date" required = "No">
<input
type = "Submit" name = ""
value = "send my information">
</cfform>
**cfinser**

**Description**
Inserts records in data sources from data in a ColdFusion form or form Scope.

**Category**
Database manipulation tags

**Syntax**
```
<cfinsert
dataSource = "ds_name"
tableName = "tbl_name"
tableOwner = "owner"
tableQualifier = "tbl_qualifier"
username = "username"
password = "password"
formFields = "formfield1, formfield2, ...">
```

**See also**
cfprocparam, cfprocresult, cfquery, cfqueryparam, cfstoredproc, cftransaction, cfupdate

**History**
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>Data source; contains table.</td>
</tr>
<tr>
<td>tableName</td>
<td>Required</td>
<td></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>
</tr>
<tr>
<td>tableOwner</td>
<td>Optional</td>
<td></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>
</tr>
<tr>
<td>tableQualifier</td>
<td>Optional</td>
<td></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>
</tr>
<tr>
<td>username</td>
<td>Optional</td>
<td></td>
<td>Overrides username specified in ODBC setup.</td>
</tr>
<tr>
<td>password</td>
<td>Optional</td>
<td></td>
<td>Overrides password specified in ODBC setup.</td>
</tr>
<tr>
<td>formFields</td>
<td>Optional</td>
<td>(all on form, except keys)</td>
<td>Comma-delimited list of form fields to insert. If not specified, all fields in the form are included. If a form field is not matched by a column name in the database, ColdFusion throws an error. The database table key field must be present in the form. It may be hidden.</td>
</tr>
</tbody>
</table>
Example

<!--- This shows how to use cfinsert instead of cfquery to put data in a datasource. --->
<!--- if form.POSTED exists, we insert new record, so begin cfinsert tag --->
<cfif IsDefined ("form.posted")>
  <cfinsert dataSource = "cfsnippets" tableName = "Comments" formFields = "Email,FromUser,Subject,MessText,Posted">
    <h3>Your record was added to the database.</h3>
  </cfif>

<cfif IsDefined ("form.posted")>
  <cfif Server.OS.Name IS "Windows NT">
    <cfinsert datasource="cfsnippets" tablename="Comments" formfields="EMail,FromUser,Subject,MessText,Posted">
  </cfifelse>
  <cfinsert datasource="cfsnippets" tablename="Comments" formfields="CommentID,EMail,FromUser,Subject,MessText,Posted">
  </cfif>
  <h3>Your record was added to the database.</h3>
</cfif>

<!--- use a query to show the existing state of the database --->
<cfquery name = "GetComments" dataSource = "cfsnippets">
  SELECT CommentID, EMail, FromUser, Subject, CommtType, MessText, Posted, Processed
  FROM Comments
</cfquery>

<html>
<head></head>
<h3>cfinsert Example</h3>
<p>First, show a list of the comments in the cfsnippets datasource.
</p>
<!--- show all the comments in the db --->
<table>
  <tr>
    <td>From User</td><td>Subject</td><td>Comment Type</td><td>Message</td><td>Date Posted</td>
  </tr>
  <cfoutput query = "GetComments">
    <tr>
      <td valign = top><a href = "mailto:#Email#">#FromUser#</a></td>
      <td valign = top>#Subject#</td>
      <td valign = top>#CommtType#</td>
      <td valign = top><font size = "-2">#Left(MessText, 125)#</font></td>
      <td valign = top>#Posted#</td>
    </tr>
  </cfoutput>
</table>
<p>Next, we'll offer the opportunity to enter a comment:
</p>
<!--- make a form for input --->
<form action = "cfinsert.cfm" method = "post">
  <pre>
  Email: <input type = "Text" name = "email">
  From: <input type = "Text" name = "fromUser">
  Subject: <input type = "Text" name = "subject">
  Message:<textarea name = "MessText" COLS = "40" ROWS = "6"></textarea>
  Date Posted: <cfoutput>#DateFormat(Now())#</cfoutput>
</form>
</cfoutput>
</table>
<p>Next, we'll offer the opportunity to enter a comment:
</p>
<!--- dynamically determine today's date --->
<pre>&lt;input type = "hidden"
    name = "posted" value = "&lt;cfoutput>#Now()#/</cfoutput>">&lt;/pre&gt;
&lt;input type = "Submit"
    name = "" value = "insert my comment"&gt;
&lt;/form&gt;
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"
inputParam1 = "value1"
inputParam2 = "value2"
...
returnVariable = "var_name"
...

OR

<!--- Syntax 4A - this syntax invokes a component. This syntax shows instantiation with the cfobject tag. This cfinvoke syntax applies to instantiating a component with the cfobject tag and to instantiating a component with the createobject function. --->
<cfobject
    component = "component name"
    name = "mystringname for instantiated object"
>
<cfinvoke
    <!--- value is object name, within pound signs --->
    component = ",#mystringname for instantiated component#"
    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
    <!--- value is object name, within pound signs --->
    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"
    proxyPassword = "string user password for proxy server"
    >

See also
cfargument, cfcomponent, cffunction, cfinvokeargument, cfobject, cfproperty, cfreturn

History

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 Usage section</td>
<td>String or component object; a reference to a component, or component to instantiate.</td>
<td></td>
</tr>
<tr>
<td>method</td>
<td>See Usage section</td>
<td>Name of a method. For a web service, the name of an operation.</td>
<td></td>
</tr>
<tr>
<td>returnVariable</td>
<td>Optional</td>
<td>Name of a variable for the invocation result.</td>
<td></td>
</tr>
<tr>
<td>argumentCollection</td>
<td>Optional</td>
<td>Name of a structure; associative array of arguments to pass to the method.</td>
<td></td>
</tr>
<tr>
<td>username</td>
<td>Optional</td>
<td>Overrides username specified in Administrator &gt; Web Services.</td>
<td></td>
</tr>
<tr>
<td>password</td>
<td>Optional</td>
<td>Overrides password specified in Administrator &gt; Web Services.</td>
<td></td>
</tr>
<tr>
<td>webservice</td>
<td>Required</td>
<td>The URL of the WSDL file for the web service.</td>
<td></td>
</tr>
<tr>
<td>timeout</td>
<td>Optional</td>
<td>The timeout for the web service request, in seconds</td>
<td></td>
</tr>
<tr>
<td>proxyServer</td>
<td>Optional</td>
<td>http.proxyHost system property, if any.</td>
<td></td>
</tr>
<tr>
<td>proxyPort</td>
<td>Optional</td>
<td>http.proxyPort system property, if any.</td>
<td></td>
</tr>
<tr>
<td>proxyUser</td>
<td>Optional</td>
<td>http.proxyUser system property, if any.</td>
<td></td>
</tr>
<tr>
<td>proxyPassword</td>
<td>Optional</td>
<td>http.proxyPassword system property, if any.</td>
<td></td>
</tr>
</tbody>
</table>

Note: If you do not specify any 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>This attribute is required, optional, ignored, or invalid:</th>
<th>Syntax 1</th>
<th>Syntax 2</th>
<th>Syntax 3</th>
<th>Syntax 4A</th>
<th>Syntax 4B</th>
</tr>
</thead>
<tbody>
<tr>
<td>component</td>
<td>Required</td>
<td>Optional</td>
<td>Invalid</td>
<td>Required</td>
<td>Invalid</td>
</tr>
<tr>
<td>method</td>
<td>Required</td>
<td>Required</td>
<td>Required</td>
<td>Required</td>
<td>Required</td>
</tr>
<tr>
<td>returnVariable</td>
<td>Optional</td>
<td>Optional</td>
<td>Optional</td>
<td>Optional</td>
<td>Optional</td>
</tr>
<tr>
<td>argumentCollection</td>
<td>Optional</td>
<td>Optional</td>
<td>Optional</td>
<td>Optional</td>
<td>Optional</td>
</tr>
</tbody>
</table>
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>
```

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">
```

<table>
<thead>
<tr>
<th>This attribute is required, optional, ignored, or invalid:</th>
<th>For this <code>cfinvoke</code> tag syntax:</th>
</tr>
</thead>
<tbody>
<tr>
<td>username</td>
<td>Ignored</td>
</tr>
<tr>
<td>password</td>
<td>Ignored</td>
</tr>
<tr>
<td>webservice</td>
<td>Ignored</td>
</tr>
<tr>
<td>timeout</td>
<td>Invalid</td>
</tr>
<tr>
<td>proxyServer</td>
<td>Invalid</td>
</tr>
<tr>
<td>proxyPort</td>
<td>Invalid</td>
</tr>
<tr>
<td>proxyUser</td>
<td>Invalid</td>
</tr>
<tr>
<td>proxyPassword</td>
<td>Invalid</td>
</tr>
<tr>
<td>input_params ...</td>
<td>Optional</td>
</tr>
</tbody>
</table>
Example 2
This example uses Syntax 1.

<!--- passing the arguments using argumentCollection --->
<cfset args = StructNew()>
<cfset args.symbol = "macr">
<cfinvoke
  component="nasdaq.quote"
  method="getLastTradePrice"
  argumentCollection="#args#"
  returnVariable="res">
<cfoutput>#res#</cfoutput>
</cfinvoke>

Example 3
This example uses Syntax 2.

<!--- called only from within a component, MyComponent--->
<cfinvoke
  method = "a method name of MyComponent"
  returnVariable = "variable name">
<cfoutput>#fooo#</cfoutput>
</cfinvoke>

For more information on the BabelFish web service example, see Chapter 32, "Using Web Services," in Developing ColdFusion MX Applications.

Example 4
This example uses Syntax 3.

<!--- using cfinvoke to consume a web service using a ColdFusion component --->
<!--- put the following code in a ColdFusion page named wscfml.cfm:--->
<cfinvoke
  webservice='http://www.xmethods.net/sd/2001/BabelFishService.wsdl'
  method='BabelFish'
  translationmode="en_es"
  sourcedata="Hello world, friend"
  returnVariable='foo'>
<cfoutput>#foo#</cfoutput>
</cfinvoke>

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>
<cfinvoke
  component="#quoteService#"
  method="getLastTradePrice"
  symbol="mot"
  returnVariable="res_mot">
<cfoutput>#res#</cfoutput>
</cfinvoke>
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
\[
\begin{align*}
&lt;cfinvokeargument &amp;name=&quot;argument name&quot;&amp;value=&quot;argument value&quot;&gt; \\
&lt;/cfinvokeargument&gt;
\end{align*}
\]

See also
cfargument, cfcomponent, cffunction, cfinvoke, cfoobject, cfproperty, 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>Argument name</td>
</tr>
<tr>
<td>value</td>
<td>Required</td>
<td></td>
<td>Argument value</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 cfif tag to determine whether to execute the cfinvokeargument tag.

Example1
\[
\begin{align*}
&\text{<cfinvoke} \\
&\text{&lt;component=&quot;nasdaq.quote&quot;}&\\
&\text{&lt;method=&quot;getLastTradePrice&quot;}&\\
&\text{&lt;returnVariable=&quot;res&quot;&gt;}&\\
&\text{&lt;cfinvokeargument} \\
&\text{&lt;name=&quot;symbol&quot; value=&quot;mot&quot;&gt;}&\\
&\text{&lt;cfinvokeargument} \\
&\text{&lt;name=&quot;symbol&quot; value=&quot;macr&quot;&gt;}&\\
&\text{&lt;/cfinvokeargument} \\
&\text{&lt;/cfinvoke}>&\\
\end{align*}
\]

Example2
\[
\begin{align*}
&\text{<cfinvoke} \\
&\text{&lt;webservice =&quot;http://www.xmethods.net/sd/2001/BabelFishService.wsdl&quot;}&\\
&\text{&lt;method=&quot;BabelFish&quot;}&\\
&\text{&lt;returnVariable = &quot;varName&quot;}&\\
&\text{&lt;/cfinvoke}>&\\
\end{align*}
\]

\[
\begin{align*}
&\text{&lt;cfinvokeargument} \\
&\text{&lt;name=&quot;translationmode&quot; value=&quot;en_es&quot;&gt;}&\\
&\text{&lt;/cfinvokeargument}>&\\
\end{align*}
\]

\[
\begin{align*}
&\text{&lt;/cfinvokeargument}>&\\
\end{align*}
\]
name="sourcedata" value="Hello world, friend">
</cfinvoke>
<cfoutput>$varName$</cfoutput>
**cflldap**

**Description**
Provides an interface to a Lightweight Directory Access Protocol (LDAP) directory server, such as the Netscape Directory Server.

**Category**
Forms tags, Internet Protocol tags

**Syntax**
```
<cflldap
  server = "server_name"
  port = "port_number"
  username = "name"
  password = "password"
  action = "action"
  name = "name"
  timeout = "seconds"
  maxRows = "number"
  start = "distinguished_name"
  scope = "scope"
  attributes = "attribute, attribute"
  filter = "filter"
  sort = "attribute[, attribute]..."
  sortControl = "nocase" and/or "desc" or "asc"
  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">
</cflldap>
```

**See also**
cfftp, cfhttp, cfmail, cfmailparam, cfpop. Chapter 23, “Managing LDAP Directories,” in Developing ColdFusion MX Applications

**History**
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>Host name or IP address of LDAP server.</td>
<td></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. If <code>secure = &quot;CFSSL_BASIC&quot;</code>, V2 encrypts the password before transmission.</td>
</tr>
<tr>
<td>action</td>
<td>Optional</td>
<td>query</td>
<td>• <code>query</code>: returns LDAP entry information only. Requires <code>name</code>, <code>start</code>, and <code>attributes</code> attributes.&lt;br&gt;• <code>add</code>: adds LDAP entries to LDAP server. Requires <code>attributes</code> attribute.&lt;br&gt;• <code>modify</code>: modifies LDAP entries, except distinguished name <code>dn</code> attribute, on LDAP server. Requires <code>dn</code>. See <code>modifyType</code> attribute.&lt;br&gt;• <code>modifyDN</code>: modifies distinguished name attribute for LDAP entries on LDAP server. Requires <code>dn</code>. &lt;br&gt;• <code>delete</code>: 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>60</td>
<td>Maximum length of time, in seconds, 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>. &lt;br&gt;• <code>oneLevel</code>: entries one level below entry.&lt;br&gt;• <code>base</code>: only the entry.&lt;br&gt;• <code>subtree</code>: entry and all levels below it.</td>
</tr>
<tr>
<td>attributes</td>
<td>Required if <code>action = &quot;Query&quot;, &quot;Add&quot;, &quot;ModifyDN&quot;, or &quot;Modify&quot;</code></td>
<td></td>
<td>For queries: comma-delimited list of attributes to return. For queries, to get all attributes, specify <code>&quot;*&quot;</code>. If <code>action = &quot;add&quot;</code> or <code>&quot;modify&quot;</code>, you can specify a list of update columns. Separate attributes with a semicolon. If <code>action = &quot;ModifyDN&quot;</code>, ColdFusion passes attributes to the LDAP server without syntax checking.</td>
</tr>
<tr>
<td>filter</td>
<td>Optional</td>
<td>&quot;objectclass = &quot;</td>
<td>Search criteria for <code>action = &quot;query&quot;</code>. List attributes in the form: &quot;(attribute operator value)&quot; Example: &quot;(sn = Smith)&quot;</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>sort</td>
<td>Optional</td>
<td></td>
<td>Attribute(s) by which to sort query results. Use a comma delimiter.</td>
</tr>
</tbody>
</table>
| sortControl| Optional| asc     | • nocase: case-insensitive sort  
• asc: ascending (a to z) case-sensitive sort  
• desc: descending (z to a) case-sensitive sort  
You can enter a combination of sort types; for example, sortControl = "nocase, asc". |
| dn         | Required if action = "Add", "Modify", "ModifyDN", Or "delete" | | Distinguished name, for update action. Example: "cn = Bob Jensen, o = Ace Industry, c = US" |
| startRow   | Optional| 1       | Used with action = "query". First row of LDAP query to insert into a ColdFusion query. |
| modifyType | Optional| replace | How to process an attribute in a multi-value list.  
• add: appends it to any attributes  
• delete: deletes it from the set of attributes  
• replace: replaces it with specified attributes  
You cannot add an attribute that is already present or that is empty. |
| rebind     | Optional| No      | • Yes: attempt to rebind referral callback and reissue query by referred address using original credentials.  
• No: referred connections are anonymous |
| referral   | Optional|         | Integer. Number of hops allowed in a referral. A value of 0 disables referred addresses for LDAP; no data is returned. |
| secure     | Optional|         | Security to employ, and required information. One option:  
• CFSSL_BASIC  
"CFSSL_BASIC" provides V2 SSL encryption and server authentication. |
If you use the `query` action, `cfldap` creates a query object, allowing access to information in the query variables, as follows:

**Usage**

If you use the `query` action, `cfldap` creates a query object, allowing access to information in the query variables, as follows:

**Variable name** | **Description**
--- | ---
`queryname.recordCount` | Number of records returned by query
`queryname.currentRow` | Current row of query that `cfoutput` is processing
`queryname.columnList` | Column names in query

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 *Developing ColdFusion MX Applications*.

**Example**

```html
<h3>cfldap Example</h3>
```
<p>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>

<!--- If form.name exists, the form was submitted; run the query ---
<cfif IsDefined("form.name")>

<!--- check to see that there is a name listed ---
<cfif form.name is not ">"

<!--- 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"

<!--- Display results ---
<center>
<table border = 0 cellspacing = 2 cellpadding = 2>
<tr>
<th colspan = 5>
<cfoutput>#results.recordCount# matches found</cfoutput></TH>
</tr>
<tr>
<th><font size = "-2">Name</font></TH>
<th><font size = "-2">Organization</font></TH>
<th><font size = "-2">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">#cn#</font></td>
<td><font size = "-2">#o#</font></td>
<td><font size = "-2">#title#</font></td>
<td><font size = "-2">
<A href = "mailto:#mail#">#mail#</A></font></td>
<td><font size = "-2">#telephonenumber#</font></td>
</tr>
</cfoutput>
</table>
</center>
</cfif>
</cfif>

<form action="#cgi.script_name#" method="POST">
<p>Enter a name to search in the database.</p>
<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**

```coldfusion
<cflocation
    url = "url"
    addToken = "Yes" or "No">
</cflocation>
```

**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>
<tr>
<td>addToken</td>
<td>Optional</td>
<td></td>
<td>clientManagement must be enabled (see cfapplication on page 51).</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Yes: appends client variable information to URL</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• No</td>
</tr>
</tbody>
</table>

**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**

```coldfusion
<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. If you remove the comments, this code redirects you to CFDOCS home page:

<!--- <cflocation url = "http://localhost:8500/cfdocs/dochome.htm" addToken = "No"> --->
```

```coldfusion
<!----<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, cffile, ` Chapter 15, “Using Persistent Data and Locking,” in Developing ColdFusion MX Applications

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 <code>throwOnTimeout</code> attribute value.</td>
</tr>
<tr>
<td>scope</td>
<td>Optional</td>
<td></td>
<td>Lock scope. Mutually exclusive with the <code>name</code> attribute. Lock name. Only one request in the specified scope can execute the code within this tag (or within any other <code>cflock</code> tag with the same lock scope <code>scope</code>) at a time.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Application</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Server</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Session</td>
</tr>
<tr>
<td>name</td>
<td>Optional</td>
<td></td>
<td>Lock name. Mutually exclusive with the <code>scope</code> attribute. Only one request can execute the code within a <code>cflock</code> 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>
</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++ using the CFAPI.)

To work safely with ColdFusion, a C++ CFX that maintains and manipulates shared (global) data structures must be made thread-safe; however, this requires advanced knowledge. You can use a CFML custom tag wrapper around a CFX to make its invocation thread-safe.

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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Req/Opt</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>throwOnTimeout</td>
<td>Optional</td>
<td>Yes</td>
<td>How timeout conditions are handled.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Yes: exception is generated for the timeout.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• No: execution continues past this tag.</td>
</tr>
<tr>
<td>type</td>
<td>Optional</td>
<td>Exclusive</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></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>

throwOnTimeout Optional Yes How timeout conditions are handled.
- Yes: exception is generated for the timeout.
- No: execution continues past this tag.

Type Optional Exclusive • readOnly: lets more than one request read shared data.
• exclusive: lets one request read or write shared data.
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:

```cfc
<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 Developing ColdFusion MX Applications
- Article #20370, ColdFusion Locking Best Practices, on the Macromedia website at www.macromedia.com/support/service/

**Example**

```cfc
<!--- This example shows how cflock can guarantee consistency of data updates to variables in the Application, Server, and Session scopes. --->
```
<!--- Copy the following code into an Application.cfm file in the application root directory. --->

<!--- beginning of Application.cfm code --->
<!--- cfapplication defines scoping for a ColdFusion application and enables or disables storing of application and session variables. Put this tag in a special file called Application.cfm. It is run before any other CF page in its directory. --->

<!--- Enable session management for this application --->
<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 "Developing ColdFusion MX Applications". --->
<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>
</cfif>

<!--- 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>
<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>
<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>
                <option>white</option>
                <option>blue</option>
                <option>turquoise</option>
                <option>black</option>
                <option>forest green</option>
            </select></td>
        </tr>
        <tr>
            <td>Select a size.</td>
            <td><select type = "Text" name = "size" >
                <option>XXsmall</option>
                <option>Xsmall</option>
                <option>small</option>
                <option>medium</option>
                <option>large</option>
                <option>XLarge</option>
            </select></td>
        </tr>
        <tr>
            <td>Press Submit when you are finished making your selection.</td>
            <td><input type = "Submit" name = "submit" value = "Submit"> </td>
        </tr>
    </table>
</form>
**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 = "application name 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 <code>file</code> attribute, writes messages to standard log file. Ignored, if you specify <code>file</code> attribute.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Application: writes to Application.log, normally used for application-specific messages.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Scheduler: writes to Scheduler.log, normally used to log the execution of scheduled tasks.</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 &quot;Testing&quot;. 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 Information</td>
</tr>
<tr>
<td>application</td>
<td>Optional</td>
<td>Yes</td>
<td>• Yes: log application name, if it is specified in a <code>cfapplication</code> tag.</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**
Extensibility tags

**Syntax**
```
<cflogin
    idletimeout = "value"
    applicationToken = "token"
    cookieDomain = "domain"
    ...
    <cfloginuser
        name = "name"
        password = "password-string"
        roles = "roles"
    ...

    ...
</cflogin>
```

**See also**
`cfloginuser`, `cflogout`, `GetAuthUser`, `IsUserInRole`, Chapter 16, “Securing Applications,” in Developing ColdFusion MX Applications

**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 with no keyboard activity after which ColdFusion logs the user off. Seconds.</td>
</tr>
<tr>
<td>applicationToken</td>
<td>Optional</td>
<td>The current application name</td>
<td>Unique application identifier. Limits the login validity to one application, as specified by the <code>cfapplication</code> tag.</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.
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.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>
    <cfloginuser name="#cflogin.name#" password="#cflogin.password#" roles="#roles#" />
  </cfif>
</cflogin>
```
cfloginuser

**Description**
Identifies an authenticated user to ColdFusion. Specifies the user ID and roles. Used within a `cflogin` tag.

**Category**
Extensibility tags

**Syntax**
```xml
<cfloginuser
  name = "name"
  password = "password-string"
  roles = "roles"/>
```

**See also**
`cflogin`, `cflogout`, `GetAuthUser`, `IsUserInRole`, `cfapplication`, Chapter 16, “Securing Applications,” in Developing ColdFusion MX Applications

**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 `cflogin` 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></td>
<td>A username.</td>
</tr>
<tr>
<td>password</td>
<td>Required</td>
<td></td>
<td>A user password.</td>
</tr>
<tr>
<td>roles</td>
<td>Required</td>
<td></td>
<td>A comma-delimited list of role identifiers. ColdFusion processes spaces in a list element as part of the element.</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 `IsUserInRole` return the user name and role information.

**Note:** By default, the user information is stored as memory-only cookies. The `cfapplication` tag can specify that login information be stored in the Session scope.

**Example**
See `cflogin` on page 228.
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
Extensibility tags

Syntax
<cflogout>

See also
cflogin, cfloginuser. Chapter 16, “Securing Applications,” in Developing ColdFusion MX Applications

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 cflogin tag.
ColdFusion MX: Added this tag.

Example
<cflogin>
  <cfloginuser
      name = "foo"
      password = "bar"
      roles = "admin">
  </cflogin>
<cfoutput>Authorized user: #getAuthUser()#</cfoutput>
<cflogout>
<cfoutput>Authorized user: #getAuthUser()#</cfoutput>
cfloop

Description
Looping is a programming technique that repeats a set of instructions or displays output repeatedly until one or more conditions are met. This tag supports the following types of loops:

- “cfloop: index loop” on page 233
- “cfloop: conditional loop” on page 235
- “cfloop: looping over a query” on page 236
- “cfloop: looping over a list or file” on page 238
- “cfloop: looping over a COM collection or structure” on page 239

Category
Flow-control tags
cfloop: index loop

Description
An index loop repeats for a number of times that is determined by a numeric value. An index loop is also known as a FOR loop.

Syntax
<cfloop index = "parameter_name" from = "beginning_value" to = "ending_value" step = "increment">
... HTML or CFML code ...
</cfloop>

See also
cfabort, cfbreak, cfdirectory, cfexecute, cfexit, cfif, cflocation, cfthrow, 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>index</td>
<td>Required</td>
<td></td>
<td>Index value. ColdFusion sets it to from value and increments or decrements by step value, until it equals 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.

```coldfusion
<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:

```coldfusion
<cfloop index = "LoopCount" from = "5" to = "1" step = "-1">
  The loop index is #LoopCount#<cfoutput></cfoutput>.<br>
</cfloop>
```

The output of this loop is as follows:
- The loop index is 5.
- The loop index is 4.
- The loop index is 3.
- The loop index is 2.
- The loop index is 1.
cfloop: conditional loop

Description
A conditional loop iterates over a set of instructions as long as a condition is True. To use this type of loop correctly, the instructions must change the condition every time the loop iterates, until the condition is False. Conditional loops are known as WHILE loops, as in, "loop WHILE this condition is true."

Syntax
```xml
<cfloop condition = "expression">
    ...
</cfloop>
```

See also
cfabsert, cfbreak, cfexecute, cfexit, cfif, cflocation, 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>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.

```xml
<!-- Set the variable CountVar to 0 -->
<cfset CountVar = 0>
<!-- Loop until CountVar = 5 -->
<cfloop condition = "CountVar LESS THAN OR EQUAL TO 5">
    <cfset CountVar = CountVar + 1>
    The loop index is <cfoutput>#CountVar#</cfoutput>.<br>
</cfloop>
```

The output of this loop is as follows:

The loop index is 1.
The loop index is 2.
The loop index is 3.
The loop index is 4.
The loop index is 5.
cfloop: looping over a query

Description
A loop over a query executes for each record in a query record set. The results are similar to those of the cfoutput tag. During each iteration, the columns of the current row are available for output. The cfloop tag loops over tags that cannot be used within a cfoutput tag.

Syntax
<cfloop query = "query_name" startRow = "row_num" endRow = "row_num">
</cfloop>

See also
cfabsort, cfbreak, cfexecute, cfexit, cfif, cflocation, cfoutput, 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>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
This example shows a cfloop looping over a query the same way as a cfoutput tag that uses the query attribute:

<cfquery name = "MessageRecords"
  dataSource = "cfsnippets">
  SELECT * FROM Messages
</cfquery>
<cfloop query = "MessageRecords">
  <cfoutput>#Message_ID#</cfoutput><br>
</cfloop>

The cfloop tag also iterates over a record set with dynamic start and stop points. This gets the next n sets of records from a query. This example loops from the 10th through the 20th record returned by MyQuery:

<cfset Start = 10>
<cfset End = 20>
<cfloop query = "MyQuery"
  startRow = "#Start#"
  endRow = "#End#">
  <cfoutput>@MyQuery.MyColName#</cfoutput><br>
</cfloop>

The loop stops when there are no more records, or when the current record index is greater than the value of the endRow attribute.

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

<cfquery name = "GetTemplate"
dataSource = "Library"
maxRows = "5">
SELECT TemplateName
FROM Templates
</cfquery>
<cfloop query = "TemplateName">
  <cfinclude template = "#TemplateName#"/>
</cfloop>
cfloop: looping over a list or file

Description
Looping over a list steps through elements contained in any of these entities:

• A variable
• A value that is returned from an expression
• A file

Syntax
<cfloop
index = "index_name"
list = "list_items"
delimiters = "item_delimiter">
...
</cfloop>

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

Attributes

<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>In a list loop, the variable to receive next list element.</td>
<td></td>
</tr>
<tr>
<td>list</td>
<td>Required</td>
<td>A list, variable, or filename; contains a list</td>
<td></td>
</tr>
<tr>
<td>delimiters</td>
<td>Optional</td>
<td>Character(s) that separates items in list</td>
<td></td>
</tr>
</tbody>
</table>

Example
This loop displays four names:

<cfloop index = "ListElement"
  list = "John,Paul,George,Ringo">
  <cfoutput>#ListElement#</cfoutput><br>
</cfloop>

You can put more than one character in the delimiters attribute, in any order. For example, this loop processes commas, colons, and slashes as list delimiters:

<cfloop index = "ListElement"
  list = "John/Paul,George::Ringo"
  delimiters = ",:/">
  <cfoutput>#ListElement#</cfoutput><br>
</cfloop>

ColdFusion skips the second and subsequent consecutive delimiters between list elements. Thus, in the example, the two colons between "George" and "Ringo" are processed as one delimiter.

To loop over each line of a file, use the tag this way:

<cfloop list="#theFile#"
  index="curLine"
  delimiters="#chr(10)##chr(13)#">
  ...
</cfloop>
**cfloop: looping over a COM collection or structure**

**Description**

The `cfloop` collection attribute loops over every object within a COM/DCOM collection object, or every element in a structure:

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

In the loop, each item is referenced by the variable name in the `item` attribute. The loop executes until all items have been accessed.

The `collection` attribute is used with the `item` attribute. In the example that follows, `item` is assigned a variable called `file2`, so that with each cycle in the `cfloop`, each item in the collection is referenced. In the `cfoutput` section, the `name` property of the `file2` item is referenced for display.

**Example**

This example uses a COM object to output a list of files. In this example, `FFunc` is a collection of `file2` objects.

```cfc
<cfobject class = FileFunctions.files name = FFunc action = Create>
<cfset FFunc.Path = "c:\">
<cfset FFunc.Mask = "*.txt">
<cfset FFunc.attributes = 16>
<cfset x = FFunc.GetFilesList()>
<cfloop collection = #FFUNC# item = "file2">
<cfoutput>#file2.name#</cfoutput>
</cfloop>
<!---Loop through a structure that is used as an associative array: --->
<table cellspacing = "2 " cellpadding = "2 ">
<tr>
<td><b>Employee</b></td>
<td><b>Dept.</b></td>
</tr>
<cfloop collection = #Departments# item = "person">
<tr>
<td><b>#person#</b></td>
<td><b>#StructFind(Departments, person)#</b></td>
</tr>
</cfloop>
</table>
```

---

cfloop: looping over a list or file   239
cfmail

Description
Sends an e-mail message that optionally contains query output, using an SMTP server.

Category
Forms tags, Internet Protocol tags

Syntax
```cfc
<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">
  (Optional) Mail message body and/or cfttpparam tags
</cfmail>
```

See also
cfmailparam, cfmailpart, cfpop, cfhttp, cfldap, Wrap

History
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, 
  "#Form.Email#".  
• Name of a query column that contains an address. For example, "#Email#". An e-mail message is sent for each returned row. |
| from      | Required|         | E-mail message sender:  
• A static string; for example, "support@mex.com"  
• A variable; for example, "#GetUser.EMailAddress#"  
This attribute does not have to be a valid internet address; it can be any text string. |
<p>| cc        | Optional|         | Address(es) to which to copy the message. |
| bcc       | Optional|         | Address(es) to which to copy the message, without listing them in the message header. |
| subject   | Required|         | Message subject. Can be dynamically generated. For example, to send messages that give customers status updates, &quot;Status of Order Number #Order_ID#&quot;. |
| replyto   | Optional|         | Address(es) to which the recipient is directed to send replies. |
| failto    | Optional|         | Address to which mailing systems should send delivery failure notifications. Sets the mail envelope reverse-path value. |
| username  | Optional|         | A user name to send to SMTP servers that require authentication. Requires a password attribute. |
| password  | Optional|         | A password to send to SMTP servers that require authentication. Requires a username attribute. |
| wraptext  | 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>
</table>
| charset    | Optional| Character encoding selected in ColdFusion MX Administrator Mail page; default is UTF-8 | Character encoding of the mail message, including the headers. The following list includes commonly used values:  
  • utf-8  
  • iso-8859-1  
  • windows-1252  
  • us-ascii  
  • shift_jis  
  • iso-2022-jl  
  • 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. |
| type       | Optional| text/plain                        | MIME type of the message. Can be a valid MIME media type or one of the following:  
  • 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/ |
| MIMEAttach | Optional| 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. |
| query      | Optional| Name of cfquery from which to draw data for message(s). Use this attribute to send more than one message, or to send query results within a message. |
| group      | Optional| CurrentRow                        | Query column to use when you group sets of records to send as a message. For example, to send a set of billing statements to a customer, group on "Customer_ID." Case-sensitive. Eliminates adjacent duplicates when data is sorted by the specified field. |
| groupCaseSensitive | Optional| No                                | Boolean. Whether to consider case when using the group attribute. To group on case-sensitive records, set this attribute to Yes. |
| startRow   | Optional| 1                                 | Row in a query to start from. |
| maxRows    | Optional|                                   | Maximum number of messages to send when looping over a query. |
| server     | Optional|                                   | 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. |
cfmail 243

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:smith@company.com">smith@company.com</a></td>
</tr>
<tr>
<td>DisplayName user@server</td>
<td>Rob Smith <a href="mailto:smith@company.com">smith@company.com</a></td>
</tr>
<tr>
<td>&quot;DisplayName&quot; user@server</td>
<td>&quot;Rob Smith&quot; <a href="mailto:smith@company.com">smith@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>
```

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Req/Opt</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>port</td>
<td>Optional</td>
<td>TCP/IP port</td>
<td>TCP/IP port on which SMTP server listens for requests (normally 25). A value here overrides the Administrator.</td>
</tr>
<tr>
<td>mailerID</td>
<td>Optional</td>
<td>ColdFusion MX Application Server</td>
<td>Mailer ID to be passed in X-Mailer SMTP header, which identifies the mailer application.</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></td>
<td></td>
<td>Yes: Saves a copy of the message until the sending operation is complete. Pages that use this option might run slower than those that use the No option.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>No: Queues the message for sending, without storing a copy until the operation is complete. If a delivery error occurs when this option is No, ColdFusion generates an Application exception and logs the error to the mail.log file.</td>
<td></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

Example cfmail

<!--- 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#"
      This message was sent by an automatic mailer built with cfmail:
      = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
      #form.body#
    </cfmail>
  </cfif>
</cfif>

<!--- 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>
**cfmailparam**

**Description**
Attaches a file or adds a header to an e-mail message. Can only be used in the `cfmail` tag. You can use more than one `cfmailparam` tag within a `cfmail` tag.

**Category**
Forms tags, Internet Protocol tags

**Syntax**
```xml
<cfmail
    to = "recipient"
    subject = "msg_subject"
    from = "sender"
    ...more attributes... >
<cfmailparam
    file = "file-name"
    type = "media type">
or
<cfmailparam
    name = "header-name"
    value = "header-value" >
...
</cfmail>
```

**See also**
cfmail, cfmailpart, cfhttp, cfldap, cfpop

**History**
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 <code>name</code> attribute</td>
<td></td>
<td>Attaches file to a message. Mutually exclusive with <code>name</code> attribute. The file is MIME encoded before sending.</td>
</tr>
</tbody>
</table>
| type      | Optional | | The MIME media type of the part. Can be a 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/
| name      | Required if you do not specify `file` attribute | | Name of header. Case-insensitive. Mutually exclusive with `file` attribute. |
| value     | Optional | | Value of the header. |

**Example**
```
<h3>cfmailparam Example</h3>
<p>This view-only example uses `cfmailparam` to attach files and add header to a message.</p>
<cfmail from = "peter@domain.com" To = "paul@domain.com"
    Subject = "See Important Attachments and Reply">
```
<cfmailparam name = "Importance" value = "High">
Please review the new logo. Tell us what you think.
<cfmailparam file = "c:\work\readme.txt" type="text/plain">
<cfmailparam file = "c:\work\logo.gif" type="image/gif">
</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
Forms tags, 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, Chapter 35, “Sending and Receiving E-Mail,” in Developing ColdFusion MX Applications

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>The MIME media type of the part. Can be a can be valid MIME media type or one of the following:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• text specifies text/plain type</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• plain specifies text/plain type</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• html specifies text/html type</td>
</tr>
</tbody>
</table>

**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

```xml
<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:</td>
</tr>
</tbody>
</table>

For more information on character encodings, see: www.w3.org/International/O-charset.html.
**cfmodule**

**Description**
Invokes a custom tag for use in ColdFusion application pages. This tag processes custom tag name conflicts.

For more information, see Chapter 9, “Creating and Using Custom CFML Tags,” in *Developing ColdFusion MX Applications*.

**Category**
Application framework tags

**Syntax**
```xml
<cfmodule
    template = "path"
    name = "tag_name"
    attributeCollection = "collection_structure"
    attribute_name1 = "valuea"
    attribute_name2 = "valueb"
...>
```

**See also**
cfapplication, cfassociate, cflock, Chapter 9, “Creating and Using Custom CFML Tags,” in *Developing ColdFusion MX Applications*

**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.

Example

```
<h3>cfmodule Example</h3>
<p>This view-only example shows use of cfmodule to call a custom tag inline.</p>
<p>This example uses a sample custom tag that is saved in myTag.cfm in the snippets directory. You can also save ColdFusion custom tags in the Cfusion\CustomTags directory.
<cfset attrCollection1 = StructNew()>
<cfparam name="attrCollection1.value1" default="22">
<cfparam name="attrCollection1.value2" default="45">
<cfparam name="attrCollection1.value3" default="88">
<!--- Call the tag with CFMODULE with Name--->
<cfmodule
Template="myTag.cfm"
X="3"
attributeCollection=#attrCollection1#
Y="4">
<!--- show the code --->
<HR size="2" color="#0000A0">
<P>Here is one way in which to invoke the custom tag, using the TEMPLATE attribute.</P>
<cfoutput>#HTMLCodeFormat("<CFMODULE
Template="myTag.cfm"
X=3
attributeCollection=#attrCollection1#
Y=4")#</cfoutput>
<!--- Call the tag with CFMODULE with Name--->
```

<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>
Here is another way to invoke the custom tag, using the NAME attribute.

```
<cfoutput><![CDATA[
<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><![CDATA[
<cfmodule Name='myTag' X='3' attributeCollection=#attrCollection1# Y='4'>
</cfoutput>
</cfoutput>
<p>The result: <cfoutput>#result#</cfoutput></p>
<!---- Call the tag using the short cut notation ---->
<!----
<cf_myTag X='3' attributeCollection=#attrCollection1# Y='4'>
<!---- show the code ---->
<p>Here is the short cut to invoking the same tag. </p>
<cfoutput><![CDATA[
<cf_mytag x='3' attributeCollection='#attrcollection1#' y='4'>
</cfoutput>
</cfoutput>
<p>The result: <cfoutput>#result#</cfoutput></p>
```
**cfobject**

**Description**

Creates a ColdFusion object, of a specified type.

*Note:* You can enable and disable this tag in the ColdFusion Administrator page, under ColdFusion Basic Security, Tag Restrictions.

**Category**

Extensibility tags

**Syntax**

The tag syntax depends on the object type. Some types use the *type* attribute; others do not. See the following sections:

- “cfobject: COM object” on page 254
- “cfobject: component object” on page 256
- “cfobject: CORBA object” on page 257
- “cfobject: Java or EJB object” on page 259
- “cfobject: web service object” on page 261

*Note:* On UNIX, this tag does not support COM objects.

**See also**

cfargument, cfcomponent, cffunction, cfinvoke, cfinvokeargument, cfproperty, cfreturn

**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 config 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.

To use this tag, you must provide the object’s program ID or filename, the methods and properties available through the IDispatch interface, and the arguments and return types of the object’s methods. For most COM objects, you can get this information with the OLEView utility.

**Note:** On UNIX, this tag does not support COM objects.

**Syntax**
```
<cfobject
    type = "com"
    action = "action"
    class = "program_ID"
    name = "text"
    context = "context"
    server = "server_name">
```

**See also**
ReleaseComObject, cfcollection, cfexecute

**Attributes**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Req/Opt</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
</table>
| type      | Optional |         | Object type:
|           |         |         | • com
|           |         |         | • corba
|           |         |         | • java
|           |         |         | (The other object types do not take the type attribute.) |
| action    | Required |         | • create: instantiates a COM object (typically, a DLL) before invoking methods or properties.
|           |         |         | • connect: connects to a COM object (typically, an EXE) running on server. |
| class     | Required |         | Component ProgID for the object to invoke. When using Java stubs to connect to the COM object, the class must be the ProgID of the COM object. |
| name      | Required |         | String; name for the instantiated component |
Example

```<h3>cfobject (COM) Example</h3>
<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()>

  <!--- Loop over all the objects in the collection.--->
  <cfloop collection = #obj# item = file2>
    <cfoutput>Last name: #file2.lastname#</cfoutput>
  </cfloop>
<br>
</cfoutput>
```

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Req/Opt</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
</table>
| context   | Optional | • inproc  
  • local  
  • remote  
  On Windows: If not specified, uses Registry setting. |
| server    | Required if context = "Remote" | Server name, using Universal Naming Convention (UNC) or Domain Name Server (DNS) convention, in one of these forms:  
  • \lanserver  
  • lanserver  
  • http://www.servername.com  
  • www.servername.com  
  • 127.0.0.1 |
**cfobject: component object**

**Description**
Creates an instance of a ColdFusion component (CFC) object.

**Syntax**
```cfobject
    name = "variable name"
    component = "component name"
</cfobject>
```

**See also**
collection, ccomponent, cfexecute, cfindex, cfreport, cfsearch, cfwddx

**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 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.

**Example**
```cfobject
<!--- separate instantiation and method invocation; permits multiple invocations --->
<cfobject
    name="quoteService"
    component="nasdaq.quote">
<cffinvoke
    component="#quoteService#"
    method="getLastTradePrice"
    symbol="macr"
    returnVariable="res">
<cfoutput>#res#</cfoutput><br>
<cffinvoke
    component="#quoteService#"
    method="getLastTradePrice"
    symbol="mot"
    returnVariable="res">
<cfoutput>#res#</cfoutput>
```
cfobject: CORBA object

Description
Calls methods on a registered CORBA object.

Syntax
<cfobject
  type = "corba"
  context = "context"
  class = "file or naming service"
  name = "text"
  locale = "type-value arguments">

See also
collection, cfexecute, cfindex, cfreport, cfsearch, cfwddx

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>• com</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• corba</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• java</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>(The other object types do not take the type attribute.)</td>
<td></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>• nameservice: ColdFusion uses naming service to access server. This option is valid only with the InitialContext of a VisiBroker Orb.</td>
<td></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>• If context = &quot;nameservice&quot;: forward slash-delimited naming context for naming service. For example: Allaire//Doc/empobject</td>
<td></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: locale = &quot; -ORBagentAddr 199.99.129.33 -ORBagentPort 19000&quot; 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

```xml
<cfobject type = "corba"
  context = "ior"
  class = "c:\myobject.ior"
  name = "GetName">
```
**cfobject: Java or EJB object**

**Description**
Creates and manipulates a Java and Enterprise Java Bean (EJB) object.

**Syntax**
```
<cfobject
    type = "Java"
    action = "Create"
    class = "Java class"
    name = "object name">
```

**See also**
collection, cfexecute, cfindex, cfreport, cfsearch, cfwddx

**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 EJ Bs**
To create and call EJB objects, use the `cfobject` tag. In the second example below, the WebLogic JNDI is used to register and find EJBHome instances.
Example

<!-- Example of a Java Object his cfobject call loads the class MyClass but does not create an instance object. Static methods and fields are accessible after a call to cfobject. -->
<cfobject
    action = "create"
    type = "java"
    class = "myclass"
    name = "myobj">
</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. -->
<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, cfexecute, cfindex, cfreport, cfsearch, cfwddx

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>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, 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</td>
<td>Clears queries from the cache in the Application scope</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**
```cfc
<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

**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></td>
<td>Name of cfquery from which to draw data for output section.</td>
</tr>
<tr>
<td>group</td>
<td>Optional</td>
<td></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>
</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></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 pound 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 queryCurrentRow 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**

```cfml
<!--- This example shows how cfoutput operates --->
<!---- run a sample query ---->
<cfquery name = "GetCourses" dataSource = "cfsnippets">
   SELECT Dept_ID, CorName, CorLevel
   FROM courseList
   ORDER by Dept_ID, CorLevel, CorName
</cfquery>

<h3>cfoutput Example</h3>
<p>cfoutput tells ColdFusion Server to begin processing, and then to hand back control of page rendering to the web server. For example, to show today's date, you could write #DateFormat("#Now()#"). If you enclosed that expression in cfoutput, the result would be <cfoutput>#DateFormat(Now())#</cfoutput>.</p>

<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>
<p><cfoutput query = "GetCourses" maxRows = 4>
   #Dept_ID##CorName##CorLevel#<br>
</cfoutput></p>

<p>The next example uses the group attribute to eliminate duplicate lines from a list of course levels taught in each department.</p>
<p><cfquery name = "GetCourses" dataSource = "cfsnippets">
   SELECT Dept_ID, CorLevel
   FROM courseList
   ORDER by Dept_ID, CorLevel
</cfquery>
<p><cfoutput query = "GetCourses" group="CorLevel" GroupCaseSensitive="True">
   #Dept_ID# #CorLevel#<br>
</cfoutput></p>

<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:</p>
<p>Today is #DayOfWeekAsString(DayofWeek(Now()))#</p>
<p>Today is <cfoutput>#DayOfWeekAsString(DayofWeek(Now()))#</cfoutput></p>
```
**cfparam**

**Description**
Tests for a parameter's existence, tests its data type, and, if a default value is not assigned, optionally provides one.

**Category**
Variable manipulation tags

**Syntax**
```cfparam
<cfparam
  name = "param_name"
  type = "data_type"
  default = "value">
```

**See also**
cfcookie, cfregistry, cfsavecontent, cfschedule, cfset

**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 parameter 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>
</tbody>
</table>
| type      | Optional| any     | The parameter data type:  
  - any: any type of value  
  - array: an array value  
  - binary: a binary value  
  - boolean: a Boolean value  
  - date: a date-time value  
  - guid: a Universally Unique Identifier that follows the Microsoft/DCE standard, as follows: "XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX" where 'X' is a hexadecimal number.  
  - numeric: a numeric value  
  - query: a query object  
  - string: a string value or single character  
  - struct: a structure  
  - UUID: a ColdFusion Universally Unique Identifier, formatted 'XXXXXXXX-XXXX-XXXX-XXXXXXXXXXXXXXX', where 'X' is a hexadecimal number. See `CreateUUID` on page 438.  
  - variableName: a string formatted according to ColdFusion variable naming conventions. |
| default   | Optional|         | Value to set parameter to if it does not exist. |

**Usage**
You can use this tag to make the following tests:

- To test whether a required variable exists, use this tag with only the `name` attribute. If it does not exist, ColdFusion 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 test whether an optional variable exists, 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 returnType attribute, the parameter 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>"
</p>
<!--- check if tempVar is still the same as StoreTempVar and that tempVar is not blank --->
<cfif tempVar is not #StoreTempVar# and tempVar is not ">>
<h3>The value of tempVar has changed; the new value is #tempVar#</h3>
</cfif>

<form action = "cfparam.cfm" method = "post">
Type in a new value for tempVar, and hit submit:<br>
<input type = "Text" name = "tempVar">
<input type = "Submit" name = "" value = "submit">
</form>
**cfpop**

**Description**
Retrieves or deletes e-mail messages from a POP mail server.

**Category**
Forms tags, Internet Protocol tags

**Syntax**
```html
cfpop
  server = "servername"
  port = "port_number"
  username = "username"
  password = "password"
  action = "action"
  name = "queryname"
  messageNumber = "number"
  uid = "number"
  attachmentPath = "path"
  timeout = "seconds"
  maxRows = "number"
  startRow = "number"
  generateUniqueFilenames = "boolean"
</cfpop>
```

**See also**
cfftp, cfhttp, cfldap, cfmail, cfmailparam, SetLocale

**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>password</td>
<td>Optional</td>
<td></td>
<td>Password that corresponds to username.</td>
</tr>
</tbody>
</table>
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.

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 <em>cfoutput</em> 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.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>not available</td>
<td>yes</td>
</tr>
<tr>
<td>queryname.textBody</td>
<td>not available</td>
<td>yes</td>
</tr>
<tr>
<td>queryname.HTMLBody</td>
<td>not available</td>
<td>yes</td>
</tr>
<tr>
<td>queryname.header</td>
<td>not available</td>
<td>yes</td>
</tr>
<tr>
<td>queryname.attachments</td>
<td>not available</td>
<td>yes</td>
</tr>
<tr>
<td>queryname.attachmentfiles</td>
<td>not available</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.attachments 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 35, “Sending and Receiving E-Mail,” in Developing ColdFusion MX Applications.

Example

```cfml
<!--- 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, un-comment this code and run with a mail-enabled CF Server.

<!---
<cfif IsDefined("form.server ")>
    <!--- make sure server, username are not empty --->
    <cfif form.server is not "" and form.username is not "">
        <cfpop server = "#server# " username = #UserName# password = #pwd# action = "GETHEADERONLY " name = "GetHeaders ">
          <h3>Message Headers in Your Inbox</h3>
          <p>Number of Records: #GetHeaders.recordCount#</p>
          <ul>
              <cfoutput query = "GetHeaders">
                <li>Row: #currentRow#: From: #From# -- Subject: #Subject#</li>
              </cfoutput>
          </ul>
          </cfpop>
    </cfif>
</cfif>
</cfif>

<form action = "cfpop.cfm " method = "post">
    <p>Enter your mail server: </p>
    <p><input type = "Text" name = "server"></p>
    <p>Enter your username: </p>
    <p><input type = "Text" name = "username"></p>
    <p>Enter your password: </p>
    <p><input type = "password" name = "pwd"></p>
    <input type = "Submit" name = "get message headers">
</form>
```
**cfprocessingdirective**

**Description**

Provides the following information to ColdFusion on 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

**Syntax**

```xml
<cfprocessingdirective
    pageencoding = "page-encoding literal string" />
```

or

```xml
<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 Developing ColdFusion MX Applications

**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>&lt;cfprocessingdirective&gt;</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>
</tbody>
</table>
| pageEncoding      | Optional| Character encoding identified by the page byte order mark, if any; otherwise, system default encoding | 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 `<cfprocessingdirective>` 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-kr  
  • big5  
  • euc-cn  
  • utf-16  
  For more information on character encodings, see: www.w3.org/International/O-charset.html. |

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.

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:
  ```cfml
  <cfif dynEncoding is not "dynamic encoding is not possible">  
  <cfprocessingdirective pageencoding="#dynEncoding">  
  </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:

  <!--- 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

  <cfprocessingdirective suppressWhiteSpace = "Yes">
    <!--- CFML code --->
    <cfprocessingdirective suppressWhiteSpace = "No">
      <cfoutput>#table_data#
      </cfoutput>
    </cfprocessingdirective>
  </cfprocessingdirective>

  The following example shows the use of the pageencoding attribute:

  <cfprocessingdirective pageencoding = "shiftjis">
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
cfindert, cfprocreturn, cfquery, cfqueryparam, cfstoredproc, cftransaction, cfupdate

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>
<tbody>
<tr>
<td>type</td>
<td>Optional</td>
<td>in</td>
<td>• in: The parameter is used to send data to the database system only. Passes the parameter by value.&lt;br&gt;• out: The parameter is used to receive data from the database system only. Passes the parameter as a bound variable.&lt;br&gt;• inout: The parameter is used to send and receive data. Passes the parameter as a bound variable.</td>
</tr>
<tr>
<td>variable</td>
<td>Required if type = &quot;OUT&quot; or &quot;INOUT&quot;</td>
<td></td>
<td>ColdFusion variable name; references the value that the output parameter has after the stored procedure is called.</td>
</tr>
<tr>
<td>value</td>
<td>Required if type = &quot;IN&quot; or &quot;INOUT&quot;</td>
<td></td>
<td>Value that ColdFusion passes to the stored procedure.</td>
</tr>
</tbody>
</table>
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.

Output variables are stored in the ColdFusion variable specified by the `variable` attribute.

You cannot use the `cfprocparam` tag for Oracle 8 reference cursors. Instead, use the `cfprocresult` tag.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Req/Opt</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CFSQLType</td>
<td>Required</td>
<td></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>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• CF_SQL_BIGINT</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• CF_SQL_BIT</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• CF_SQL_BLOB</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• CF_SQL_CHAR</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• CF_SQL_CLOB</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• CF_SQL_DATE</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• CF_SQL_DECIMAL</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• CF_SQL_DOUBLE</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• CF_SQL_FLOAT</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• CF_SQL_IDSTAMP</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• CF_SQL_INTEGER</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• CF_SQL_LONGVARCHAR</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• CF_SQL_MONEY</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• CF_SQL_MONEY4</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• CF_SQL_NUMERIC</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• CF_SQL_REAL</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• CF_SQL_REFCURSOR</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• CF_SQL_SMALLINT</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• CF_SQL_TIME</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• CF_SQL_TIMESTAMP</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• CF_SQL_TINYINT</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• CF_SQL_VARCHAR</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 <code>maxLength</code> of 0 allows any length. The <code>maxLength</code> attribute is not required when specifying <code>type=out</code>.</td>
</tr>
<tr>
<td>scale</td>
<td>Optional</td>
<td>0</td>
<td>Number of decimal places in numeric parameter. A <code>scale</code> of 0 allows any number of decimal places.</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 <code>value</code> attribute</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• No</td>
</tr>
</tbody>
</table>

cfprocparam
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.

```sql
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>
  <cfprocparam type = "Out" CFSQLType = "CF_SQL_REFCURSOR"
    variable = "param2">
  </cfprocparam>
  <cfprocparam type = "IN" CFSQLType = "CF_SQL_INTEGER" value = "1">
    <cfprocparam variable = "param3">
  </cfprocparam>
  <cfprocparam type = "OUT" CFSQLType = "CF_SQL_VARCHAR"
    variable = "FOO">
    <cfprocparam variable = "pParam4">
  </cfprocparam>
  <cfprocresult name = "rs1">
    <cfprocresult name = "rs2" resultSet = "2"/>
</cfstoredproc>
<b>The first result set:</b><br>
```

The first result set:
<hr>
<cfoutput>
<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

**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 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**
```cfml
<!--- This example executes a Sybase stored procedure that returns three result sets, two of which we want. The stored procedure returns status code and one output parameter, which we display. We use named notation for parameters. --->
<cfstoredproc procedure = "foo_proc"
    dataSource = "MY_SYBASE_TEST" username = "sa"
    password = "" dbServer = "scup" dbName = "pubs2"
    returnCode = "Yes" debug = "Yes">
<cfprocresult name = RS1>
<cfprocresult name = RS3 resultSet = 3>
<!--- cfprocparam tags --->
```
<cfprocresult type="IN"
    CFSQLType = CF_SQL_INTEGER
    value = "1" dbVarName = @param1>
</cfprocresult>
<cfprocparam type = "OUT" CFSQLType = CF_SQL_DATE
    variable = FOO dbVarName = @param2>
</cfprocparam>
<!--- Close the cfstoredproc tag --->
</cfstoredproc>
<cfoutput>
The output param value: '#foo#'
</cfoutput>
<h3>The Results Information</h3>
<cfoutput query = RS1>#name#,DATE_COL#</cfoutput>
<p>
<cfoutput>
<hr>
<p>Record Count: #RS1.recordCount# <p>Columns: #RS1.columnList#
</cfoutput>
</cfoutput>
<cfoutput query = RS3>#col1#,col2#,col3#</cfoutput>
<p>
<cfoutput>
<hr>
<p>Record Count: #RS3.recordCount# <p>Columns: #RS3.columnList#
</cfoutput>
</cfoutput>
The return code for the stored procedure is: '#cfstoredproc.statusCode#'
**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**
```xml
<cfproperty
    name="name"
    type="type"
    required="boolean"
    default="default value"
    displayname="descriptive name"
    hint="extended description"
>
```

**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: any, array, binary, boolean, date, guid. The argument must be a UUID or GUID of the form <code>xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx</code> 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 <code>xxxxxxxx-xxxx-xxxx-xxxxxxxxxxxx</code> where each x is a character representing a hexadecimal number (0-9A-F). variableName: a string formatted according to ColdFusion variable naming conventions.. 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.</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.

Exceptions
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.

Macromedia recommends that you use the `cfqueryparam` tag within every `cfquery` tag, to help secure your databases from unauthorized users. For more information, see:

- Chapter 20, “Accessing and Retrieving Data,” in *Developing ColdFusion MX Applications*

**Category**
Database manipulation tags

**Syntax**
```
<cfquery
  name = "query_name"
  dataSource = "ds_name"
  dbtype = "query"
  username = "username"
  password = "password"
  maxRows = "number"
  blockFactor = "blocksize"
  timeout = "seconds"
  cachedAfter = "date"
  cachedWithin = "timespan"

  Either of the following:
  debug = "Yes" or "No"
  or:
  debug

  SQL statement(s)
</cfquery>
```

**See also**
`cfinsert, cfproccparam, cfprocresult, cfqueryparam, cfstoredproc, cftransaction, cfupdate`. chapters 19-22 of *Developing ColdFusion MX Applications*

**History**
ColdFusion MX:

- Changed Query of Queries behavior: it now supports a larger subset of standard SQL. For more information, see Chapter 22, “Using Query of Queries,” in *Developing ColdFusion MX Applications*.

- Changed dot notation support: ColdFusion now supports dot notation within a record set name. ColdFusion interprets such a name as a structure. For more information, see Chapter 22, “Using Query of Queries,” in *Developing ColdFusion MX Applications*.

- 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.
### 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</td>
<td>Name of data source from which query gets data.</td>
<td></td>
</tr>
<tr>
<td>dbtype</td>
<td>Optional</td>
<td>query</td>
<td>query. Use this value to specify the results of a query as input.</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>-1 (All)</td>
<td>Maximum number of rows to return in record set.</td>
</tr>
<tr>
<td>blockFactor</td>
<td>Optional</td>
<td>1</td>
<td>Maximum rows to get at a time from server. Range: 1 - 100. Might not be supported by some database systems.</td>
</tr>
<tr>
<td>timeout</td>
<td></td>
<td></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>
</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>
<tr>
<td>cachedWithin</td>
<td>Optional</td>
<td>Timespan, using the <code>CreateTimeSpan</code> function. If original query date falls within the time span, cached query data is used. <code>CreateTimeSpan</code> 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.</td>
<td></td>
</tr>
<tr>
<td>debug</td>
<td>Optional; value and equals sign may be omitted</td>
<td>• 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.</td>
<td></td>
</tr>
</tbody>
</table>

### Usage

Because the `timeout` parameter only the maximum time for each sub-operation of a query, the cumulative time may exceed its value. To set a timeout for a page that might get a very large result set, set the Administrator > Server Settings > Timeout Requests option to an appropriate value.

This tag returns data and query information from a ColdFusion data source. The cumulative query execution time, in seconds, is returned in the variable `cfquery.ExecutionTime`. 
This tag creates a query object, providing this information in query variables:

<table>
<thead>
<tr>
<th>Variable name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>query_name.currentRow</td>
<td>Current row of query that cfoutput is processing</td>
</tr>
<tr>
<td>query_name.columnList</td>
<td>Comma-delimited list of the query columns</td>
</tr>
<tr>
<td>query_name.RecordCount</td>
<td>Number of records (rows) returned from the query</td>
</tr>
<tr>
<td>cfquery.ExecutionTime</td>
<td>Cumulative time required to process the query</td>
</tr>
</tbody>
</table>

You can cache query results and execute stored procedures. For information about this and about displaying cfquery output, see Developing ColdFusion MX Applications.

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:

```sql
SELECT [count] FROM MYTABLE.
```

For a list of reserved keywords in ColdFusion MX, see Chapter 22, “Using Query of Queries,” in Developing ColdFusion MX Applications.

Database query results for date and time values can vary in sequence and formatting, unless you use functions to format the results. To ensure that customers using your ColdFusion application are not confused by the display, Macromedia recommends that you use the DateFormat and TimeFormat functions to format values from queries. For more information and examples, see TechNote 22183, "ColdFusion Server (5 and 4.5.x) with Oracle: Formatting Date and Time Query Results," at www.coldfusion.com/Support/KnowledgeBase/SearchForm.cfm.

**Example**

```cfml
<!--- This example shows the use of CreateTimeSpan with CFQUERY ------>
<!--- define startrow and maxrows to facilitate 'next N' style browsing ---->
<cfparam name="MaxRows" default="10">
<cfparam name="StartRow" default="1">
<!--------------------------------------------------------------------
Query database for information if cached database information has not been updated in the last six hours; otherwise, use cached data. 
--------------------------------------------------------------------->
<cfquery
    name="GetParks" datasource="cfsnippets" 
    cachedWithin="#CreateTimeSpan(0, 0, 6, 0)#">
    SELECT PARKNAME, REGION, STATE
    FROM Parks
    ORDER BY ParkName, State
</cfquery>
<!---- build HTML table to display query ------------------------->
<table cellpadding="1" cellspacing="1">
    <tr>
        <td colspan="2" bgcolor="f0f0f0">
            <b><i>Park Name</i></b>
        </td>
        <td bgcolor="f0f0f0">
            <b><i>Region</i></b>
        </td>
    </tr>
    <!--- iterate through query results --->
    <cfloop query="GetParks">
        <tr>
            <td bgcolor="#f0f0f0">
                <b>Park Name</b>
            </td>
            <td bgcolor="#f0f0f0">
                <b>Region</b>
            </td>
        </tr>
    </cfloop>
</table>
```
<table>
<thead>
<tr>
<th>State</th>
</tr>
</thead>
</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>
</td>
</tr>
</cfoutput>

<!----- If the total number of records is less than or equal to the total number of rows, then offer a link to the same page, with the startrow value incremented by maxrows (in the case of this example, incremented by 10) --------->
<tr>
<td colspan="4">
<cfif (StartRow + MaxRows) LTE GetParks.RecordCount>
<a href="index.cfm?startrow=#Evaluate(StartRow + MaxRows)#">See next #MaxRows# rows</a>
</cfif>
</td>
</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.

Macromedia recommends that you use the cfqueryparam tag within every cfquery tag, to help secure your databases from unauthorized users. For more information, see:

- Chapter 20, “Accessing and Retrieving Data,” in Developing ColdFusion MX Applications

**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**
cfinser, 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 that ColdFusion passes to the right of the comparison operator in a <code>where</code> 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>
<td></td>
</tr>
<tr>
<td>CFSQIType</td>
<td>Optional</td>
<td>CF_SQL_CHAR</td>
<td>SQL type that parameter (any type) is bound to. • CF_SQL_BIGINT • CF_SQL_BIGINT • CF_SQL_BIT • CF_SQL_CHAR • CF_SQL_BLOB • CF_SQL_CLOB • CF_SQL_DATE • CF_SQL_DECIMAL • CF_SQL_DOUBLE • CF_SQL_FLOAT • CF_SQL_IDSTAMP • CF_SQL_INTEGER • CF_SQL_LONGVARCHAR • CF_SQL_LONGVARCHAR • CF_SQL_MONEY • CF_SQL_MONEY • CF_SQL_MONEY4 • CF_SQL_NUMERIC • CF_SQL_REAL • CF_SQL_REFCURSOR • CF_SQL_SMALLINT • CF_SQL_TIME • CF_SQL_TIMESTAMP • CF_SQL_TINYINT • CF_SQL_VARCHAR</td>
</tr>
<tr>
<td>maxLength</td>
<td>Optional</td>
<td></td>
<td>Length of string in <code>value</code> attribute. Maximum length of parameter.</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. • Yes: tag ignores the <code>value</code> attribute • No</td>
</tr>
<tr>
<td>list</td>
<td>Optional</td>
<td>No</td>
<td>• Yes: The <code>value</code> attribute value is a delimited list • 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 `cfqueryparam` in any SQL statement (for example, SELECT, INSERT, UPDATE, and DELETE) that uses ColdFusion variables.

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 quotation 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; it then lists the values beneath the query, in order of usage.

Example

```cfml
<!--- This example shows cfqueryparam with VALID input in Course_ID. --->
<h3>cfqueryparam Example</h3>
<cfset Course_ID = 12>
<cfquery name = "getFirst" dataSource = "cfsnippets">
   SELECT *
   FROM courses
   WHERE Course_ID = <cfqueryPARAM value = "#Course_ID#" CFSQLType = "CF_SQL_INTEGER">
</cfquery>
<cfoutput query = "getFirst">
   <p>Course Number: #Course_ID#<br> Description: #descript#</p>
</cfoutput>

<!--- This example shows the use of CFQUERYPARAM when INVALID string data is in Course_ID. ---->
<p>This example throws an error because the value passed in the CFQUERYPARAM tag exceeds the MAXLENGTH attribute</p>
<cfset LastName="Peterson; DELETE employees WHERE LastName='Peterson'">
<!--- Note that for string input you must specify the MAXLENGTH attribute for validation. ------------------------>
<cfquery
    name="getFirst" datasource="cfsnippets">
    SELECT *
    FROM employees
    WHERE LastName=<cfqueryparam
        value="#LastName#"
        cfsqltype="CF_SQL_VARCHAR"
        maxlength="17">
</cfquery>
<cfoutput
    query="getFirst"> <p>
    Course Number: #FirstName# #LastName#
    Description: #Department# </p>
</cfoutput>
**cfregistry**

**Description**
This tag is deprecated for the UNIX platform.
Reads, writes, and deletes keys and values in the system registry. Provides persistent storage of client variables.

**Note:** For this tag execute, it must be enabled in the ColdFusion Administrator. For more information, see 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 291
- `cfregistry action = "get"` on page 292
- `cfregistry action = "set"` on page 293
- `cfregistry action = "delete"` on page 294

**See also**
cfcookie, cfparam, cfsavecontent, cfschedule, cfset

**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
<cfregistry
action = "getAll"
branch = "branch"
type = "data type"
name = "query name"
sort = "criteria">

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>getAll</td>
<td>Name of a registry branch.</td>
</tr>
<tr>
<td>branch</td>
<td>Required</td>
<td></td>
<td>Name of record set to contain returned keys and values.</td>
</tr>
<tr>
<td>type</td>
<td>Optional</td>
<td>String</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: sort = &quot;value desc, entry asc&quot;</td>
</tr>
<tr>
<td>name</td>
<td>Required</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sort</td>
<td>Optional</td>
<td>ASC</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
<!--- This example uses cfregistry with the getAll Action --->
<cfregistry action = "getAll"
  branch = "HKEY_LOCAL_MACHINE\Software\Microsoft\Java VM"
  type = "Any"
  name = "RegQuery">
<p><h1>cfregistry action = "getAll"
</p><cftable query = "RegQuery" colHeaders HTMLTable border = "Yes">
  <cfcol header = "<b>Entry</b>" width = "35" text = "#RegQuery.Entry#">
  <cfcol header = "<b>Type</b>" width = "10" text = "#RegQuery.type#">
  <cfcol header = "<b>Value</b>" width = "35" text = "#RegQuery.Value#">
</cftable>
**cfregistry action = "get"**

**Description**
Accesses a registry value and stores it in a ColdFusion variable.

**Syntax**
```
<cfregistry
  action = "get"
  branch = "branch"
  entry = "key or value"
  variable = "variable"
  type = "data type">
```

**Attributes**

<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>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>
</tbody>
</table>
| type      | Optional| string  | • string: return string value  
           |          |   • dWord: return DWord value 
           |          |   • key: return key’s default value |

**Usage**
If the value does not exist, **cfregistry** does not create an entry.

**Example**
```
<!--- This example uses cfregistry with the Get Action --->
<cfregistry action = "get"
  branch = "HKEY_LOCAL_MACHINE\Software\Microsoft\Java VM"
  entry = "ClassPath" type = "String" variable = "RegValue">
<h1>cfregistry action = "get"</h1>
<cfoutput>
  <p>Java ClassPath value is #RegValue#</p>
</cfoutput>
```
**cfregistry action = "set"**

**Description**
Adds a registry key, adds a value, or updates a value.

**Syntax**
```cfc```
cfregistry
  action = "set"
  branch = "branch"
  entry = "key or value"
  type = "value type"
  value = "data"
```

**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>set</td>
<td></td>
</tr>
<tr>
<td>branch</td>
<td>Required</td>
<td></td>
<td>Name of a registry branch.</td>
</tr>
<tr>
<td>entry</td>
<td>Required</td>
<td></td>
<td>Key or value to set.</td>
</tr>
</tbody>
</table>
| type      | Optional|         | • string: set a string value (default).
|           |         |         | • dWord: set a DWord value. |
|           |         |         | • key: create a key. |
| value     | Optional|         | Value data to set. If you omit this attribute, cfregistry creates default value, as follows:
|           |         |         | • string: creates an empty string: ""
|           |         |         | • dWord: creates a value of 0 (zero) |

**Usage**

If it does not exist, cfregistry creates the key or value.

**Example**
```cfc```
<!--- This example uses cfregistry Set Action to modify registry value data -->
<!--- Normally you pass in a file name instead of setting one here. -->
cfset FileName = "dummy.cfm"
<cfregistry action = "set"
  branch = "HKEY_LOCAL_MACHINE\Software\cflangref"
  entry = "LastCFM01" type = "String" value = "#FileName#">
```

<h1>cfregistry action = "set"</h1>
**cfregistry action = "delete"**

**Description**
Deletes a registry key or value.

**Syntax**
```<cfregistry
    action = "delete"
    branch = "branch"
    entry = "keyorvalue"/>```

**Attributes**

<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>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.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- For value deletion: name of registry branch that contains value to delete.</td>
</tr>
<tr>
<td>entry</td>
<td>Required for</td>
<td>Value to</td>
<td></td>
</tr>
<tr>
<td></td>
<td>value deletion</td>
<td>delete</td>
<td></td>
</tr>
</tbody>
</table>

**Usage**
If you delete a key, `cfregistry` also deletes values and subkeys defined beneath it.

**Example**
```<cfregistry action = "delete"
    branch = "HKEY_LOCAL_MACHINE\Software\cflangref\tempkey"
    entry = "LastCFM01"/>
<h1>cfregistry action = "delete"</h1>```
**cfreport**

**Description**
Runs a predefined Crystal Reports report. Applies only to Windows systems. Uses the CFCRYSTAL.exe file to generate reports. Sets parameters in the Crystal Reports engine according to its attribute values.

**Category**
Extensibility tags

**Syntax**
```cfcollection, cfexecute, cfindex, cfobject, cfsearch, cfwddx```

**See also**

**History**
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 Administrator.

**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>Optional</td>
<td>Name of registered or native data source.</td>
<td></td>
</tr>
<tr>
<td>type</td>
<td>Optional</td>
<td>standard</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• standard (not valid for Crystal Reports 8.0)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• netscape</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• microsoft</td>
<td></td>
</tr>
<tr>
<td>timeout</td>
<td>Optional</td>
<td>Maximum time, in seconds, in which a connection must be made to a Crystal Report.</td>
<td></td>
</tr>
<tr>
<td>report</td>
<td>Required</td>
<td>Report path. Store Crystal Reports files in the same directories as ColdFusion page files.</td>
<td></td>
</tr>
<tr>
<td>orderBy</td>
<td>Optional</td>
<td>Orders results according to your specifications.</td>
<td></td>
</tr>
<tr>
<td>username</td>
<td>Optional</td>
<td>Username required for entry into database from which report is created. Overrides default settings for data source in ColdFusion Administrator.</td>
<td></td>
</tr>
</tbody>
</table>
This tag requires an end tag.

Example

<!--- This view-only example shows the use of cfreport --->
<h3>cfreport Tag</h3>
<p>cfreport lets reports from the Crystal Reports Professional report writer display through a ColdFusion interface. To run, the tag requires the name of the report. cfreport can also pass information to the report file displayed, to change the output conditions.

This example would run a report called "monthlysales.rpt" and pass it an optional filter condition to show only the information for a subset of the report.

<cfreport report="/reports/monthlysales.rpt">
    {Departments.Department} = 'International'
</cfreport>

<p>Substitute your report files and filters for this code. cfreport can put Crystal Reports into web pages.
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

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>
  <cfquery name = "GetMessages" dataSource = "cfsnippets">
    SELECT * 
    FROM Messages
  </cfquery>
  <cfcatch type = "DATABASE">
    <cfif cfcatch.sqlstate neq 50555>
      <cfrethrow>
    </cfif>
  </cfcatch>
</cftry>
<br>
<h3>Sorry, this request can't be completed</h3>
<br>
<h4>Catch variables</h4>
<cfoutput>
  <cfloop collection = #cfcatch# item = "c">
    <br>
    <cfif IsSimpleValue(cfcatch[c])>
      #c# = #cfcatch[c]#
    </cfif>
  </cfloop>
</cfoutput>
cfreturn

Description
Returns result values from a component method. Contains an expression returned as result of the function.

Return value
An expression; the result of the function from which this tag is called.

Category
Extensibility tags

Syntax
<cfreturn
  expr>

See also
cfargument, cfcomponent, cffunction, cfinvoke, cfinvokeargument, cfobject, cfproperty

History
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 11, “Building and Using ColdFusion Components,” in Developing ColdFusion MX Applications.

Example
<cffunction>
  <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>
</cffunction>
</cfcomponent>
cfsavecontent

Description
Saves the generated content of the `cfsavecontent` tag, including the results of evaluating expressions and executing custom tags, in the specified variable.

Category
Variable manipulation tags

Syntax
```<cfsavecontent variable = "variable name"> the content </cfsavecontent>```

See also
`cfcookie, cfparam, cfregistry, cfschedule, cfset`

Attributes

<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.

```<cfsavecontent variable="content"> <CF_OutputBigReport> </cfsavecontent> <cfoutput> #replace(content, "report", "MyCompany Quarterly Report", "all")# </cfoutput>```
cfschedule

Description
Provides a programmatic interface to the ColdFusion scheduling engine. 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
<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>

<cfschedule
action = "run"
task = "TaskName">
</cfschedule>

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.

See also
cfcookie, cfparam, cfregistry, cfsavecontent, cfset
## 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  
• run: executes the specified task |
| task        | Required|         | Name of the task.                                                                                                                            |
| operation   | Required if action = "update" |          | Operation that the scheduler performs. Must be HttpServletRequest.                                                                         |
| 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: save the result to a file  
• No                                                                                                                                            |
| endDate     | Optional |         | Date when scheduled task ends.                                                                                                                |
| endTime     | Optional |         | Time when scheduled task ends (seconds).                                                                                                     |
| interval    | Required if action = "update" | One hour | Interval at which task is scheduled.  
• number of seconds (minimum is 60)  
• once  
• daily  
• weekly  
• monthly |
| requestTimeOut | Optional |         | Can be used to extend the default timeout period.                                                                                           |
| username    | Optional |         | Username, if URL is protected.                                                                                                               |
| password    | Optional |         | Password, if URL is protected.                                                                                                               |
| proxyServer | Optional |         | Host name or IP address of a proxy server.                                                                                                   |
| proxyPort   | Optional | 80      | Port number to use on the proxy server.                                                                                                       |
| proxyUser   | Opt     |         | User name to provide to the proxy server.                                                                                                    |
Usage

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 \cfusion\log\schedule.log file.

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">
--->
```

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Req/Opt</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>proxyPassword</td>
<td>Opt</td>
<td></td>
<td>Password to provide to the proxy server.</td>
</tr>
</tbody>
</table>
| resolveURL      | Optional| No      | • Yes: resolve links in the output page to absolute references  
|                 |         |         | • No                                             |

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
<cfscript>
cfscript code here
</cfscript>

See also
cfinvoke, cfmodule, CreateObject, Chapter 6, “Extending ColdFusion Pages with CFML Scripting,” in Developing ColdFusion MX Applications

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 Developing ColdFusion MX Applications.

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:

```cfscript
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 Developing ColdFusion MX Applications.

**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 32, “Using Web Services,” in Developing ColdFusion MX Applications.

**Example**

<p>This simple example shows variable declaration and manipulation. <cfif IsDefined("form.myValue")>
<cfif IsNumeric(form.myValue)>
<cfset x = form.myValue>
<cfscript>
y = x;
z = 2 * y;
StringVar = form.myString;
</cfscript>
<cfoutput><p>twice #x# is #z#.
<p>Your string value was: <b><i>#StringVar#</i></b></cfoutput>
<cfelse>
**cfsearch**

**Description**

Searches Verity collections using ColdFusion or K2Server, whichever search engine a collection is registered by. (ColdFusion can also search collections that have not been registered, with the `cfcollection` tag.)

A collection must be created and indexed before this tag can return search results.

A collection can be created in these ways:

- With the `cfcollection` tag
- In the ColdFusion Administrator, which calls the `cfcollection` tag
- Externally, using a native Verity indexing tool, such as Vspider or MKVDK

A collection can be registered with ColdFusion in the following ways:

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

A collection can be registered with K2Server by editing the k2server.ini file.

A collection can be indexed in the following ways:

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

For more information, see Chapter 24, “Building a Search Interface,” in *Developing ColdFusion MX Applications*.

**Category**

Extensibility tags

**Syntax**

```html
<cfsearch
    name = "search_name"
    collection = "collection_name"
    type = "criteria"
    criteria = "search_expression"
    maxRows = "number"
    startRow = "row_number"
    language = "language">
</cfsearch>
```

**See also**

cfcollection, cfexecute, cfindex, cfobject, cfreport, cfwddx

**History**

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>Name of the search query.</td>
<td></td>
</tr>
<tr>
<td>collection</td>
<td>Required</td>
<td>One or more path(s) and/or registered collection name(s). For a registered collection, specify the collection name. For an unregistered collection, specify an absolute path. Registered names are listed in the ColdFusion Administrator, Verity Collections and Verity Server pages. To specify multiple collections, use a comma delimiter. For example: &quot;CFUSER, e:\collections\personnel&quot; If you specify multiple collections, you cannot include a combination of collections that are registered by K2Server and registered by Verity.</td>
<td></td>
</tr>
<tr>
<td>type</td>
<td>Optional</td>
<td>simple • simple: STEM and MANY operators are implicitly used. See Chapter 25, &quot;Using Verity Search Expressions,&quot; in Developing ColdFusion MX Applications. • explicit: operators must be invoked explicitly</td>
<td></td>
</tr>
<tr>
<td>criteria</td>
<td>Optional</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, &quot;Using Verity Search Expressions,&quot; in Developing ColdFusion MX Applications.</td>
<td></td>
</tr>
<tr>
<td>maxRows</td>
<td>Optional</td>
<td>All Maximum number of rows to return in query results. Use double or single quotation marks.</td>
<td></td>
</tr>
<tr>
<td>startRow</td>
<td>Optional</td>
<td>1 First row number to get.</td>
<td></td>
</tr>
<tr>
<td>language</td>
<td>Optional</td>
<td>english For options, see cfcollection on page 80. Requires the ColdFusion International Search Pack.</td>
<td></td>
</tr>
</tbody>
</table>

Usage

To permit application users to search Verity collections for non-standard strings, words or characters (for example, "AB23.45.67" or "--->") that would otherwise cause an error, you can create a text file that lists these elements and defines their formats for Verity. Name the file style.lex and put copies of the file in these directories:

• Windows:
  ■ cf_root\lib\common\style (typically, cf_root = c:\cfusionmx)
  ■ cf_root\lib\common\style\custom
Macromedia does not recommend using the **cflock** tag with this tag; Verity provides the locking function. Using the **cflock** tag slows search performance.

This tag returns a record set whose columns you can reference in a **cfoutput** tag. For example, the following code specifies a search for the exact terms “filming” or “filmed”:

```coldfusion
<cfsearch
    name = "mySearch"
    collection = "myCollection"
    criteria = '<WILDCARD>`film{ing,ed}`'
    type="explicit"
    startrow=1>
<cfdump var = "#mySearch#>
```

In this example, the single quotation mark (‘) and backtick (`) characters are used as delimiters; for more information, see Chapter 25, “Using Verity Search Expressions,” in *Developing ColdFusion MX Applications*.

### **cfsearch** result columns

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>url</td>
<td>Value of URLPath attribute in the <strong>cfindex</strong> tag used to populate a collection. If type = &quot;custom&quot;, the value is always empty when you populate a collection.</td>
</tr>
<tr>
<td>key</td>
<td>Value of the attribute in the <strong>cfindex</strong> tag used to populate collection</td>
</tr>
<tr>
<td>title</td>
<td>Value of title attribute in <strong>cfindex</strong> operation used to populate the collection, including PDF and Office document titles. If title is not provided, the tag uses the <strong>cfindex</strong> title 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</td>
<td>Value of custom fields in <strong>cfindex</strong> operation used to populate collection.</td>
</tr>
<tr>
<td>summary</td>
<td>Contents of automatic summary generated by <strong>cfindex</strong>, Default: best three matching sentences, up to 500 characters.</td>
</tr>
<tr>
<td>recordCount</td>
<td>Number of records returned in record set</td>
</tr>
<tr>
<td>currentRow</td>
<td>Current row that <strong>cfoutput</strong> 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</td>
</tr>
</tbody>
</table>
You can use query result columns in standard CFML expressions, preceding the result column name with the name of the query, as follows:

```cfml
#DocSearch.url#
#DocSearch.key#
#DocSearch.title#
#DocSearch.score#
```

**Example**

```
<!--- #1 (TYPE=SIMPLE) -----------------------------
<cfsearch
  name="name"
  collection="snippets,syntax,snippets"
  criteria="example"
> -->
<p>
<cfoutput>Search Result total = #name.RecordCount# </cfoutput><br>
<cfoutput>
  url=#name.url#<br>
  key=#name.key#<br>
  title=#name.title#<br>
  score=#name.score#<br>
  custom1=#name.custom1#<br>
  custom2=#name.custom2#<br>
  summary=#name.summary#<br>
  recordcount=#name.recordcount#<br>
  currentrow=#name.currentrow#<br>
  columnlist=#name.columnlist#<br>
  recordssearched=#name.recordssearched#<br>
</cfoutput>
<cfdump var = #name#>
<br>
<!--- #2 (TYPE=EXPLICIT) -----------------------------
<cfsearch
  name = "snippets"
  collection = "snippets"
  criteria = '<wildcard>`film{ing,ed}`'
  type="explicit"
  startrow=1>
<cfoutput
query="snippets">
  url=#url#<br>
  key=#key#<br>
  title=#title#<br>
  score=#score#<br>
  custom1=#custom1#<br>
  custom2=#custom2#<br>
  summary=#summary#<br>
  recordcount=#recordcount#<br>
  currentrow=#currentrow#<br>
  columnlist=#columnlist#<br>
  recordssearched=#recordssearched#<br>
</cfoutput>
<cfdump var = #snippets#>
<br>
<!--- #3 (search by CF key) -----------------------------
<cfsearch
  name = "book"
  collection = "custom_book"
criteria = "cf_key=bookid2"
<cfoutput>
  url=#book.url#<br>
  key=#book.key#<br>
  title=#book.titleE#<br>
  score=#book.score#<br>
  custom1=#book.custom1#<br>
  custom2=#book.custom2#<br>
  summary=#book.summary#<br>
  recordcount=#book.recordcount#<br>
  currentrow=#book.currentrow#<br>
  columnlist=#book.columnlist#<br>
  recordssearched=#book.recordssearched#<br>
</cfoutput>
<cfdump var = #book#>
<br>
cfselect

Description
Constructs a drop-down list box form control. Used within a cfform tag.
You can populate the list from a query, or by using the HTML option tag.

Category
Forms tags

Syntax
<cfselect
    name = "name"
    required = "Yes" or "No"
    message = "text"
    onError = "text"
    size = "integer"
    multiple = "Yes" or "No"
    query = "queryname"
    selected = "column_value"
    value = "text"
    display = "text"
    passThrough = "HTML_attributes">
</cfselect>

See also
cfapplet, cfform, cfgrid, cfinput, cfgridcolumn, cfgridrow, cfgridupdate, cfslider, cftextinput, cftree, cftreeitem

Attributes

<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 select form element</td>
</tr>
<tr>
<td>size</td>
<td>Optional</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</td>
<td>No</td>
<td>• Yes: a list element must be selected when 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. • No</td>
</tr>
<tr>
<td>message</td>
<td>Optional</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</td>
<td></td>
<td>Custom JavaScript function to execute if validation fails</td>
</tr>
<tr>
<td>multiple</td>
<td>Optional</td>
<td>No</td>
<td>• Yes: allow selecting multiple elements in drop-down list • No</td>
</tr>
<tr>
<td>query</td>
<td>Optional</td>
<td></td>
<td>Name of query to populate drop-down list.</td>
</tr>
<tr>
<td>selected</td>
<td>Optional</td>
<td></td>
<td>A list of option values to preselect in the selection list. This attribute applies only if list items are generated from a query. The cfform preservadata attribute value can override this value.</td>
</tr>
</tbody>
</table>
In addition to the listed attributes, you can use the following HTML attributes in the `cfform` tag without using the `passThrough` attribute. The tag does not use these attributes, but includes them in the HTML of the `form` tag that it generates and returns to the browser:

- `class`
- `id`
- `onBlur`
- `onChange`
- `onClick`
- `onDblclick`
- `onFocus`
- `style`
- `tabIndex`

Usage

To ensure that a selected list box item persists across postbacks, use the `cfform preserveData` attribute with a list generated from a query. (This strategy works only with data that is populated from a query.)

If the `cfform preserveData` attribute is true and the form posts back to the same page, and if the control is populated by a query, the posted selection(s) for the `cfselect` control are used instead of the `Selected` attribute. For controls that are populated with regular HTML option tags, the developer must dynamically add the `Selected` attribute to the appropriate option tag(s).

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

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

If you put a value in quotation marks, you must escape them; for example:

```plaintext
passThrough = "readonly = " "yes"
```

For more information, see the `cfform` tag entry.

This tag requires an end tag.
Example

<!--- This example shows the use of cftree, cfselect and cfgrid in a cfform. The query takes a list of employees, and uses cftree and cfselect to display results of query. cfgrid is used to show an alternate means of displaying the data -->
<!--- set a default for the employeeNames variable -->
<cfparam name = "employeeNames" default = "">
<!--- if an employee name has been passed from the form, set employeeNames variable to this value -->
<cfif IsDefined("form.employeeNames") is not "False">
<cfset employeeNames = form.employeeNames>
</cfif>

<!--- query the datasource to find the employee information-->
<cfquery name = "GetEmployees" dataSource = "cfsnippets"> SELECT Emp_ID, FirstName, LastName, EMail, Phone, Department FROM Employees WHERE 0=0
<cfif employeeNames is not ""> AND LastName = '#employeeNames#'
</cfif>
</cfquery>

<h3>cfselect Example</h3>
<!--- Use cfform when using other cfinput tools -->
<cfform action = "cfselect.cfm">
<!--- Use cfselect to present the contents of the query by column -->
<h3>cfselect Presentation of Data</h3>
<h4>Click on an employee's last name and hit "see information for this employee" to see expanded information.</h4>
<cfselect name = "EmployeeNames" message = "Select an Employee Name" size = "#GetEmployees.recordCount#" query = "GetEmployees" value = "LastName" required = "No">
<option value = "">Select All</option>
<input type="Submit" name="" value="see information for this employee">
</cfselect>

<!--- showing the use of cftree ---------------------------------------->
<!--- use cftree for an expanded presentation of the data Loop through the query to create each branch of the CFTREE ----------------------------->
<h3>cftree Presentation of Data</h3>
<h4>Click on the folders to "drill down" and reveal information.</h4>
<cftree name="SeeEmployees" height="150" width="240" font="Arial Narrow" bold="No" italic="No" border="Yes" hscroll="Yes" vscroll="Yes" required="No" completpath="No" appendkey="Yes" highlightref="Yes">
<cfloop query="GetEmployees">
<cftreecell value="#Emp_ID#" parent="SeeEmployees" expand="No">
<cftreecell value="#LastName#" display="Name" parent="#Emp_ID#" queryasroot="No" expand="No">
<cftreecell value="#FirstName#, #LastName#" parent="#LastName#" expand="No" queryasroot="No">
<cftreecell value="#Department#" display="Department" parent="#Emp_ID#" queryasroot="No" expand="No">
<cftreecell value="#Department#" parent="#Department#"
<cfselect>
<cfselectitem value="#Phone#" display="Phone" parent="#Emp_ID#" expand="No" queryasroot="No">
<cfselectitem value="#Phone#" parent="#Phone#" expand="No" queryasroot="No">
<cfselectitem value="#Email#" display="Email" parent="#Emp_ID#" queryasroot="No" expand="No">
<cfselectitem value="#Email#" parent="#Email#" expand="No" queryasroot="No">
</cfloop>
</cftree>

<!----- You can also use CFGRID for a more comprehensive, quicker view at the data -->
<h3>CFGRID Presentation of Data</h3>
<cfgrid name="SampleGrid" width="600" query="GetEmployees" insert="No" delete="No" sort="No" font="Verdana" bold="No" italic="No" appendkey="No" highlightref="No" griddataalign="LEFT" gridlines="Yes" rowheaderbold="No" rowheaderitalic="No" colheaderbold="Yes" colheaderalign="CENTER" colheaderitalic="No" colheaderbold="No" bgcolor="Teal" selectmode="BROWSE" picturebar="No">
<cfgridcolumn name="LastName" header="Last Name" headeralign="LEFT" dataalign="LEFT" bold="No" italic="No" select="Yes" display="Yes" headeritalic="No" headerbold="No">
<cfgridcolumn name="FirstName" header="First Name" headeralign="LEFT" dataalign="LEFT" bold="No" italic="No" select="No" display="Yes" headeritalic="No" headerbold="No">
<cfgridcolumn name="Email" header="Email" headeralign="LEFT" dataalign="LEFT" bold="No" italic="Yes" select="No" display="Yes" headeritalic="No" headerbold="No">
<cfgridcolumn name="Phone" header="Phone" headeralign="LEFT" dataalign="LEFT" bold="No" italic="Yes" select="No" display="Yes" headeritalic="No" headerbold="No">
<cfgridcolumn name="Department" header="Department" headeralign="LEFT" dataalign="LEFT" bold="Yes" italic="No" select="No" display="Yes" headeritalic="No" headerbold="No">
<cfgridcolumn name="Emp_ID" header="ID" headeralign="LEFT" dataalign="LEFT" width="40" bold="No" italic="No" select="No" display="Yes" headeritalic="No" headerbold="No">
</cfgrid>
</cfgridform>
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:

```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.
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:

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**

```xml
<cfset
   var variable_name = expression
>
```

**See also**
cfcookie, cfparam, cfregistry, cfsavecontent, cfschedule

**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>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>
<td></td>
</tr>
<tr>
<td>variable_name</td>
<td>Required</td>
<td>Variable</td>
<td></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:

```xml
<cfset StructDelete(Application, "MyVariable")>
```

**Arrays**

The following example assigns a new array to the variable `months`:

```xml
<cfset months = ArrayNew(1)>
```

This example creates a variable `Array_Length` that resolves to the length of the array `Scores`:

```xml
<cfset Array_Length = ArrayLen(Scores)>
```

This example assigns, to index position two in the array `months`, the value `February`:

```xml
<cfset months[2] = "February">
```

**Dynamic variable names**

In this example, the variable name is itself a variable:

```xml
<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:

```coldfusion
<cffunction name="myFunct">
    <cfset var myVar = "This is a test">
    <cfreturn myVar & " Message."/>
</cffunction>
<cfoutput><cfoutput>
</cfoutput><cfoutput>&myFunct()&</cfoutput>
</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.

```coldfusion
<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>
</cfobject>
```

Example

```coldfusion
<!--- This example shows how to use cfset --->
<cfquery name = "GetMessages" dataSource = "cfsnippets">
    SELECT * 
    FROM   Messages 
</cfquery>

<h3>cfset Example</h3>
<p>cfset sets and reassigns values to local or global variables within a page.

    <cfset NumRecords = GetMessages.recordCount>
    <p>For example, the variable NumRecords has been declared on this page to hold the number of records returned from query
    <cfoutput>&NumRecords#</cfoutput>.
```
In addition, cfset can be used to pass variables from other pages, such as this example, which takes the url parameter Test from this link:

<a href="cfset.cfm?test = "<cfoutput>
#URLEncodedFormat("hey, you, get off of my cloud")#</cfoutput>
"click here"></a>) to display a message:

<p>
<cfif IsDefined("url.test") is "True">
<cfoutput><b><I>#url.test#</i></b></cfoutput>
<cfelse>
<h3>The variable url.test has not been passed from another page.</h3>
</cfif>
</cfif>

cfset can also be used to collect environmental variables, such as the time, the IP address of the user, or another function or expression.

<cfset the_date = #DateFormat(Now())# & " " & #TimeFormat(Now())#>
<cfset user_ip = CGI.REMOTE_ADDR>
<cfset complex_expr = (23 MOD 12) * 3>
<cfset str_example = Reverse(Left(GetMessages.body, 35))>

<cfoutput>
<ul>
<li>The date: #the_date#
<li>User IP Address: #user_ip#
<li>Complex Expression ((23 MOD 12) * 3): #complex_expr#
<li>String Manipulation (the first 35 characters of the body of the first message in our query)
<br>[Reversed]: #str_example#
<br>[Normal]: #Reverse(str_example)#
</ul>
</cfoutput>
cfsetting

Description
Controls aspects of page processing, such as the output of HTML code in pages.

Category
Page processing tags, Variable manipulation tags

Syntax
<cfsetting
   enableCFoutputOnly = "Yes" or "No"
   showDebugOutput = "Yes" or "No"
   requestTimeOut = "value in seconds" >

See also
cfcache, cfflush, cfheader, cfhtmlhead, cfinclude, cfsilent

History
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</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• 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</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• No: suppresses debugging information that would otherwise display at end of 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.

**Note:** If the debugging service is enabled and `showDebugOutput = "Yes"`, the `IsDebugMode` function returns `Yes`; otherwise, `No`. ColdFusion MX 6.1 allows a `</cfsetting>` end tag; however, this end tag does not effect 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>
<p>This text is shown
<cfsetting enableCFoutputOnly = "No">
  <cfoutput>
    <p>Text within cfoutput is always shown
  </cfoutput>
<cfsetting enableCFoutputOnly = "Yes">
  <cfoutput>
    <p>Text within cfoutput is always shown
  </cfoutput>
```
cfsilent

Description
Suppresses output produced by CFML within a tag's scope.

Category
Data output tags, Page processing tags

Syntax
<cfsilent>
   ...
</cfsilent>

See also
cfcache,cfflush,cfheader,cfhtmlhead,cfinclude,cfsetting

Usage
This tag requires an end tag.

Example
<h3>cfsilent</h3>

<cfsilent>
   <cfset a = 100>
   <cfset b = 99>
   <cfset c = b-a>
   <cfoutput>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 `cfform` tag.

**Category**

Forms tags

**Syntax**

```cfml
<cfslider
    name = "name"
    label = "text"
    refreshLabel = "Yes" or "No"
    range = "min_value, max_value"
    scale = "uinteger"
    value = "integer"
    onValidate = "script_name"
    message = "text"
    onError = "text"
    height = "integer"
    width = "integer"
    vSpace = "integer"
    hSpace = "integer"
    align = "alignment"
    tickMarkMajor = "Yes" or "No"
    tickMarkMinor = "Yes" or "No"
    tickMarkImages = "URL1, URL2, URLn"
    tickMarkLabels = "Yes" or "No" or "list"
    lookAndFeel = "motif" or "windows" or "metal"
    vertical = "Yes" or "No"
    bgColor = "color"
    textColor = "color"
    font = "font_name"
    fontSize = "integer"
    italic = "Yes" or "No"
    bold = "Yes" or "No"
    notSupported = "text">
</cfslider>
```

**See also**

cfapplet, cfinput, cfform, cfselect, cftextinput, cftree, cftreeitem

**History**

ColdFusion MX: Deprecated the `img`, `imgStyle`, and `grooveColor` 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 <code>cfslider</code> 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;$value&quot;. If &quot;$&quot; is omitted, slider value displays directly after label.</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>
| refreshLabel    | Optional| Yes      | • Yes: when user moves slider, label is refreshed  
• No                                                                                                                                 |
| range           | Optional| "0,100" | Numeric slider range values. Separate values with a comma.                                                                                     |
| scale           | Optional|          | Unsigned integer. Defines slider scale, within range. For example, if range = "0,1000" and scale = "100", the display values are:  
0, 100, 200, 300, ...  
Signed and unsigned integers in ColdFusion are in the range -2,147,483,648 to 2,147,483,647.                                         |
| value           | Optional| Minimum  | Starting slider setting. Must be within the range values.                                                                                        |
| onValidate      | Optional|          | Custom JavaScript function to validate user input; in this case, a change to the default slider value. Specify only the function name.       |
| message         | Optional|          | Message text to appear if validation fails.                                                                                                                                                               |
| onError         | Optional|          | Custom JavaScript function to execute if validation fails. Specify only the function name.                                                                                                                |
| height          | Optional| 40       | Slider control height, in pixels.                                                                                                                                                                         |
| width           | Optional|          | Slider control width, in pixels.                                                                                                                                                                          |
| vSpace          | Optional|          | Vertical spacing above and below slider, in pixels.                                                                                                                                                       |
| hSpace          | Optional|          | Horizontal spacing to left and right of slider, in pixels.                                                                                                                                               |
| align           | Optional|          | Alignment of slider  
• top  
• left  
• bottom  
• baseline  
• texttop  
• absbottom  
• middle  
• absmiddle  
• right                                                                                                                                 |
| tickMarkMajor   | Optional| No       | • Yes: render major tickmarks in slider scale.  
• No: no major tickmarks. Major tick marks display at intervals specified by scale.                                                        |
| tickMarkMinor   | Optional| No       | • Yes: render minor tickmarks in slider scale.  
• No: no minor tickmarks Minor tickmarks display between major tickmarks.                                                                          |
<p>| tickMarkImages  | Optional|          | Comma-delimited list of URLs specifying images in slider tickmark scale. If you do not specify enough values, the last value is repeated for remaining tickmarks. If you specify too many values, extra values are ignored. |</p>
<table>
<thead>
<tr>
<th>Attribute</th>
<th>Req/Opt</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tickMarkLabels</td>
<td>Optional</td>
<td>No</td>
<td>• yes: numeric tickmarks based on the value of the range and scale attributes.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• no: prevents label text from displaying</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Comma-delimited list of strings for tickmark labels; for example, &quot;ten, twenty, thirty, forty&quot;.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>If you do not specify enough values, the last value is repeated for remaining tickmarks. If you specify too many values, extra values are ignored.</td>
</tr>
<tr>
<td>lookAndFeel</td>
<td>Optional</td>
<td>Windows</td>
<td>• motif: renders slider using Motif style</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• windows: renders slider using Windows style</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• metal: renders slider using Java Swing style</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>If platform does not support choice, the tag defaults to the platform’s default style.</td>
</tr>
<tr>
<td>vertical</td>
<td>Optional</td>
<td>No</td>
<td>• Yes: renders slider in browser vertically. You must set width and height attributes; ColdFusion does not automatically swap width and height values.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• No: renders slider horizontally.</td>
</tr>
<tr>
<td>bgColor</td>
<td>Optional</td>
<td></td>
<td>Background color of slider label.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>For a hex value, use the form: bgColor = &quot;##xxxxxx&quot;, where x = 0-9 or A-F; use two pound signs or none.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Any color, in hex format</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• black</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• red</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• blue</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• magenta</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• cyan</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• orange</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• darkgray</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• pink</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• gray</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• white</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• lightgray</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• yellow</td>
</tr>
<tr>
<td>textColor</td>
<td>Optional</td>
<td>Options</td>
<td>same as for bgColor attribute</td>
</tr>
<tr>
<td>font</td>
<td>Optional</td>
<td></td>
<td>Font name for label text.</td>
</tr>
<tr>
<td>fontSize</td>
<td>Optional</td>
<td></td>
<td>Font size for label text, in points.</td>
</tr>
<tr>
<td>italic</td>
<td>Optional</td>
<td>No</td>
<td>• Yes: italics label text</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• No: normal text</td>
</tr>
<tr>
<td>bold</td>
<td>Optional</td>
<td>No</td>
<td>• Yes: bold label text</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• No: medium text</td>
</tr>
</tbody>
</table>
| 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: *
|                     |          |         | '&lt;b&gt;Browser must support Java to view ColdFusion Java Applets&lt;/b&gt;'<n>Default message: '<n><b>B</b>rowser must support Java to view ColdFusion Java Applets</b>' |

326 Chapter 2: ColdFusion Tags
**Usage**

This tag requires the client to download a Java applet. Using this tag may be slightly slower than using an HTML form element to display the same information.

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

If the following conditions are true, a user's selection from query data that populates this tag's options continues to display after the user submits the form:

- The `cfform preserveData` attribute is set to "Yes"
- The `cfform action` attribute posts to the same page as the form itself (this is the default), or the action page has a form that contains controls with the same names as corresponding controls on the user entry form

For more information, see the `cfform` tag entry.

**Example**

```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
    <cfstoredproc
        procedure = "procedure name"
        dataSource = "ds_name"
        username = "username"
        password = "password"
        blockFactor = "blocksize"
        debug = "Yes" or "No"
        returnCode = "Yes" or "No">
    </cfstoredproc>

See also
cfinsert, cfqueryparam, cfprocparam, cfprocresult, cftransaction, cfquery, 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. (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>
<tr>
<td>debug</td>
<td>Optional</td>
<td>No</td>
<td>• Yes: Lists debug information on each statement</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• No</td>
</tr>
<tr>
<td>returnCode</td>
<td>Optional</td>
<td>No</td>
<td>• Yes: Tag populates cfstoredproc.statusCode with status code returned by stored procedure.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• No</td>
</tr>
</tbody>
</table>

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. 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 `cfstoredproc.statusCode`, which
holds the status code for a stored procedure. Status code values vary by DBMS. For the meaning
of code values, see your DBMS documentation.

This tag sets the variable `cfstoredproc.ExecutionTime`, which contains the execution time of
the stored procedure, in milliseconds.

Before implementing this tag, ensure that you understand stored procedures and their usage.

The following examples use a Sybase stored procedure; for an example of an Oracle 8 stored
procedure, see **cfprocparam**.

**Example**

<!--- This view-only example executes a Sybase stored procedure that
returns three result sets, two of which we want. The stored
procedure returns the status code and one output parameter,
which we display. We use named notation for the parameters. --->

<!---
<cfstoredproc procedure = "foo_proc"
dataSource = "MY_SYBASE_TEST" username = "sa"
password = "" dbServer = "scup" dbName = "pubs2"
returnCode = "Yes" debug = "Yes">
  <cfprocparam type = "IN" CFSQLType = CF_SQL_INTEGER
  value = "1" dbVarName = @param1>
  <cfprocparam type = "OUT" CFSQLType = CF_SQL_DATE
  variable = "FOO" dbVarName = @param2>
  <cfprocresult name = RS1>
  <cfprocresult name = RS3 resultSet = 3>
    <cfprocparam type = "IN" CFSQLType = CF_SQL_INTEGER
    value = "I" dbVarName = @param1>
    <cfprocparam type = "OUT" CFSQLType = CF_SQL_DATE
    value = "FOO" dbVarName = @param2>
  </cfstoredproc>
  --->

<!---
<cfoutput>The output param value: '#foo#'<br></cfoutput>
<h3>The Results Information</h3>
<cfoutput query = RS1>#name#,#DATE_COL#<br></cfoutput><p>
<cfoutput>
  <hr>
  <p>Record Count: #RS1.recordCount# <p>Columns: #RS1.columnList#<hr>
</cfoutput>
<cfoutput query = RS3>#col1#,#col2#,#col3#<br>
</cfoutput><p>
<cfoutput>
  <hr>
  <p>Record Count: #RS3.recordCount# <p>Columns: #RS3.columnList#<hr>
The return code for the stored procedure is: '#cfstoredproc.statusCode#'<br>
</cfoutput>
--->
**cfswitch**

**Description**
Evaluates a passed expression and passes control to the cfcase tag that matches the expression result. You can, optionally, code a cfdefaultcase tag, which receives control if there is no matching cfcase tag value.

**Category**
Flow-control tags

**Syntax**
```xml
<cfswitch expression = "expression">
  one or more cfcase tags
  zero or one cfdefaultcase tags
</cfswitch>
```

**See also**
cfcase, cfdefaultcase
also cfabort, cffloop, cfbreak, cfthrow, cfexecute, cfexit, cfthrow, cfif, cftry, cflocation

**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**
```xml
<!--- This example shows the use of cfswitch and cfcase to exercise a case statement in CFML --->
<cfquery name = "GetEmployees" dataSource = "cfsnippets">
  SELECT Emp_ID, FirstName, LastName, EMail, Phone, Department
  FROM   Employees
</cfquery>

<h3>cfswitch Example</h3>
```
<p>By outputting the query and using cfswitch, we classify the output without using a cfloop construct. --->
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
<cfoutput
query = "query_name"
maxRows = "maxrows_table"
colSpacing = "number_of_spaces"
headerLines = "number_of_lines"
HTMLTable
border
colHeaders
startRow = "row_number">
...
</cfoutput>

See also
cfcol, cfoutput, cffcontent, cfprocessingdirective, cflog, cftable

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**

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

<html>
<body>
<h3>cftable Example</h3>

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

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Req/Opt</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>colHeaders</td>
<td>Optional</td>
<td></td>
<td>Displays column heads. If you use this attribute, you must also use the <code>cfcol</code> tag <code>header</code> attribute to define them. If you use this attribute (regardless of its value), ColdFusion displays column heads.</td>
</tr>
<tr>
<td>startRow</td>
<td>Optional</td>
<td>1</td>
<td>The query result row to put in the first table row.</td>
</tr>
</tbody>
</table>
cftextinput

Description
Puts a single-line text entry box in a cfform tag and controls its display characteristics.

Category
Forms tags

Syntax
```
<cftextinput
   name = "name"
   value = "text"
   required = "Yes" or "No"
   range = "min_value, max_value"
   validate = "data_type"
   pattern = "Java regular expression"
   onValidate = "script_name"
   message = "text"
   onError = "text"
   size = "integer"
   font = "font_name"
   fontSize = "integer"
   italic = "Yes" or "No"
   bold = "Yes" or "No"
   height = "integer"
   width = "integer"
   vSpace = "integer"
   hSpace = "integer"
   align = "alignment"
   bgColor = "color"
   textColor = "color"
   maxLength = "integer"
   notSupported = "text"/>
```

See also
cfapplet, cfform, cfgrid, cfgridcolumn, cfgridrow, cfgridupdate, cfinput, cfselect, cfslider, cftree, cftreeitem

History
ColdFusion MX 6.1: Changed the validate = "creditcard" option requirements: the text entry must have 13-16 digits.

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 cftextinput control.</td>
</tr>
<tr>
<td>value</td>
<td>Optional</td>
<td></td>
<td>Initial value to display in text control.</td>
</tr>
</tbody>
</table>
| required  | Optional| No      | • Yes: the user must enter or change text  
                    • No |
<p>| range     | Optional|         | Minimum–maximum value range, delimited by a comma. Valid only for numeric data. |</p>
<table>
<thead>
<tr>
<th>Attribute</th>
<th>Req/Opt</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>validate</td>
<td>Optional</td>
<td></td>
<td>• date: verifies format mm/dd/yy.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• eurodate: verifies date format dd/mm/yyyy.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• time: verifies time format hh:mm:ss.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• float: verifies floating point format.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• integer: verifies integer format.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• telephone: verifies telephone format ###-###-####. The separator can be a blank.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• zip code: verifies, in U.S. formats only, 5- or 9-digit format ####-####. The separator can be a blank.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• creditcard: strips blanks and dashes; verifies number using mod10 algorithm. Number must have 13-16 digits.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• social_security_number: verifies format ###-##-####. The separator can be a blank.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• regular_expression: matches input against pattern attribute.</td>
</tr>
<tr>
<td>onValidate</td>
<td>Optional</td>
<td></td>
<td>Custom JavaScript function to validate user input. The form object, input object, and input object value are passed to routine, which should return True if validation succeeds, False otherwise. The validate attribute is ignored.</td>
</tr>
<tr>
<td>pattern</td>
<td>Required if validate = &quot;regular_expression&quot;</td>
<td></td>
<td>JavaScript regular expression pattern to validate input. Omit leading and trailing slashes.</td>
</tr>
<tr>
<td>message</td>
<td>Optional</td>
<td></td>
<td>Message text to display if validation fails.</td>
</tr>
<tr>
<td>onError</td>
<td>Optional</td>
<td></td>
<td>Custom JavaScript function to execute if validation fails.</td>
</tr>
<tr>
<td>size</td>
<td>Optional</td>
<td></td>
<td>Number of characters displayed before horizontal scroll bar displays.</td>
</tr>
<tr>
<td>font</td>
<td>Optional</td>
<td></td>
<td>Font name for text.</td>
</tr>
<tr>
<td>fontSize</td>
<td>Optional</td>
<td></td>
<td>Font size for text.</td>
</tr>
<tr>
<td>italic</td>
<td>Optional</td>
<td>No</td>
<td>• Yes: italics text</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• No: normal text</td>
</tr>
<tr>
<td>bold</td>
<td>Optional</td>
<td>No</td>
<td>• Yes: bold text</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• No: medium text</td>
</tr>
<tr>
<td>height</td>
<td>Optional</td>
<td>40</td>
<td>Height of the control, in pixels.</td>
</tr>
<tr>
<td>width</td>
<td>Optional</td>
<td>200</td>
<td>Width of the control, in pixels.</td>
</tr>
<tr>
<td>vSpace</td>
<td>Optional</td>
<td></td>
<td>Vertical spacing of the control, in pixels.</td>
</tr>
<tr>
<td>hSpace</td>
<td>Optional</td>
<td></td>
<td>Horizontal spacing of the control, in pixels.</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 the following conditions are true, a user’s selection from query data that populates this tag’s options continues to display after the user submits the form:

- The cfform preserveData attribute is set to "Yes"
• The cfform action attribute posts to the same page as the form itself (this is the default), or the action page has a form that contains controls with the same names as corresponding controls on the user entry form.

For more information, see cfform.

If the cfform preserveData attribute is "yes", and the form posts back to the same page, the posted value (not the value of the value attribute) of the cftextinput control is used.

Example

```html
<h3>cftextinput Example</h3>
cftextinput provides simple validation for text fields in cfform and control over font information displayed in cfform input boxes for text. For example, the field below must not be blank, and provides a client-side message upon erring.

<cfform action = "cftextinput.cfm" method = "post">
  <cfif IsDefined("form.myInput")>
    <h3>You entered <cfoutput>#form.myInput#</cfoutput> into the text box </h3>
  </cfif>
  <cftextinput name = "myInput"
    font = "Courier" fontSize = 12
    value = "Look, this text is red!"
    textColor = "FF0000"
    message = "This field must not be blank"
    required = "Yes">
    <input type = "Submit" name = "" value = "submit">
  </cftextinput>
</cfform>
```
cfthrow

Description
Throws a developer-specified exception, which can be caught with a cfcatch tag that has any of the following type attribute options:

- type = "custom_type"
- type = "Application"
- type = "Any"

Category
Exception handling tags, Flow-control tags

Syntax 1
<cfthrow
type = "exception_type"
message = "message"
detail = "detail_description"
errorCode = "error_code"
extendedInfo = "additional_information"
object = "java_except_object">

Syntax 2
<cfthrow
object = #object_name#>

See also
cferror, cfrethrow, cftry, Chapter 14, “Handling Errors,” in Developing ColdFusion MX Applications

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>
</table>
| type      | Optional| Application | • A custom type
|           |         |          | • Application
|           |         |          | Do not enter another predefined type; types are not generated by ColdFusion applications. If you specify Application, you need not specify a type for cfcatch. |
| message   | Optional|          | Message that describes exception event. |
| detail    | Optional|          | Description of the event. ColdFusion appends error position to description; server uses this parameter if an error is not caught by your code. |
| errorCode | Optional|          | A custom error code that you supply. |
| extendedInfo | Optional|          | A custom error code that you supply. |
| object    | Optional|          | Requires the value of the cfobject tag name attribute. Throws a Java exception from a CFML tag. This attribute is mutually exclusive with all other attributes of this tag. |
Usage
Use this tag within a cftry block, to throw an error. The cfcatch block can access accompanying information, as follows:

- Message, with cfcatch.message
- Detail, with cfcatch.detail
- Error code, with cfcatch.errorcode

To get more information, use cfcatch.tagContext. This 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):

```
<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.

```
<cfset obj.init("You must save your work before preceding")>
```

You can then use the, the cfthrow statement to throw the exception as follows:

```
<cfthrow object=#obj#>
```

For more information on using Java objects in ColdFusion, see Chapter 33, “Integrating J2EE and Java Elements in CFML Applications,” in Developing ColdFusion MX Applications.

Example1

```
<h3>cfthrow Example</h3>
<!--- open a cftry block --->
<cftry>
  <!--- define a condition upon which to throw the error --->
  <cfif NOT IsDefined("URL.myID")>
    <!--- throw the error --->
    <cfthrow message = "ID is not defined">
  </cfif>

  <!--- perform the error catch --->
  <cfcatch type = "application">
    <!--- display your message --->
    <h3>You’ve Thrown an &lt;b&gt;Error&lt;/b&gt;&lt;/h3>
    <cfoutput>
      <!--- and the diagnostic feedback from the application server --->
      &lt;p&gt;#cfcatch.message#&lt;/p&gt;
    </cfoutput>
    <cfloop
      index = i
      from = 1 to = #ArrayLen(cfcatch.tagContext)#
      <cfset sCurrent = #cfcatch.tagContext[i]#>
      <br/#i# #sCurrent["ID"]#
  </cfcatch>
</cftry>
```
Example2

The following example shows how to throw an exception from a component method:

```coldfusion
<cfcomponent>
<cffunction name="getEmp">
<cfargument name="lastName" required="yes">
<cfquery name="empQuery" datasource="ExampleApps" >
SELECT LASTNAME, FIRSTNAME, EMAIL
FROM tblEmployees
WHERE LASTNAME LIKE '##arguments.lastName##'
</cfquery>
<cfif empQuery.recordcount LT 1>
<cfthrow type="noQueryResult"
message="No results were found. Please try again.">
<cfelse>
<cfreturn empQuery>
</cfif>
</cffunction>
</cfcomponent>

For an explanation of the example and more information, see Chapter 11, “Building and Using ColdFusion Components,” in Developing ColdFusion MX Applications.
**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 Administrator. Optionally, to report trace summaries, enable the Trace section.

**Category**
Debugging tags, Variable manipulation tags

**Syntax**
```cftag
<cftrace
    abort = "Yes or No"  
category = "string"
    inline = "Yes or No"
    text = "string"
    type = "format"
    var = "variable_name"
</cftrace>
```

**See also**
`cfdump`, `cferror`, `cfrethrow`, `cftry`, Chapter 18, “Debugging and Troubleshooting Applications,” in *Developing ColdFusion MX Applications*

**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, addition to the debugging information output.  
|           |         |         | • No        |
| text      | Optional|         | User-defined string, which can include simple variable, but not complex variables such as arrays. Outputs to `cflog` `text` attribute. |

---

 cftrace 341
**Usage**

You cannot put application code within this tag. (This avoids problems that can occur if you disable debugging.)

This tag is useful for debugging CFML code during application development.

You can display `cftrace` tag output in the following ways:

- As a section in the debugging output
- In-line in an application page, and as a section in debugging output. If you specify in-line tracing, ColdFusion flushes all output up to the `cftrace` tag, and displays the trace output when it encounters the tag.

This is an example of a log file entry:

```
"Information"."web-4"."04/08/02"."23:21:30". ."[30 ms (1st trace)]
[C:\cfusion\wwwroot\generic.cfm @ line: 9] -
[thisPage = /generic.cfm] *
"Information"."web-0"."04/08/02"."23:58:58". ."[5187 ms (10)]
[C:\cfusion\wwwroot\generic.cfm @ line: 14] - [category]
[thisPage = /generic.cfm] [ABORTED] thisPage *
```

For a complex variable, ColdFusion lists the variable name and the number of elements in the object; it does not log the contents of the variable.

The following example traces a FORM variable that is evaluated by a `cfif` block:

**Example**

```coldfusion
<cftrace var="FORM.variable"
  text="doing equivalency check for FORM.variable"
  category="form_vars"
  inline="true">
  <cfif isDefined("FORM.variable") AND #FORMvariable# EQ 1>
    <h1>Congratulations, you're a winner!</h1>
  </cfif>
  <cfelse>
    <h1>Sorry, you lost!</h1>
  </cfelse>
</cftrace>
```

<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>Information</td>
<td>Corresponds to the <code>cflog type</code> attribute; displays an appropriate icon.</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 Information</td>
</tr>
<tr>
<td>var</td>
<td>Optional</td>
<td></td>
<td>The name of a simple or complex variable to display. Useful for displaying a temporary value, or a value that does not display on any CFM page.</td>
</tr>
</tbody>
</table>

342  Chapter 2: ColdFusion Tags
**cftransaction**

**Description**
Instructs the database management system to treat multiple database operations as a single transaction. Provides database commit and rollback processing.

**Category**
Database manipulation tags

**Syntax**
```html
<cftransaction
  action = "begin" or "commit" or "rollback"
  isolation = "read_uncommitted" or "read_committed" or "repeatable_read" />
</cftransaction>
```

**See also**
cfinsert, cfprocparam, cfprocresult, cfquery, cfqueryparam, cfstoredproc, cfupdate

**Attributes**

<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. commit: commits a pending transaction rollback: rolls back a pending transaction</td>
</tr>
<tr>
<td>isolation</td>
<td>Optional</td>
<td></td>
<td>ODBC lock type. read_uncommitted read_committed repeatable_read serializable</td>
</tr>
</tbody>
</table>

**Usage**
Within a transaction block, you can do the following:

- Commit a database transaction by nesting the `<cftransaction action = "commit"/>` tag within the block
- Roll back a transaction by nesting the `<cftransaction action = "rollback"/>` tag within the block

(In these examples, the slash is alternate syntax that is the equivalent of an end tag.)

Within a transaction block, you can write queries to more than one database, but you must commit or rollback a transaction to one database before writing a query to another. Using CFML error handling, you control whether each transaction is committed, based on the success or failure of the database query. To control how the database engine performs locking during the transaction, use the `isolation` attribute.

**Example**

```html
<p>CFTRANSACTION can be used to group multiple queries that use CFQUERY into one business event. Changes to data that is requested by the queries are not committed to the datasource until all actions within the transaction block have executed successfully. 
<p>This a view-only example.
```

---
<cftransaction>
  <cfquery name='makeNewCourse' datasource='Snippets'>
    INSERT INTO Courses
    (Number, Descript)
    VALUES
    ('#myNumber#', '#myDescription#')
  </cfquery>

  <cfquery name='insertNewCourseToList' datasource='Snippets'>
    INSERT INTO CourseList
    (CorNumber, CorDesc, Dept_ID,
    CorName, CorLevel, LastUpdate)
    VALUES
    ('#myNumber#', '#myDescription#', '#myDepartment#',
    '#myDescription#', '#myCorLevel#', #Now()#)
  </cfquery>
</cftransaction>

---
cftree

Description
Inserts a tree control in a form. Validates user selections. Used within a cftree tag block. You can use a ColdFusion query to supply data to the tree.

Category
Forms tags

Syntax
```<cftree name = "name"
  required = "Yes" or "No"
  delimiter = "delimiter"
  completePath = "Yes" or "No"
  appendKey = "Yes" or "No"
  highlightHref = "Yes" or "No"
  onValidate = "script_name"
  message = "text"
  onError = "text"
  lookAndFeel = "motif" or "windows" or "metal"
  font = "font"
  fontSize = "size"
  italic = "Yes" or "No"
  bold = "Yes" or "No"
  height = "integer"
  width = "integer"
  vSpace = "integer"
  hSpace = "integer"
  align = "alignment"
  border = "Yes" or "No"
  hScroll = "Yes" or "No"
  vScroll = "Yes" or "No"
  notSupported = "text">
</cftree>```

See also
cfapplet, cfform, cfgrid, cfgridcolumn, cfgridrow, cfgridupdate, cfinput, cfselect, cfslider, cftextinput, cftree, cftreeitem

History
ColdFusion MX: Changed behavior: ColdFusion renders a tree control regardless of whether there are any treeitems within it.

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 tree control.</td>
</tr>
</tbody>
</table>
| required  | Optional| No      | • Yes: user must select an item in tree control  
<p>|           |         |          | • No |
| delimiter | Optional| \ \    | Character to separate elements in form variable path. |</p>
<table>
<thead>
<tr>
<th>Attribute</th>
<th>Req/Opt</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
</table>
| completePath   | Optional| No; tree name is returned as root | • Yes: passes the root part of treename.path form variable when cfftree is submitted  
• No, or omitted: root level of form variable is not passed; path value starts with the first node  
For the preserveData attribute of cfform to work with the tree, you must set this attribute to Yes.  
If you specify a root name for a tree item with cfftreeitem queryasroot, that value is returned. If you do not specify a root name, ColdFusion returns the query name as the root. |
| appendKey      | Optional| Yes     | • Yes: when used with href, passes CFTREEITEMKEY variable with the value of the selected tree item in URL to the application page specified in the cfform action attribute  
• No: disables highlight |
| highlightHref  | Optional| Yes     | • Yes: highlights links that are associated with a cfftreeitem with a URL attribute value  
• No: disables highlight |
| onValidate     | Optional|         | JavaScript function to validate user input. The form object, input object, and input object value are passed to the specified routine, which should return True if validation succeeds; False, otherwise. |
| message        | Optional|         | Message to display if validation fails. |
| onError        | Optional|         | JavaScript function to execute if validation fails. |
| lookAndFeel   | Optional| windows | • motif: renders slider in Motif style  
• windows: renders slider in Windows style  
• metal: renders slider in Java Swing style  
If platform does not support style option, tag defaults to platform default style. |
| font           | Optional|         | Font name for data in tree control. |
| fontSize       | Optional|         | Font size for text in tree control, in points. |
| italic         | Optional| No      | • Yes: displays tree control text in italics  
• No |
| bold           | Optional| No      | • Yes: displays tree control text in bold  
• No |
| height         | Optional| 320     | Tree control height, in pixels. |
| width          | Optional| 200     | Tree control width, in pixels. |
| vSpace         | Optional|         | Vertical margin above and below tree control, in pixels. |
| hSpace         | Optional|         | Horizontal spacing to left and right of tree control, in pixels. |
Usage

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

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

If the following conditions are true, a user’s selection from query data that populates this tag’s options continues to display after the user submits the form:

- The `cfform preserveData` attribute is set to "Yes"
- The `cfform action` attribute posts to the same page as the form itself (this is the default), or the action page has a form that contains controls with the same names as corresponding controls on the user entry form

For more information, see the `cfform` tag entry.

Example

```coldfusion
<!--- This example shows the use of cftree in a cfform. The query takes a list of employees, and uses cftree and cfselect to display the results. cfgrid is used to show an alternative means of displaying the same data -->
<!--- set a default for the employeeNames variable --->
<cfparam name = "employeeNames" default = "">
<!--- if an employee name has been passed from the form, set employeeNames variable to this value --->
<cfif IsDefined("form.employeeNames")>
  <cfset employeeNames = form.employeeNames>
</cfif>
```
<!-- query the datasource to find the employee information -->
<cfquery name="GetEmployees" dataSource="cfsnippets">
  SELECT Emp_ID, FirstName, LastName, Email, Phone, Department
  FROM Employees where lastname
  <cfif #employeeNames# is not "">
    = '#employeeNames#'</cfif>
</cfquery>
</html>
<body>
<h3>cftree Example</h3>
<!-- Use cfinput when using other cfinput tools -->
<cfinput action="cftree.cfm">
<!-- Use cfselect to present the contents of the query by column -->
<h3>cfselect Presentation of Data</h3>
<h4>Click an employee's last name and "see information for this employee",
to see expanded information.</h4>
<cfselect name="EmployeeNames" message="Select an Employee Name"
  size="#getEmployees.recordcount#" query="GetEmployees"
  value="LastName" required="No">
  <option value="">Select All</option>
</cfselect>
<input type="Submit" name="" value="see information for this employee">
<!-- showing the use of cftree -->
<h3>cftree Presentation of Data</h3>
<h4>Click on the folders to "drill down" and reveal information.</h4>
<p>cftreeitem is used to create the "branches" of the tree. </p>
<cfloop query="GetEmployees">
  <cftreeitem value="#Emp_ID#" parent="SeeEmployees" expand="No">
    <cftreeitem value="#LastName#" display="Name"
      parent="#Emp_ID#" queryAsRoot="No" expand="No">
      <cftreeitem value="#LastName#, #FirstName#"
        parent="#LastName#" queryAsRoot="No" expand="No">
        <cftreeitem value="#Department#" display="Department"
          parent="#Emp_ID#" queryAsRoot="No" expand="No">
          <cftreeitem value="#Department#" parent="#Department#"
            queryAsRoot="No" expand="No">
            <cftreeitem value="#Phone#" display="Phone"
              parent="#Emp_ID#" queryAsRoot="No" expand="No">
              <cftreeitem value="#Phone#" parent="#Phone#"
                queryAsRoot="No" expand="No">
              <cftreeitem value="#Email#" display="Email"
                parent="#Emp_ID#" queryAsRoot="No" expand="No">
              </cfloop>
</cftreeitem>
</cfloop>
cftreeitem

Description

Populates a form tree control, created with the cftree tag, with elements. To display icons, you can use the img values that ColdFusion provides, or reference your own icons.

Category

Forms tags

Syntax

```<cftreeitem
  value = "text"
  display = "text"
  parent = "parent_name"
  img = "filename"
  imgopen = "filename"
  href = "URL"
  target = "URL_target"
  query = "queryname"
  queryAsRoot = "Yes" or "No"
  expand = "Yes" or "No">
</cftreeitem>```

See also

cfapplet, cfform, cfgrid, cfgridcolumn, cfgridrow, cfgridupdate, cfinput, cfselect, cfslider, cftextinput, cftree

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 passed when cfform is submitted. When populating a tree with data from a cfquery, specify columns in a delimited list. Example: value = &quot;dept_id,emp_id&quot;</td>
<td></td>
</tr>
<tr>
<td>display</td>
<td>Optional</td>
<td>value</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</td>
<td></td>
<td>Value for tree item parent.</td>
</tr>
</tbody>
</table>
| img       | Optional | folder | Image name, filename, or file URL for tree item icon. The following values are provided:  
  • cd  
  • computer  
  • document  
  • element  
  • folder  
  • floppy  
  • fixed  
  • remote  
  You can specify a custom image. To do so, include path and file extension; for example:  
  img = "/images/page1.gif"  
  To specify more than one image in a tree, or an image at the second or subsequent level, use commas to separate names, corresponding to level; for example:  
  img = "folder,document"  
  img = ",document" (example of second level) |
## Usage

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

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

## Example

```coldfusion
<!--- This example shows the use of cftreeitem in cfform. Query takes
employee list, and uses cftree and cfselect to display query results.
Shows an alternate means of displaying the data --->

<cfform action = "cfindex.cfm">
</cfform>
```

## Attribute

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Req/Opt</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>imgopen</td>
<td>Optional</td>
<td>Icon displayed with open tree item. You can specify icon filename with a relative path. You can use a ColdFusion image.</td>
<td></td>
</tr>
<tr>
<td>href</td>
<td>Optional</td>
<td>URL to associate with tree item or query column for a tree that is populated from a query. If href is a query column, its value is the value populated by query. If href is not recognized as a query column, it is assumed that its text is an HTML href. When populating a tree with data from a query, HREFs can be specified in delimited list; for example: href = &quot;http://dept_svr,http://emp_svr&quot;</td>
<td></td>
</tr>
<tr>
<td>target</td>
<td>Optional</td>
<td>Target attribute of href URL. When populating a tree with data from a query, specify target in delimited list: target = &quot;FRAME_BODY,_blank&quot;</td>
<td></td>
</tr>
<tr>
<td>query</td>
<td>Optional</td>
<td>Query name to generate data for the treeitem.</td>
<td></td>
</tr>
<tr>
<td>queryAsRoot</td>
<td>Optional</td>
<td>Defines query as the root level. This avoids having to create another parent cftreeitem.</td>
<td></td>
</tr>
<tr>
<td>expand</td>
<td>Optional</td>
<td>Yes</td>
<td>Yes: expands tree to show tree item children</td>
</tr>
</tbody>
</table>

---

350 Chapter 2: ColdFusion Tags
<!--- Use cfselect to present the contents of the query by column --->
<h3>cfselect Presentation of Data</h3>

<h4>Click an employee's last name and click "see information for this employee" to see expanded information.</h4>

<cfselect name = "EmployeeNames" message = "Select an Employee Name"
size = "#getEmployees.recordcount#" query = "GetEmployees"
value = "LastName" required = "No">
<option value = "> Select All </option>
<input type = "Submit" name = "" value = "see information for this employee">
</cfselect>

<!--- showing use of cftree for expanded presentation of data --->
<h3>cftree Presentation of Data</h3>
<h4>Click the folders to "drill down" and reveal information.</h4>
<p>cftreeitem is used to create the branches of the cftree.</p>

<p><cfselect name = "SeeEmployees" height = "150" width = "240"
font = "Arial Narrow" bold = "No" italic = "No" border = "Yes"
hscroll = "Yes" vscroll = "Yes" required = "No"
completePath = "No"
appendKey = "Yes" highlightHref = "Yes">
<cfloop query = "GetEmployees">
<cftreeitem value = "#Emp_ID#" parent = "SeeEmployees" expand = "No">
<cftreeitem value = "#LastName#" display = "Name"
parent = "#Emp_ID#" queryAsRoot = "No" expand = "No">
<cftreeitem value = "#LastName#, #FirstName#
parent = "#LastName#" expand = "No" queryAsRoot = "No">
<cftreeitem value = "#Department#
parent = "#Department#" expand = "No" queryAsRoot = "No">
<cftreeitem value = "#Phone#
parent = "#Phone#" expand = "No" queryAsRoot = "No">
<cftreeitem value = "#Email#
parent = "#Email#" expand = "No" queryAsRoot = "No">
</cfloop>
</cfselect>
</p>

<!--------For a more comprehensive, quicker view, you can use CFGRID -------->
<h3>cfgrid Presentation of Data</h3>
<cfgrid name="SampleGrid" width="600" query="GetEmployees" insert="No"
delete="No" sort="No" font="Verdana" bold="No" italic="No" appendKey="No"
highlightHref="No" griddataalign="LEFT" gridlines="Yes" rowheaders="No"
rowheaderalign="LEFT" rowheaderitalic="No" rowheaderbold="No"
colheaders="Yes"
colheaderalign="CENTER" colheaderitalic="No" colheaderbold="No"
colbgcolor="#E1F5FE" picturebar="No">
<cfgridcolumn name="LastName" header="Last Name" headeralign="LEFT"
dataalign="LEFT" bold="No" italic="No" select="Yes" display="Yes"
headerbold="No" headeritalic="No" headeralign="LEFT">
<cfgridcolumn name="FirstName" header="First Name" headeralign="LEFT"
dataalign="LEFT" fontsize="2"
bold="No" italic="No" select="No" display="Yes" headerbold="No"
headeritalic="No" headeralign="LEFT">
<cfgridcolumn name="Email" header="Email" headeralign="LEFT"
dataalign="LEFT" bold="No" italic="No" select="No" display="Yes" headerbold="No"
<cfgridcolumn name="Phone" header="Phone" headeralign="LEFT" dataalign="LEFT" bold="No" italic="Yes" select="No" display="Yes" headerbold="No" headeritalic="No">
  <cfgridcolumn name="Department" header="Department" headeralign="LEFT" dataalign="LEFT" bold="Yes" italic="No" select="No" display="Yes" headerbold="No" headeritalic="No">
    <cfgridcolumn name="Emp_ID" header="ID" headeralign="LEFT" dataalign="LEFT" width="40" bold="No" italic="No" select="No" display="Yes" headerbold="No" headeritalic="No">
  </cfgrid>
</cfgrid>
</cfform>
cftry

Description
Used with one or more cfcatch tags. Together, they catch and process exceptions in ColdFusion pages. Exceptions are events that disrupt the normal flow of instructions in a ColdFusion page, such as failed database operations, missing include files, and developer-specified events.

Category
Exception handling tags

Syntax
<cftry>
   Code that might throw an exception
   One or more cfcatch blocks
</cftry>

See also
cfcatch, cferror, cfrethrow, cfthrow. Chapter 14, “Handling Errors,” in Developing ColdFusion MX Applications

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 = "cfsnippets">
      SELECT *
      FROM EMPLOYEEAS
   </cfquery>

   <!--- <p>... other processing goes here --->
   <!--- specify the type of error for which we search --->
   <cfcatch type = "Database">
      <!--- the message to display --->
      <h3>You've Thrown a Database <b>Error</b></h3>
      <cfoutput>
         <!--- and the diagnostic message from the ColdFusion server --->
         <p>#cfcatch.message#</p>
         <p>Caught an exception, type = #CFCATCH.TYPE#</p>
         <p>The contents of the tag stack are:</p>
         <cfloop index = i from = 1 to = #ArrayLen(CFCATCH.TAGCONTEXT)#>
            <cfset sCurrent = #CFCATCH.TAGCONTEXT[i]#>
            <br> #i# #sCurrent["ID"]# (#sCurrent["LINE"]#, #sCurrent["COLUMN"]#)
         </cfloop>
      </cfoutput>
   </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>
      <p>The contents of the tag stack are:</p>
      <cfloop index = i from = 1 to = #ArrayLen(CFCATCH.TAGCONTEXT)#>
         <cfset sCurrent = #CFCATCH.TAGCONTEXT[i]#>
         <br> #i# #sCurrent["ID"]# (#sCurrent["LINE"]#, #sCurrent["COLUMN"]#)
      </cfloop>
   </cfoutput>
</cfcatch>

</cftry>
</cfoutput>
</cfcatch>
</cftry>
**cfupdate**

**Description**
Updates records in a data source from data in a ColdFusion form or form Scope.

**Category**
Database manipulation tags

**Syntax**

```<cfupdate
dataSource = "ds_name"
tableName = "table_name"
tableOwner = "name"
tableQualifier = "qualifier"
username = "username"
password = "password"
formFields = "field_names">```

**See also**

cfinser, cprovparam, cprocresult, cfquery, cfqueryparam, cfstoredproc, cftransaction

**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 database that contains 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 <code>username</code> value specified in ODBC setup.</td>
</tr>
<tr>
<td>password</td>
<td>Optional</td>
<td></td>
<td>Overrides <code>password</code> value specified in ODBC setup.</td>
</tr>
<tr>
<td>formFields</td>
<td>Optional</td>
<td>(all on form, except keys)</td>
<td>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 <code>formFields</code> field must include the database table primary key field, which must be present in the form. It can be hidden.</td>
</tr>
</tbody>
</table>
Example

<!-- This example shows the use of CFUPDATE to change records in a datasource. -->
<!-- if course_ID has been passed to this form, then perform the update on that record in the datasource -->

<cfif IsDefined("form.Course_ID")>
<!-- check that course_id is numeric -->
<cfif Not IsNumeric(form.Course_ID)>
<cfabort>
</cfif>
</cfif>

<!-- Now, do the update -->
<cfupdate datasource="cfsnippets"
tablename="Courses"
formfields="Course_ID,Number,Descript">
</cfif>

<!-- Perform a query to reflect any updated information if Course_ID is passed through a url, we are selecting a record to update ... select only that record with the WHERE clause. -->
<cfquery name="GetCourseInfo" DATASOURCE="cfsnippets">
SELECT Course_Number, Course_ID, Descript
FROM Courses
<cfif IsDefined("url.Course_ID")>
WHERE Course_ID = #Trim(url.Course_ID)#
</cfif>
ORDER by Course_Number
</cfquery>

<html>
<head>
<title>CFUPDATE Example</title>
<cfset css_path = "../../css">
<cfinclude template="../../resource/include/mm_browsersniff.cfm">
</head>
<body>

<H3>CFUPDATE Example</H3>
<!-- If we are updating a record, don't show the entire list. -->
<cfif IsDefined("url.Course_ID")>
<form method="post" action="index.cfm">
<H3>You can alter the contents of this record, and then click "Update" to use CFUPDATE and alter the database</H3>
<P>Course Number <INPUT TYPE="Text" name="Number" value="<cfoutput>#Trim(GetCourseInfo.Course_Number)#</cfoutput>">
<P>Course Description<br>
<textarea name="Descript" cols="40" rows="5">#Trim(GetCourseInfo.Descript)#</textarea>
<input type="Hidden" NAME="Course_ID" value="#Trim(GetCourseInfo.Course_ID)#">
<p><input type="Submit" value="Click to Update"></p>
</form>
</cfif>

<cfelse>
<!-- Show the entire record set in CFTABLE form -->
<cftable query="GetCourseInfo" htmltable colheaders>
<cfcol text="<a href='index.cfm?Course_ID=#Trim(Course_ID)#'>Edit Me</a>" width=10 header="Edit this Entry">
<cfcol text="#Trim(Course_Number)#" WIDTH=4 HEADER="Course Number">
<cfcol text="#Trim(Descript)#" WIDTH=100 HEADER="Course Description">
</cftable>
</cfelse>

</body>
cfwddx

Description
Serializes and deserializes CFML data structures to the XML-based WDDX format. The WDDX is an XML vocabulary for describing complex data structures in a standard, generic way. Implementing it lets you use the HTTP protocol to such information among application server platforms, application servers, and browsers.

This tag generates JavaScript statements to instantiate JavaScript objects equivalent to the contents of a WDDX packet or CFML data structure. Interoperates with Unicode.

Category
Extensibility tags

Syntax
```
<cfwddx
    action = "action"
    input = "inputdata"
    output = "resultvariablename"
    topLevelVariable = "toplevelvariablenameforjavascript"
    useTimeZoneInfo = "Yes" or "No"
    validate = "Yes" or "No"
>

See also
collection, cfdump, cfform, cfreport, cfsearch

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: serialize CFML to WDDX</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• wddx2cfml: deserialize WDDX to CFML</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• cfml2js: serialize CFML to JavaScript</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• wddx2js: deserialize WDDX to JavaScript</td>
</tr>
<tr>
<td>input</td>
<td>Required</td>
<td></td>
<td>A value to process</td>
</tr>
<tr>
<td>output</td>
<td>Required if action = &quot;wddx2cfml&quot;</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>
<tr>
<td>topLevelVariable</td>
<td>Required if action = &quot;wddx2js&quot; or &quot;cfml2js&quot;</td>
<td></td>
<td>Name of top-level JavaScript object created by deserialization. The object is an instance of the WddxRecordset object.</td>
</tr>
</tbody>
</table>

358 Chapter 2: ColdFusion Tags
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>From</th>
<th>To</th>
</tr>
</thead>
<tbody>
<tr>
<td>CFML</td>
<td>WDDX</td>
</tr>
<tr>
<td>CFML</td>
<td>JavaScript</td>
</tr>
<tr>
<td>WDDX</td>
<td>CFML</td>
</tr>
<tr>
<td>WDDX</td>
<td>JavaScript</td>
</tr>
</tbody>
</table>

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 31, “Using XML and WDDX,” in Developing ColdFusion MX Applications.

**Example**

```
<!--- This shows basic use of the cfwddx tag. --->
<html>
<body>
<!--- Create a simple query --->
<cfquery name = "q" dataSource = "cfsnippets">
    select Message_Id, Thread_id, Username from messages
</cfquery>

The recordset data is:...
<cfoutput query = q>
    #Message_ID# #Thread_ID# #Username#
</cfoutput>

<!--- 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...<p>
<cfwddx action = "wddx2cfml" input = #wddxText# output = "qnew">
The recordset data is:....<p>
<cfoutput query = qnew>
  #Message_ID# #Thread_ID# #Username#<br>
</cfoutput><p>
cfxmxml

Description
Creates a ColdFusion XML document object that contains the markup in the tag body. This tag
can include XML and CFML tags. ColdFusion processes the CFML code in the tag body, then
assigns the resulting text to an XML document object variable.

Category
Extensibility tags

Syntax
<CFXML
  variable="xmlVarName"
  caseSensitive="yes" or "no">

See also
IsXmlDoc, IsXmlElem, IsXmlRoot, XmlChildPos, XmlNew, XmlParse, XmlSearch, XmlTransform

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>variable</td>
<td></td>
<td></td>
<td>Name of an xml variable</td>
</tr>
<tr>
<td>caseSensitive</td>
<td>Optional</td>
<td>no</td>
<td>• yes: maintains the case of document elements and attributes • no</td>
</tr>
</tbody>
</table>

Usage
An XML document object is represented in ColdFusion as a structure.
The following example creates a document object whose root element is MyDoc. The object
includes text that displays the value of the ColdFusion variable testVar. The code creates four
nested child elements, which are generated by an indexed cfloop tag. The cfdump tag displays
the XML document object.

Note: Do not include an <?xml ?> processing directive in the cfxmxml tag body. This directive is not
required for processing, and causes an error. To process XML text that includes this directive, use the
XmlParse function.

Example
<cfset testVar = True>
<cfxml variable="MyDoc">
  <MyDoc>
    <cfif testVar IS True>
      <cfoutput>The value of testVar is True.</cfoutput>
    <cfelse>
      <cfoutput>The value of testVar is False.</cfoutput>
    </cfif>
    <cfloop index = "LoopCount" from = "1" to = "4">
      <childNode>
        This is Child node<cfoutput>#LoopCount#.</cfoutput>
      </childNode>
    </cfloop>
  </MyDoc>
</cfxml>
</cfloop>
</MyDoc>
</cfxm1>
<cfdump var=#MyDoc#>
CHAPTER 3
ColdFusion Functions

This chapter lists and categorizes ColdFusion Markup Language (CFML) functions.

Contents
Function list.................................................. 364
Functions by category................................. 367
Function changes since ColdFusion 5.............. 372
Function descriptions................................. 374
**Function list**

ColdFusion Markup Language (CFML) includes a set of functions that you use in ColdFusion pages to perform logical and arithmetic operations and manipulate data.

The following table lists CFML functions:

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abs</td>
<td>GetHttpTimeString</td>
<td>Min</td>
<td>GetK2ServerDocCount</td>
</tr>
<tr>
<td>ACos</td>
<td>GetK2ServerDocCountLimit</td>
<td>Month</td>
<td>GetLocale</td>
</tr>
<tr>
<td>ArrayAppend</td>
<td>GetK2ServerDocCountLimit</td>
<td>MonthAsString</td>
<td>GetMetaData</td>
</tr>
<tr>
<td>ArrayAvg</td>
<td>GetLocale</td>
<td>Month</td>
<td>GetMetaData</td>
</tr>
<tr>
<td>ArrayClear</td>
<td>GetMetricData</td>
<td>Minute</td>
<td>GetMetricData</td>
</tr>
<tr>
<td>ArrayDeleteAt</td>
<td>GetPageContext</td>
<td>NumberFormat</td>
<td>GetProfileSections</td>
</tr>
<tr>
<td>ArrayInsertAt</td>
<td>GetProfileString</td>
<td>ParseDateTime</td>
<td>GetProfileSections</td>
</tr>
<tr>
<td>ArrayIsEmpty</td>
<td>GetProfileSections</td>
<td>ParameterExists</td>
<td>GetProfileSections</td>
</tr>
<tr>
<td>ArrayLen</td>
<td>GetProfileString</td>
<td>Pi</td>
<td>GetProfileSections</td>
</tr>
<tr>
<td>ArrayMax</td>
<td>GetTempDirectory</td>
<td>Pi</td>
<td>GetProfileSections</td>
</tr>
<tr>
<td>ArrayMin</td>
<td>GetTempFile</td>
<td>Pi</td>
<td>GetProfileSections</td>
</tr>
<tr>
<td>ArrayNew</td>
<td>GetTemplatePath</td>
<td>Quarter</td>
<td>GetTemplatePath</td>
</tr>
<tr>
<td>ArrayPrepend</td>
<td>GetTickCount</td>
<td>QueryAddColumn</td>
<td>GetTimeZoneInfo</td>
</tr>
<tr>
<td>ArrayResize</td>
<td>GetTimeZoneInfo</td>
<td>QueryAddRow</td>
<td>Hash</td>
</tr>
<tr>
<td>ArraySet</td>
<td>GetToken</td>
<td>QueryNew</td>
<td>Hash</td>
</tr>
<tr>
<td>ArraySort</td>
<td>Hash</td>
<td>QuotedValueList</td>
<td>HTMLCodeFormat</td>
</tr>
<tr>
<td>ArraySum</td>
<td>HTMLCodeFormat</td>
<td>Rand</td>
<td>IIf</td>
</tr>
<tr>
<td>ArraySwap</td>
<td>Hash</td>
<td>Randomize</td>
<td>IncrementValue</td>
</tr>
<tr>
<td>ArrayToList</td>
<td>Hash</td>
<td>REFind</td>
<td>InputBaseN</td>
</tr>
<tr>
<td>Asc</td>
<td>IIf</td>
<td>REFindNoCase</td>
<td>Int</td>
</tr>
<tr>
<td>ASin</td>
<td>IncrementValue</td>
<td>ReleaseComObject</td>
<td>IsArray</td>
</tr>
<tr>
<td>Atn</td>
<td>InputBaseN</td>
<td>ReleaseComObject</td>
<td>RepeatString</td>
</tr>
<tr>
<td>BitAnd</td>
<td>Int</td>
<td>Replace</td>
<td>IsBinary</td>
</tr>
<tr>
<td>BitMaskClear</td>
<td>Int</td>
<td>ReplaceList</td>
<td>IsBoolean</td>
</tr>
<tr>
<td>BitMaskRead</td>
<td>IsArray</td>
<td>ReplaceNoCase</td>
<td>IsCustomFunction</td>
</tr>
<tr>
<td>BitMaskSet</td>
<td>IsBinary</td>
<td>ReplaceNoCase</td>
<td>IsDate</td>
</tr>
<tr>
<td>BitNot</td>
<td>IsBoolean</td>
<td>ReplaceNoCase</td>
<td>IsDebugMode</td>
</tr>
<tr>
<td>BitOr</td>
<td>IsCustomFunction</td>
<td>ReplaceNoCase</td>
<td>ReplaceNoCase</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
<td></td>
<td></td>
</tr>
<tr>
<td>---------------------------</td>
<td>--------------------------------------------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BitXor</td>
<td>IsDefined</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ceiling</td>
<td>IsK2ServerABroker</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Chr</td>
<td>IsK2ServerDocCountExceeded</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CJustify</td>
<td>IsK2ServerOnline</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Compare</td>
<td>IsLeapYear</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CompareNoCase</td>
<td>IsNumeric</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cos</td>
<td>IsNumericDate</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CreateDate</td>
<td>IsObject</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CreateDateTime</td>
<td>IsQuery</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CreateObject</td>
<td>IsSimpleValue</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CreateODBCDate</td>
<td>IsStruct</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CreateODBCDateTime</td>
<td>IsUserInRole</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CreateOOBCTime</td>
<td>IsWDOX</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CreateTime</td>
<td>IsXmlDoc</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CreateTimeSpan</td>
<td>IsXmlElement</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CreateUUID</td>
<td>IsXmlRoot</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DateAdd</td>
<td>JavaCast</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DateCompare</td>
<td>JSStringFormat</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DateConvert</td>
<td>LCase</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DateDiff</td>
<td>Left</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DateFormat</td>
<td>Len</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DatePart</td>
<td>ListAppend</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Day</td>
<td>ListChangeDelims</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DayOfWeek</td>
<td>ListContains</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DayOfWeekAsString</td>
<td>ListContainsNoCase</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DayOfYear</td>
<td>ListDeleteAt</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DaysInMonth</td>
<td>ListFind</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DaysInYear</td>
<td>ListFindNoCase</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DE</td>
<td>ListFirst</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DecimalFormat</td>
<td>ListGetAt</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DecrementValue</td>
<td>ListInsertAt</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Decrypt</td>
<td>ListLast</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DeleteClientVariable</td>
<td>ListLen</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DirectoryExists</td>
<td>ListPrepend</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IsDefined</td>
<td>Reverse</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IsK2ServerABroker</td>
<td>Right</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IsK2ServerDocCountExceeded</td>
<td>RJustify</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IsK2ServerOnline</td>
<td>Round</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IsLeapYear</td>
<td>RTrim</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IsNumeric</td>
<td>Second</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IsNumericDate</td>
<td>SetEncoding</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IsObject</td>
<td>SetLocale</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IsQuery</td>
<td>SetProfileString</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IsSimpleValue</td>
<td>SetVariable</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IsStruct</td>
<td>Sgn</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IsUserInRole</td>
<td>Sin</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IsWDOX</td>
<td>SpanExcluding</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IsXmlDoc</td>
<td>SpanIncluding</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IsXmlElement</td>
<td>Sqr</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IsXmlRoot</td>
<td>StripCR</td>
<td></td>
<td></td>
</tr>
<tr>
<td>JavaCast</td>
<td>StructAppend</td>
<td></td>
<td></td>
</tr>
<tr>
<td>JSStringFormat</td>
<td>StructClear</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LCase</td>
<td>StructCopy</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Left</td>
<td>StructCount</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Len</td>
<td>StructDelete</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ListAppend</td>
<td>StructFind</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ListChangeDelims</td>
<td>StructFindKey</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ListContains</td>
<td>StructFindValue</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ListContainsNoCase</td>
<td>StructGet</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ListDeleteAt</td>
<td>StructInsert</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ListFind</td>
<td>StructIsEmpty</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ListFindNoCase</td>
<td>StructKeyArray</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ListFirst</td>
<td>StructKeyExists</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ListGetAt</td>
<td>StructKeyList</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ListInsertAt</td>
<td>StructNew</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ListLast</td>
<td>StructSort</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ListLen</td>
<td>StructUpdate</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ListPrepend</td>
<td>Tan</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ColdFusion Function</td>
<td>List Function</td>
<td>Common Function</td>
<td></td>
</tr>
<tr>
<td>----------------------</td>
<td>---------------------</td>
<td>------------------------</td>
<td></td>
</tr>
<tr>
<td>DollarFormat</td>
<td>ListQualify</td>
<td>TimeFormat</td>
<td></td>
</tr>
<tr>
<td>Duplicate</td>
<td>ListRest</td>
<td>ToBase64</td>
<td></td>
</tr>
<tr>
<td>Encrypt</td>
<td>ListSetAt</td>
<td>ToBinary</td>
<td></td>
</tr>
<tr>
<td>Evaluate</td>
<td>ListSort</td>
<td>ToString</td>
<td></td>
</tr>
<tr>
<td>Exp</td>
<td>ListToArray</td>
<td>Trim</td>
<td></td>
</tr>
<tr>
<td>ExpandPath</td>
<td>ListValueCount</td>
<td>UCase</td>
<td></td>
</tr>
<tr>
<td>FileExists</td>
<td>ListValueCountNoCase</td>
<td>URLDecode</td>
<td></td>
</tr>
<tr>
<td>Find</td>
<td>LJustify</td>
<td>URLEncodedFormat</td>
<td></td>
</tr>
<tr>
<td>FindNoCase</td>
<td>Log</td>
<td>URLSessionFormat</td>
<td></td>
</tr>
<tr>
<td>FindOneOf</td>
<td>Log10</td>
<td>Val</td>
<td></td>
</tr>
<tr>
<td>FirstDayOfMonth</td>
<td>LSCurrencyFormat</td>
<td>ValueList</td>
<td></td>
</tr>
<tr>
<td>Fix</td>
<td>LSDateFormat</td>
<td>Week</td>
<td></td>
</tr>
<tr>
<td>FormatBaseN</td>
<td>LSIsCurrency</td>
<td>WriteOutput</td>
<td></td>
</tr>
<tr>
<td>GetAuthUser</td>
<td>LSIsDate</td>
<td>XMLChildPos</td>
<td></td>
</tr>
<tr>
<td>GetBaseTagData</td>
<td>LSIsNumeric</td>
<td>XMLElemNew</td>
<td></td>
</tr>
<tr>
<td>GetBaseTagList</td>
<td>LSIsNumeric</td>
<td>XMLElement</td>
<td></td>
</tr>
<tr>
<td>GetBaseTemplatePath</td>
<td>LSNumberFormat</td>
<td>XMlFormat</td>
<td></td>
</tr>
<tr>
<td>GetClientVariablesList</td>
<td>LSParseCurrency</td>
<td>XMLElement</td>
<td></td>
</tr>
<tr>
<td>GetCurrentTemplatePath</td>
<td>LSParseDateTime</td>
<td>XMLElement</td>
<td></td>
</tr>
<tr>
<td>GetDirectoryFromPath</td>
<td>LSParseEuroCurrency</td>
<td>XMLElement</td>
<td></td>
</tr>
<tr>
<td>GetEncoding</td>
<td>LSParseNumber</td>
<td>XMLTransform</td>
<td></td>
</tr>
<tr>
<td>GetException</td>
<td>LSTimeFormat</td>
<td>Year</td>
<td></td>
</tr>
<tr>
<td>GetFileFromPath</td>
<td>LTrim</td>
<td>YesNoFormat</td>
<td></td>
</tr>
<tr>
<td>GetFunctionList</td>
<td>Max</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GetHttpRequestData</td>
<td>Mid</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Functions by category

The following tables list functions by their category or purpose.

<table>
<thead>
<tr>
<th>Category</th>
<th>Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Array functions</td>
<td>ArrayAppend, ArrayIsEmpty, ArrayPrepend, ArraySwap</td>
</tr>
<tr>
<td>Authentication functions</td>
<td>GetAuthUser, IsUserInRole</td>
</tr>
<tr>
<td>Conversion functions</td>
<td>ArrayAvg, ArrayLen, ArrayClear, ArrayDeleteAt, ArrayInsertAt</td>
</tr>
<tr>
<td>Date and time functions</td>
<td>ArrayToList, Hash, URLEncodedFormat, XmlTransform</td>
</tr>
<tr>
<td>Decision functions</td>
<td>ArrayMax, ArrayMin, ArraySet, ArraySort, ListToArray</td>
</tr>
<tr>
<td>Display and formatting functions</td>
<td>ArrayNew, ArraySum</td>
</tr>
<tr>
<td>Dynamic evaluation functions</td>
<td></td>
</tr>
<tr>
<td>Extensibility functions</td>
<td></td>
</tr>
<tr>
<td>Full-text search functions</td>
<td></td>
</tr>
<tr>
<td>International functions</td>
<td></td>
</tr>
<tr>
<td>List functions</td>
<td></td>
</tr>
<tr>
<td>Mathematical functions</td>
<td></td>
</tr>
<tr>
<td>Other functions</td>
<td></td>
</tr>
<tr>
<td>Query functions</td>
<td></td>
</tr>
<tr>
<td>String functions</td>
<td></td>
</tr>
<tr>
<td>Structure functions</td>
<td></td>
</tr>
<tr>
<td>System functions</td>
<td></td>
</tr>
<tr>
<td>XML functions</td>
<td></td>
</tr>
</tbody>
</table>

Array functions

<table>
<thead>
<tr>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>ArrayAppend</td>
</tr>
<tr>
<td>ArrayIsEmpty</td>
</tr>
<tr>
<td>ArrayPrepend</td>
</tr>
<tr>
<td>ArraySwap</td>
</tr>
<tr>
<td>ArrayAvg</td>
</tr>
<tr>
<td>ArrayLen</td>
</tr>
<tr>
<td>ArrayResize</td>
</tr>
<tr>
<td>ArrayToList</td>
</tr>
<tr>
<td>ArrayClear</td>
</tr>
<tr>
<td>ArrayMax</td>
</tr>
<tr>
<td>ArraySet</td>
</tr>
<tr>
<td>IsArray</td>
</tr>
<tr>
<td>ArrayDeleteAt</td>
</tr>
<tr>
<td>ArrayMin</td>
</tr>
<tr>
<td>ArraySort</td>
</tr>
<tr>
<td>ListToArray</td>
</tr>
<tr>
<td>ArrayInsertAt</td>
</tr>
<tr>
<td>ArrayNew</td>
</tr>
<tr>
<td>ArraySum</td>
</tr>
<tr>
<td>GetAuthUser</td>
</tr>
<tr>
<td>IsUserInRole</td>
</tr>
</tbody>
</table>

Authentication functions

<table>
<thead>
<tr>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>GetAuthUser</td>
</tr>
<tr>
<td>IsUserInRole</td>
</tr>
</tbody>
</table>

Conversion functions

<table>
<thead>
<tr>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>ArrayToList</td>
</tr>
<tr>
<td>Hash</td>
</tr>
<tr>
<td>URLEncodedFormat</td>
</tr>
<tr>
<td>XmlTransform</td>
</tr>
<tr>
<td>ToBase64</td>
</tr>
<tr>
<td>LCase</td>
</tr>
<tr>
<td>Val</td>
</tr>
</tbody>
</table>
Date and time functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Function</th>
<th>Function</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>CreateDate</td>
<td>DateFormat</td>
<td>GetTimeZoneInfo</td>
<td>MonthAsString</td>
</tr>
<tr>
<td>CreateDateTime</td>
<td>DatePart</td>
<td>Hour</td>
<td>Now</td>
</tr>
<tr>
<td>CreateODBCDate</td>
<td>Day</td>
<td>IsDate</td>
<td>ParseDateTime</td>
</tr>
<tr>
<td>CreateODBCDateTime</td>
<td>DayOfWeek</td>
<td>IsLeapYear</td>
<td>Quarter</td>
</tr>
<tr>
<td>CreateODBCTime</td>
<td>DayOfWeekAsString</td>
<td>IsNumericDate</td>
<td>Second</td>
</tr>
<tr>
<td>CreateTime</td>
<td>DayOfYear</td>
<td>LSDateFormat</td>
<td>TimeFormat</td>
</tr>
<tr>
<td>CreateTimeSpan</td>
<td>DaysInMonth</td>
<td>LSIsDate</td>
<td>Week</td>
</tr>
<tr>
<td>DateAdd</td>
<td>DaysInYear</td>
<td>LSIsDate</td>
<td>Year</td>
</tr>
<tr>
<td>DateCompare</td>
<td>FirstDayOfMonth</td>
<td>LSTimeFormat</td>
<td></td>
</tr>
<tr>
<td>DateConvert</td>
<td>GetHttpTimeString</td>
<td>Minute</td>
<td></td>
</tr>
<tr>
<td>DateDiff</td>
<td>GetTickCount</td>
<td>Month</td>
<td></td>
</tr>
</tbody>
</table>

Decision functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Function</th>
<th>Function</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>DirectoryExists</td>
<td>IsDebugMode</td>
<td>IsObject</td>
<td>IsXmlRoot</td>
</tr>
<tr>
<td>FileExists</td>
<td>IsDefined</td>
<td>IsQuery</td>
<td>LSIsCurrency</td>
</tr>
<tr>
<td>IIf</td>
<td>IsK2ServerABroker</td>
<td>IsSimpleValue</td>
<td>LSIsDate</td>
</tr>
<tr>
<td>IsArray</td>
<td>IsK2ServerDocCountExceeded</td>
<td>IsStruct</td>
<td>LSIsNumeric</td>
</tr>
<tr>
<td>IsBinary</td>
<td>IsK2ServerOnline</td>
<td>IsUserInRole</td>
<td>StructIsEmpty</td>
</tr>
<tr>
<td>IsBoolean</td>
<td>IsLeapYear</td>
<td>IsWDDX</td>
<td>StructKeyExists</td>
</tr>
<tr>
<td>IsCustomFunction</td>
<td>IsNumeric</td>
<td>XmlNode</td>
<td>YesNoFormat</td>
</tr>
<tr>
<td>IsDate</td>
<td>IsNumericDate</td>
<td>IsXmlNode</td>
<td></td>
</tr>
<tr>
<td>IsDate</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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>HTMLEditFormat</td>
<td>LSNumberFormat</td>
<td>ParagraphFormat</td>
</tr>
<tr>
<td>DateFormat</td>
<td>LJustify</td>
<td>LSParseCurrency</td>
<td>RJustify</td>
</tr>
<tr>
<td>DecimalFormat</td>
<td>LSCurrencyFormat</td>
<td>LSParseDateTime</td>
<td>StripCR</td>
</tr>
<tr>
<td>DollarFormat</td>
<td>LSDateFormat</td>
<td>LSParseEuroCurrency</td>
<td>TimeFormat</td>
</tr>
<tr>
<td>FormatBaseN</td>
<td>LSEuroCurrencyFormat</td>
<td>LSParseNumber</td>
<td>YesNoFormat</td>
</tr>
<tr>
<td>GetLocale</td>
<td>LSIsCurrency</td>
<td>LSTimeFormat</td>
<td>StripCR</td>
</tr>
<tr>
<td>HTMLCodeFormat</td>
<td>LSIsDate</td>
<td></td>
<td>NumberFormat</td>
</tr>
</tbody>
</table>
Dynamic evaluation functions

| DE                | Evaluate | IIf        | SetVariable |

Extensibility functions

<table>
<thead>
<tr>
<th>CreateObject</th>
<th>XmlElemNew</th>
<th>XmlParse</th>
</tr>
</thead>
<tbody>
<tr>
<td>ReleaseComObject</td>
<td>XmlFormat</td>
<td>XmlSearch</td>
</tr>
<tr>
<td>XmlChildPos</td>
<td>XmlNew</td>
<td>XmlTransform</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.

| GetK2ServerDocCount | IsK2ServerABroker | IsK2ServerOnline |
| GetK2ServerDocCountLimit | IsK2ServerDocCountExceeded |

International functions

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

List functions

| ArraySort          | FindNoCase       | ListContainsNoCase | ListQualify     |
| ArrayToList        | FindOneOf        | ListDeleteAt       | ListRest        |
| Asc                | FormatBaseN      | ListFind           | ListSetAt       |
| Chr                | GetClientVariablesList | ListFindNoCase     | ListSort        |
| CJustify           | LCase            | ListFirst          | ListToArray     |
| Compare            | Left             | ListGetAt          | ListValueCount  |
| CompareNoCase      | Len              | ListInsertAt       | ListValueCountNoCase |
| Decrypt            | ListAppend       | ListLast           | ReplaceList     |
| Encrypt            | ListChangeDelims | ListLen            |                |
| Find               | ListContains     | ListPrepend        |                |
## 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>Atn</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>QuotedValueList</th>
<th>URLENCodedFormat</th>
</tr>
</thead>
<tbody>
<tr>
<td>Decrypt</td>
<td>GetBaseTemplatePath</td>
<td>StripCR</td>
<td>URLEncodedFormat</td>
</tr>
<tr>
<td>DeleteClientVariable</td>
<td>GetClientVariablesList</td>
<td>ToBase64</td>
<td>ValueList</td>
</tr>
<tr>
<td>Duplicate</td>
<td>GetTickCount</td>
<td>ToBinary</td>
<td>WriteOutput</td>
</tr>
<tr>
<td>Encrypt</td>
<td>Hash</td>
<td>ToBinary</td>
<td>WriteOutput</td>
</tr>
<tr>
<td>GetBaseTagData</td>
<td>PreserveSingleQuotes</td>
<td>URLDecode</td>
<td></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>

## 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>JSStringFormat</th>
<th>MonthAsString</th>
<th>SpanExcluding</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chr</td>
<td>LCase</td>
<td>ParagraphFormat</td>
<td>SpanIncluding</td>
</tr>
<tr>
<td>CJustify</td>
<td>Left</td>
<td>ParseDateTime</td>
<td>StripCR</td>
</tr>
</tbody>
</table>
Functions by category

See also “Conversion functions” on page 367.

**Structure functions**

- Duplicate
- StructCount
- StructGet
- StructKeyList
- IsStruct
- StructDelete
- StructInsert
- StructNew
- StructAppend
- StructFind
- StructIsEmpty
- StructSort
- StructClear
- StructFindKey
- StructKeyArray
- StructUpdate
- StructCopy
- StructFindValue
- StructKeyExists

**System functions**

- DirectoryExists
- GetEncoding
- GetMetaData
- GetTemplatePath
- ExpandPath
- GetException
- GetMetricData
- GetPageContext
- FileExists
- GetFileFromPath
- GetProfileSections
- SetEncoding
- GetBaseTemplatePath
- GetFunctionList
- GetProfileString
- SetLocale
- GetCurrentTemplatePath
- GetHttpRequestData
- GetTempDirectory
- SetProfileString
- GetDirectoryFromPath
- GetLocale
- GetTempFile

**XML functions**

- IsXmlDoc
- IsWDDX
- XmlFormat
- XmlSearch
- IsXmlElem
- XmlChildPos
- XmlNew
- XmlTransform
- IsXmlRoot
- XmlElemNew
- XmlParse
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.

Array functions ................................. 367
Deprecated functions, parameters, and values ................................. 373
Obsolete functions, parameters, and values .................................. 373

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>CreateObject</td>
<td>All</td>
<td>ColdFusion MX</td>
</tr>
<tr>
<td>DateAdd</td>
<td>key of datepart</td>
<td>ColdFusion MX 6.1</td>
</tr>
<tr>
<td>DatePart</td>
<td>key of datepart</td>
<td>ColdFusion MX 6.1</td>
</tr>
<tr>
<td>GetAuthUser</td>
<td>All</td>
<td>ColdFusion MX</td>
</tr>
<tr>
<td>GetEncoding</td>
<td>All</td>
<td>ColdFusion MX</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>IsObject</td>
<td>All</td>
<td>ColdFusion MX</td>
</tr>
<tr>
<td>IsUserInRole</td>
<td>All</td>
<td>ColdFusion MX</td>
</tr>
<tr>
<td>IsXmlDoc</td>
<td>All</td>
<td>ColdFusion MX</td>
</tr>
<tr>
<td>IsXmlElem</td>
<td>All</td>
<td>ColdFusion MX</td>
</tr>
<tr>
<td>IsXmlRoot</td>
<td>All</td>
<td>ColdFusion MX</td>
</tr>
<tr>
<td>LSTimeFormat</td>
<td>key of mask</td>
<td>ColdFusion MX 6.1</td>
</tr>
<tr>
<td>ReleaseComObject</td>
<td>All</td>
<td>ColdFusion MX 6.1</td>
</tr>
<tr>
<td>SetEncoding</td>
<td>All</td>
<td>ColdFusion MX</td>
</tr>
<tr>
<td>TimeFormat</td>
<td>key of mask</td>
<td>ColdFusion MX 6.1</td>
</tr>
<tr>
<td>URLEncode</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>charset parameter</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>XmlNew</td>
<td>All</td>
<td>ColdFusion MX</td>
</tr>
<tr>
<td>XmlParse</td>
<td>All</td>
<td>ColdFusion MX</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>XmlSearch</td>
<td>All</td>
<td>ColdFusion MX</td>
</tr>
<tr>
<td>XmlTransform</td>
<td>All</td>
<td>ColdFusion MX</td>
</tr>
<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 Limit</td>
<td>All</td>
<td>ColdFusion MX.6.1</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 Exceeded</td>
<td>All</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 might not work, and might cause an error, 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., -3.2, 6 is</p>
<pre><cfoutput>
#Abs(1)#,#Abs(3)#,#Abs(-4)#,#Abs(-3.2)#,#Abs(6)#
</cfoutput></pre>
<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
<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>
    <cfoutput>ACos(#FORM.CosNum#) = #ACos(FORM.cosNum) * 180/PI()#</cfoutput>
  <cfelse>
    <!--- if it is empty, output an error message --->
    <h4>Enter a number between -1 and 1</h4>
  </cfif>
</cfif>
</cfif>
<form method="post" action = "acos.cfm">
  Enter a number to get its arccosine in Radians and Degrees.
  <input type = "Text" name = "cosNum" size = "25">
  <input type = "Submit" name = "" > <input type = "RESET">
</Form>
ArrayAppend

Description
Appends an array element to an array.

Returns
True, on successful completion.

Category
Array functions

Function syntax
ArrayAppend(array, value)

See also
ArrayPrepend

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

```cfm
<h3>ArrayAppend Example</h3>
<cfquery name = "GetEmployeeNames" datasource = "cfsnippets">
SELECT FirstName, LastName FROM Employees
</cfquery>
<!--- create an array --->
<cfset myArray = ArrayNew(1)>
<!--- set element one to show where we are --->
<cfset myArray[1] = "Test Value">
<!--- loop through the query; append these names successively to the last element --->
<cfloop query = "GetEmployeeNames">
<cfoutput>#ArrayAppend(myArray, "#FirstName# #LastName#")#
</cfoutput>. Array was appended<br>
</cfloop>
<!--- show the resulting array as a list --->
<cfset myList = ArrayToList(myArray, ",")>
<!--- output the array as a list --->
<cfoutput>
<p>The contents of the array are as follows:
<p>#myList#
</cfoutput>
```
ArrayAvg

Description
Calculates the average of the values in an array.

Returns
Number. If the array parameter value is an empty array, returns zero.

Category
Array functions, Mathematical functions

Function syntax
ArrayAvg(array)

See also
ArraySum

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>
<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. To enter more than two numbers press the more button. </p>
<form action = "arrayavg.cfm">
  <input type = "submit" name = "submit" value = "more">
</form>
</body>
</html>
<table cellspacing="2" cellpadding="2" border="0">
   <cfloop index="LoopItem" from="1" to="#FormElem#">
      <tr>
         <th align="left">Number #LoopItem#</th>
         <td><input type="text" name="number#LoopItem#"></td>
      </tr>
   </cfloop>
</table>
<input type="submit" name="submit" value="get the average">

<cfif IsDefined("FORM.submit")>
   <cfset myNumberArray = ArrayNew(1)>
   <cfset Count = 1>
   <cfloop index="ListItem" list="#Form.Fieldnames#">
      <cfif Left(ListItem,3) is "Num">
         <cfset myNumberArray[Count] = Val(Evaluate("number#Count#")))>
         <cfset count = IncrementValue(Count)>
      </cfif>
   </cfloop>
   <cfif Form.Submit is "get the average">
      <!--- use ArrayAvg to get the average of the two numbers --->
      <p>The average of the numbers that you entered is<br>
         #ArrayAvg(myNumberArray)#.</p>
   </cfif>
   <cfelse>
      <cfoutput>Try again. You must enter at least two numeric values.</cfoutput>
   </cfif>
</cfif>
ArrayClear

**Description**

Deletes the data in an array.

**Returns**

True, on successful completion.

**Category**

Array functions

**Function syntax**

ArrayClear(array)

**See also**

ArrayDeleteAt

**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**

```cfml
<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

**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**

```
<h3>ArrayDeleteAt Example</h3>
<pre>
<!--- create an array --->
<cfset DaysArray = ArrayNew(1)>
<!--- populate an element or two --->
<cfset DaysArray[1] = "Monday">
<cfset DaysArray[2] = "Tuesday">
<cfset DaysArray[3] = "Wednesday">
<!--- delete the second element --->
<cfset ArrayDeleteAt(DaysArray,2)>  
<p>Is the second element gone?: #ArrayDeleteAt(DaysArray,2)#</p>
<!--- the formerly third element, "Wednesday" is second element --->
<p>The second element is now: #DaysArray[2]#</p>
</pre>
```
ArrayInsertAt

Description
Inserts a value into an array. Array elements whose indexes are greater than the new position are incremented by one. The array length increases by one.

Returns
True, on successful completion.

Category
Array functions

Function syntax
ArrayInsertAt(array, position, value)

See also
ArrayDeleteAt

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>

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 --->
<cfoutput>ArrayInsertAt(DaysArray,3,"Wednesday")</cfoutput>
<p>Now output the array as a list:
<cfoutput>ArrayToList(DaysArray)</cfoutput>
<!--- The array now has four elements. Element 3, "Thursday", has become element four --->```
ArrayIsEmpty

**Description**
Determine if an array is empty of data elements.

**Returns**
True if the array is empty; otherwise, False.

**Category**
Array functions

**Function syntax**
ArrayIsEmpty(array)

**See also**
ArrayLen

**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**

```xml
<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
ArrayIsEmpty

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 = "cfsnippets">
SELECT FirstName, LastName FROM Employees
</cfquery>
<!--- create an array --->
<cfset myArray = ArrayNew(1)>
<!--- set element one to show where we are --->
<cfset myArray[1] = "Test Value">
<!--- loop through the query and append these names successively to the last element --->
<cfloop query = "GetEmployeeNames">
<cfset temp = ArrayAppend(myArray, "#FirstName# #LastName#")>
</cfloop>
<!--- show the resulting array as a list --->
<cfset myList = ArrayToList(myArray, ",")>
<!--- output the array as a list --->
<cfoutput>
<p>The contents of the array are as follows: </p>
<p>#myList#</p>
<p>This array has #ArrayLen(MyArray)# elements. </p>
</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**

```cfml
eh3>ArrayMax Example</h3>
<p>This example uses ArrayMax to find the largest number in an array.<br></p>
<cfif IsDefined("FORM.submit")>
  <cfset myNumberArray = ArrayNew(1)>
  <cfset myNumberArray[1] = number1>
  <cfset myNumberArray[2] = number2>
  <cfif Form.Submit is "Maximum Value">
    <!--- use ArrayMax to find the largest number in the array --->
    <p>The largest number that you entered is <cfoutput>#ArrayMax(myNumberArray)#.</cfoutput>
  </cfif>
</cfif>
</cfif>

<cfif IsDefined("FORM.submit")>
  <cfset myNumberArray = ArrayNew(1)>
  <cfset myNumberArray[1] = number1>
  <cfset myNumberArray[2] = number2>
  <cfif Form.Submit is "Maximum Value">
    <!--- use ArrayMax to find the largest number in the array --->
    <p>The largest number that you entered is <cfoutput>#ArrayMax(myNumberArray)#.</cfoutput>
  </cfif>
</cfif>
```

The following form provides two numeric fields that are compared when the form is submitted.

```cfml
<form action = "arraymax.cfm">
  <input type = "hidden" name = "number1_Float">
  <input type = "hidden" name = "number2_Float">
  <input type = "text" name = "number1">
  <input type = "text" name = "number2">
  <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.<br></p>
<!--- After checking whether the form has been submitted, this code creates an array and assigns the form fields to the first two elements. ----->
<cfif IsDefined("FORM.submit")>
<cfset myNumberArray = ArrayNew(1)>
<cfset myNumberArray[1] = FORM.number1>
<cfset myNumberArray[2] = FORM.number2>

<cfif Form.Submit is "Minimum Value">
<!--- use ArrayMin to find the smallest number in the array --->
<p>The smallest number that you entered is #ArrayMin(myNumberArray)#.</p>
</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.</td>
</tr>
<tr>
<td></td>
<td>1, 2, or 3</td>
</tr>
</tbody>
</table>

Example

```cfml
<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>
<p>It has #ArrayLen(MyNewArray)# elements.</p>
<p>Contents: #ArrayToList(MyNewArray)#</p>
<!---- 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

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

<h3>ArrayPrepend Example</h3>
<cfquery name = "GetEmployeeNames" datasource = "cfsnippets">
  SELECT FirstName, LastName FROM Employees
</cfquery>
<!--- create an array --->
<cfset myArray = ArrayNew(1)>
<!--- set element one to show where we are --->
<cfset myArray[1] = "Test Value">
<!--- loop through query. Append names successively before last element (list reverses itself from the standard queried output, as it keeps prepending the array entry) --->
<cfloop query = "GetEmployeeNames">
  <cfoutput>#ArrayPrepend(myArray, "#FirstName# #LastName#")#</cfoutput>, Array was prepended<br>
</cfloop>
<!--- show the resulting array as a list --->
<cfset myList = ArrayToList(myArray, ",")>
<!--- output the array as a list --->
<cfoutput>
  <p>The contents of the array are as follows: <p>#myList#</cfoutput>
ArrayResize

Description
Resets an array to a specified minimum number of elements. This can improve performance, if used to size an array to its expected maximum. For more than 500 elements, use `ArrayResize` immediately after using the `ArrayNew` tag.

ColdFusion arrays expand dynamically as data is added.

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

```coldfusion
<!--- perform a query to get the list --->
<cfquery name = "GetCourses" datasource = "cfsnippets">
SELECT * FROM Courses
</cfquery>
<!--- make a new array --->
<cfset MyArray = ArrayNew(1)>
<!--- resize that array to the number of records in the query --->
<cfset temp = ArrayResize(MyArray, GetCourses.RecordCount)>
<cfoutput>
The array is now #ArrayLen(MyArray)# elements, to match the query of #GetCourses.RecordCount# records.
</cfoutput>
```
ArraySet

Description
In a one-dimensional array, sets the elements in a specified index range to a value. Useful for initializing an array after a call to ArrayNew.

Returns
True, on successful completion.

Category
Array functions

Function syntax
ArraySet(array, start_pos, end_pos, value)

See also
ArrayNew

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 --->
<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>
| sort_type | • numeric: sorts numbers  
             • text: sorts text alphabetically, taking case into account (also known as case sensitive). All letters of one case precede the first letter of the other case:  
               - aabzABZ, if sort_order = "asc" (ascending sort)  
               - ZBAzbaa, if sort_order = "desc" (descending sort)  
             • textnocase: sorts text alphabetically, without regard to case (also known as case-insensitive). A letter in varying cases precedes the next letter:  
               - aAaBbBzzZ, in an ascending sort; preserves original intra-letter order  
               - ZzzBbBaAa, in a descending sort; reverses original intra-letter order |
| sort_order| • asc - ascending sort order, Default.  
             - aabzABZ or aAaBbBzzZ, depending on value of sort_type, for letters  
               - from smaller to larger, for numbers  
             • desc - descending sort order.  
               - ZBAzbaa or ZzzBbBaAa, depending on value of sort_type, for letters  
               - from larger to smaller, for numbers |
Throws

If an array element is other than a simple element, this function throws an
ArraySortSimpleValueException error. If sort_type is numeric and an array element is not
numeric, this function throws a ValueNotNumeric error.

Example

<!--- This example shows ArraySort --->
<cfquery name = "GetEmployeeNames" datasource = "cfsnippets">
SELECT FirstName, LastName FROM Employees
</cfquery>
<!--- create an array --->
<cfset myArray = ArrayNew(1)>
<!--- loop through the query and append these names successively to the last
element --->
<cfloop query = "GetEmployeeNames">
    <cfset temp = ArrayAppend(myArray, "#FirstName# #LastName#")>
</cfloop>
<!--- show the resulting array as a list --->
<cfset myList = ArrayToList(myArray, ",")>
<!--- sort that array descending alphabetically --->
<cfset isSuccessful = ArraySort(myArray, "textnocase", "desc")>
...
ArraySum

Description
Array sum function.

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 #ArraySum(myNumberArray)#.</p>
   </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 `cfsset` tags.

**Returns**
True, on successful completion.

**Category**
Array functions

**Function syntax**
`ArraySwap(array, position1, position2)`

**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**

```
<h3>ArraySwap Example</h3>

<cfset month = ArrayNew(1)>
<cfset month[1] = "February">
<cfset month[2] = "January">
<cfset temp = ArraySwap(month, 1, 2)>
<cfset temp = ArrayToList(month)>

<p>Show the results: <cfoutput>#temp#</cfoutput>
```
ArrayToList

**Description**
Converts a one-dimensional array to a list.

**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 multi-character string to separate list elements. Default: comma.</td>
</tr>
</tbody>
</table>

**Example**

```coldfusion
<h3>ArrayToList Example</h3>
<cfquery name = "GetEmployeeNames" datasource = "cfsnippets">
SELECT FirstName, LastName FROM Employees
</cfquery>
<!---- create an array --->
<cfset myArray = ArrayNew(1)>
<!---- loop through query, append names successively to last element --->
<cfloop query = "GetEmployeeNames">
<cfset temp = ArrayAppend(myArray, "+FirstName+ +LastName+")>
</cfloop>
<!---- show the resulting array as a list --->
<cfset myList = ArrayToList(myArray, ",")>
<!---- sort that array descending alphabetically --->
<cfset myAlphaArray = ArraySort(myArray, "textnocase", "desc")>
<!---- show the resulting alphabetized array as a list --->
<cfset myAlphaList = ArrayToList(myArray, ",")>
<!---- output the array as a list --->
<cfoutput>
<p>The contents of the array are as follows:
<p>#myList#
<p>This array, alphabetized by first name (descending):
<p>#myAlphaList#
<p>This array has #ArrayLen(MyArray)# elements.
</cfoutput>
```
Asc

Description
Determines the value of a character.

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>
    <cfelse>
        <!--- if it is empty, output an error message --->
        <h4>Enter a character</h4>
    </cfif>
</cfif>
</form>
```

Parameter Description
ASin

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

Example
<h3>ASin Example</h3>
<!--- output its arcsine value --->
<cfif IsDefined("FORM.SinNum")>
  <cfif IsNumeric(FORM.SinNum)>
    <cfif FORM.SinNum LESS THAN OR EQUAL TO 1>
      <cfif FORM.SinNum GREATER THAN OR EQUAL TO -1>
        ASin(<cfoutput>#FORM.SinNum#</cfoutput>) = <cfoutput>#Evaluate(ASin(FORM.SinNum))# Radians</cfoutput>
      </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>
</cfif>

<!- if it is empty, output an error message --->
<h4>Enter the sine of the angle to calculate, in degrees and radians. The value must be between 1 and -1, inclusive.</h4>
</cfif>
</cfif>

<form action = "asin.cfm">
<p>Enter a number to get its arcsine in Radians and Degrees.</p>
</form>
<br><input type="Text" name="sinNum" size="25">
<p><input type="Submit" name="" />
<input type="RESET" />
</form>
Atn

**Description**

Arctangent function. The arctangent is the angle whose tangent is *number*.

**Returns**

The arctangent, in radians, of a number.

**Category**

Mathematical functions

**Function syntax**

\[
\text{Atn}(\text{number})
\]

**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>) = <cfoutput>#Atn(FORM.AtnNum)# radians = #Evaluate(Atn(FORM.AtnNum * 180/PI())# Degrees</cfoutput>
  </cfif>
</cfif>
<!--- if it is empty, output an error message --->
<h4>Enter a number</h4>
</cfif>
</form>
```

---

398   Chapter 3: ColdFusion Functions
AuthenticatedContext

Description
This function is obsolete. Use the newer security tools; see “Authentication functions” on page 367 and Chapter 16, “Securing Applications,” in Developing ColdFusion MX Applications.

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 “Authentication functions” on page 367 and Chapter 16, “Securing Applications,” in Developing ColdFusion MX Applications.

**History**

ColdFusion MX: This function is obsolete. It does not work in ColdFusion MX and later ColdFusion releases.
BitAnd

Description
Performs a bitwise logical AND operation.

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
<h3>BitAnd Example</h3>

<p>Returns the bitwise AND of two long integers.</p>
BitAnd(5, 255): <cfoutput>@BitAnd(5, 255)@</cfoutput>
BitAnd(5, 0): <cfoutput>@BitAnd(5, 0)@</cfoutput>
BitAnd(128, 128): <cfoutput>@BitAnd(128, 128)@</cfoutput>
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>

```cfc
<cfoutput>
BitMaskClear(255, 4, 4): #BitMaskClear(255, 4, 4)#
</cfoutput>
<cfoutput>
BitMaskClear(255, 0, 4): #BitMaskClear(255, 0, 4)#
</cfoutput>
<cfoutput>
BitMaskClear(128, 0, 7): #BitMaskClear(128, 0, 7)#
</cfoutput>
```
**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><code>number</code></td>
<td>32-bit signed integer to mask</td>
</tr>
<tr>
<td><code>start</code></td>
<td>Integer, in the range 0-31, inclusive; start bit for read</td>
</tr>
<tr>
<td><code>length</code></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**

```cftml
<h3>BitMaskRead Example</h3>
<p>Returns integer created from <em>length</em> bits of <em>number</em>, beginning with <em>start</em>.</p>

```
<cfoutput>
BitMaskRead(255, 4, 4): #BitMaskRead(255, 4, 4)#
</cfoutput>
```

```
BitMaskRead(255, 0, 4): #BitMaskRead(255, 0, 4)#
</cfoutput>
```

```
BitMaskRead(128, 0, 7): #BitMaskRead(128, 0, 7)#
</cfoutput>
```
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>
<p>BitMaskSet(255, 255, 4, 4):</p>
<p>&lt;cfoutput&gt;@BitMaskSet(255, 255, 4, 4)&lt;/cfoutput&gt;</p>
<p>BitMaskSet(255, 0, 4, 4):</p>
<p>&lt;cfoutput&gt;@BitMaskSet(255, 0, 4, 4)&lt;/cfoutput&gt;</p>
<p>BitMaskSet(0, 15, 4, 4):</p>
<p>&lt;cfoutput&gt;@BitMaskSet(0, 15, 4, 4)&lt;/cfoutput&gt;</p>
**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**

```html
<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**
```
BitOr(number1, 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**

```
<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
<br><h3>BitSHLN Example</h3><br>\n<br>\n<p>Returns the number, bitwise shifted, without rotation, to the left by count bits.</p>

BitSHLN(1,1): <cfoutput>#{BitSHLN(1,1)}</cfoutput>
BitSHLN(1,30): <cfoutput>#{BitSHLN(1,30)}</cfoutput>
BitSHLN(1,31): <cfoutput>#{BitSHLN(1,31)}</cfoutput>
BitSHLN(2,31): <cfoutput>#{BitSHLN(2,31)}</cfoutput>
BitSHRN

**Description**
Perform 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>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**

```
<p>BitSHRN Example</p>

<p>Returns a number, bitwise shifted, without rotation, to the right, by count bits.</p>

<p>BitSHRN(1,1): <cfoutput>#BitSHRN(1,1)#</cfoutput></p>
<p>BitSHRN(255,7): <cfoutput>#BitSHRN(255,7)#</cfoutput></p>
<p>BitSHRN(-2147483548,1): <cfoutput>#BitSHRN(-2147483548,1)#</cfoutput>
```

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
<h3>BitXor Example</h3>
<p>Returns the bitwise XOR of two long integers.</p><p>BitXor(5,255): <cfoutput>@BitXor(5,255)@</cfoutput></p><p>BitXor(5,0): <cfoutput>@BitXor(5,0)@</cfoutput></p><p>BitXor(128,128): <cfoutput>@BitXor(128,128)@</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>
**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**

```plaintext
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/.

**Example**

```plaintext
<!--- If the character string is not empty, then output its Chr value --->
<cfif IsDefined("form.charVals")>
  <cfoutput>#form.charVals# = #Chr(form.charVals)#</cfoutput>
</cfif>

<form action="#CGI.script_name#" method="POST">
  <p>Type an integer character code from 1 to 65535<br>
     to see its corresponding character.<br></p>
  <cfinput type="Text" name="CharVals"
           range="1,65535"
           message="Please enter an integer from 1 to 65535"
           validate="integer"
           required="Yes"
           size="5"
           maxLength="5" />
  <p><input type="Submit" name="" /> <input type="RESET"></p>
</form>
```
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>
<tr>
<td>length</td>
<td>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, &quot;6&quot; Any other value causes ColdFusion to throw an error.</td>
</tr>
</tbody>
</table>

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"> </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
<h3>Compare Example</h3>
The compare function performs a case-sensitive comparison of two strings.

<cfif IsDefined("FORM.string1")>
<cfset comparison = Compare(FORM.string1, FORM.string2)>
<!--- switch on the variable to give various responses --->
<cfswitch expression = #comparison#>
<cfcase value = "-1">
<h3>String 1 is less than String 2</h3>
<i>The strings are not equal</i>
</cfcase>
<cfcase value = "0">
<h3>String 1 is equal to String 2</h3>
<i>The strings are equal!</i>
</cfcase>
<cfcase value = "1">
<h3>String 1 is greater than String 2</h3>
<i>The strings are not equal</i>
</cfcase>
<CFDEFAULTCASE>
<h3>This is the default case</h3>
</CFDEFAULTCASE>
</cfswitch>
</cfif>
<form action = "compare.cfm"/>
<p>String 1</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>
**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

**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**

```html
<!-- Begin example code -->

```<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 -->

<CF SWITCH EXPRESSION=#comparison#>

<table>
<thead>
<tr>
<th>CASE</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>String 1 is less than String 2 (string1 is less than string2)</td>
</tr>
<tr>
<td>0</td>
<td>String 1 is equal to String 2 (string1 is equal to string2)</td>
</tr>
<tr>
<td>1</td>
<td>String 1 is greater than String 2 (string1 is greater than string2)</td>
</tr>
</tbody>
</table>

<CFDEFAULTCASE>

This is the default case
```

```html

<!-- End example code -->
```
<P><INPUT TYPE="Submit" VALUE="Compare these Strings" NAME="">
   <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
<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 cosine</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

```
<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>
<!--- If input is not a number, show an error message --->
<cfelse>
<h4>You must enter a numeric angle in degrees.</h4>
</cfif>
</cfif>
<form action="#CGI.script_name#" method="post">
Enter an angle in degrees to get its cosine:
<br><input type="Text" name="cosNum" size="15">
```
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
CreateDate, CreateDateTime, CreateODBCDate

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>
<li>Formatted with CreateDateTime: #CreateDateTime(form.year, form.month, form.day, 12,13,0)#</li>
<li>Formatted with CreateODBCDate: #CreateODBCDate(yourDate)#</li>
<li>Formatted with CreateODBCDateTime: #CreateODBCDateTime(yourDate)#</li>
</ul>
</cfoutput>
</CFIF>

<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: </li>
</ul>
<CFIF>
#DateFormat(CreateODBCDateTime(yourDate), "d/m/yy")#
</cfoutput>
</CFIF>

<CFFORM ACTION="createdate.cfm" METHOD="POST">
<p>Enter the year, month and day, as integers:
<pre>
Year<CFINPUT TYPE="Text" NAME="year" VALUE="1998" VALIDATE="integer"
REQUIRED="Yes">
Month<CFINPUT TYPE="Text" NAME="month" VALUE="6" VALIDATE="integer"
REQUIRED="Yes">
Day<CFINPUT TYPE="Text" NAME="day" VALUE="8" VALIDATE="integer"
REQUIRED="Yes">
</pre>
<p><INPUT TYPE="Submit" NAME="">
<INPUT TYPE="RESET">
</cfform>
CreateDateTime

Description
Creates a date-time object.

Returns
A date/time value.

Category
Date and time functions

Function syntax
CreateDateTime(year, month, day, hour, minute, second)

See also
CreateDate, CreateTime, CreateODBCDateTime, Now

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
<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)>
</cfif>

<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>

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>
<li>Formatted with CreateODBCDate and DateFormat:
    #DateFormat(CreateODBCDate(yourDate), "mmmm d, yyyy")#
<li>Formatted with CreateODBCDateTime and DateFormat:
    #DateFormat(CreateODBCDateTime(yourDate), "d/m/yy")#
</ul>
</cfoutput>
</CFIF>

<CFORM ACTION="createdatetime.cfm" METHOD="POST">
<p>Please enter the year, month, and day, in integer format, for a date to view:
<pre>
Year<CFINPUT TYPE="Text" NAME="year" VALUE="1998" VALIDATE="integer" REQUIRED="Yes">
Month<CFINPUT TYPE="Text" NAME="month" VALUE="6" RANGE="1,12" MESSAGE="Please enter a month (1-12)" VALIDATE="integer" REQUIRED="Yes">
Day<CFINPUT TYPE="Text" NAME="day" VALUE="8" RANGE="1,31" MESSAGE="Please enter a day of the month (1-31)" VALIDATE="integer" REQUIRED="Yes">
Hour<CFINPUT TYPE="Text" NAME="hour" VALUE="16" RANGE="0,23" MESSAGE="You must enter an hour (0-23)" VALIDATE="integer" REQUIRED="Yes">
Minute<CFINPUT TYPE="Text" NAME="minute" VALUE="12" RANGE="0,59" MESSAGE="You must enter a minute value (0-59)" VALIDATE="integer" REQUIRED="Yes">
Second<CFINPUT TYPE="Text" NAME="second" VALUE="0" RANGE="0,59" MESSAGE="You must enter a value for seconds (0-59)" VALIDATE="integer" REQUIRED="Yes">
</pre>
<p><INPUT TYPE="Submit" NAME="" > <INPUT TYPE="RESET"></p></CFORM>
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:
• 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 Developing ColdFusion MX Applications.
• 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 config that contains the properties file.

CreateObject object types
For information about using this function, see these sections:
• “CreateObject: COM object” on page 424
• “CreateObject: component object” on page 425
• “CreateObject: CORBA object” on page 426
• “CreateObject: Java or EJB object” on page 428
• “CreateObject: web service object” on page 429

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

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://www.servername.com">http://www.servername.com</a></td>
</tr>
<tr>
<td></td>
<td>• <a href="http://www.servername.com">www.servername.com</a></td>
</tr>
<tr>
<td></td>
<td>• 127.0.0.1</td>
</tr>
<tr>
<td></td>
<td>If context = &quot;remote&quot;, this parameter is required.</td>
</tr>
</tbody>
</table>

Usage
The following example creates the Windows Collaborative Data Objects (CDO) for NTS NewMail object to send mail. You would use this code within a cfscript tag:
Mailer = CreateObject("COM", "CDONTS.NewMail");

For more information, see Chapter 34, “Integrating COM and CORBA Objects in CFML Applications,” in Developing ColdFusion MX Applications.
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)

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 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.

In the following example, the CFScript statements assign the tellTimeCFC variable to the tellTime component using the CreateObject function. The CreateObject function references the component in another directory. To invoke component methods, you use function syntax. For more information, see Chapter 11, “Building and Using ColdFusion Components,” in Developing ColdFusion MX Applications.

Example
<b>Server's Local Time:</b>
<cfscript>
tellTimeCFC=CreateObject("component","appResources.components.tellTime");
tellTimeCFC.getLocalTime();
</cfscript>
<br>
<b>Calculated UTC Time:</b>
<cfscript>
tellTimeCFC.getUTCTime();
</cfscript>
CreateObject: CORBA object

Description
The CreateObject function can call a method on a CORBA object. The object must be defined and registered for use.

Returns
A handle to a CORBA interface.

Function syntax
CreateObject(type, context, class, locale)

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>
</tbody>
</table>
| context   | • IOR: ColdFusion uses IOR to access CORBA server  
|           | • NameService: ColdFusion uses naming service to access server. Valid only with the InitialContext of a VisiBroker ORB. |
| class     | • If context = "ior"; absolute path of file that contains string version of the Interoperable Object Reference (IOR). ColdFusion must be able to read file; it should be local to ColdFusion server or accessible on network  
|           | • If context = "nameservice"; forward slash-delimited naming context for naming service. For example: Allaire//Doc/empobject |
| locale    | The name of the Java config that holds the properties file. For more information, see Configuring and Administering ColdFusion MX. |

Usage
In the class attribute, if "context=NameService", use a dot separator for the first part of the string. Use either of the following formats:
- "Macromedia/Eng/CF"
- "Macromedia.current/Eng.current/CF"

ColdFusion Enterprise supports CORBA through the Dynamic Invocation Interface (DII). To use this function with CORBA objects, you must provide the name of the file that contains a string version of the IOR, or the object's naming context in the naming service. You must provide the object's attributes, method names and method signatures.

This function supports user-defined types (structures, arrays, and sequences).

Example
myobj = CreateObject("corba", "d:\temp\tester.ior", "ior", "visibroker") // uses IOR
myobj = CreateObject("corba", "Macromedia/Eng/CF", 
    "nameservice", "visibroker")  // uses nameservice

myobj = CreateObject("corba", "d:\temp\tester.ior", 
    "nameservice")    // uses nameservice and default configuration
CreateObject: Java or EJB object

Description
The `CreateObject` function can create a Java object, and, by extension, an EJB object.

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></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>A Java class name</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)

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>
</tbody>
</table>

Usage
You can use the CreateObject function to create a web service.

Example
newobject2 = CreateObject("webservice","wsdlurl")
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

```coldfusion
<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>
<li>Formatted with CreateDateTime: #CreateDateTime(form.year, form.month, form.day, form.hour, form.minute, form.second)#</li>
<li>Formatted with CreateODBCDate: #CreateODBCDate(yourDate)#</li>
<li>Formatted with CreateODBCDateTime: #CreateODBCDateTime(yourDate)#</li>
</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, form.hour, form.minute, form.second), "mmm-dd-yyyy")#</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>
</cfoutput>
</CFIF>
```
<cfoutput>
</cfif>
<cfform action="createodbcdate.cfm" method="POST">
<p>Enter the year, month and day, as integers:
<pre>
Year <cfinput type="text" name="year" value="1998" validate="integer" required="yes">
Month <cfinput type="text" name="month" value="6" range="1,12" message="please enter a month (1-12)" validate="integer" REQUIRED="Yes">
Day <cfinput type="text" name="day" value="8" range="1,31" MESSAGE="Please enter a day of the month (1-31)" VALIDATE="integer" REQUIRED="Yes">
Hour <cfinput type="text" name="hour" value="16" range="0,23" MESSAGE="You must enter an hour (0-23)" VALIDATE="integer" REQUIRED="Yes">
Minute <cfinput type="text" name="minute" value="12" range="0,59" MESSAGE="You must enter a minute value (0-59)" VALIDATE="integer" REQUIRED="Yes">
Second <cfinput type="text" name="second" value="0" range="0,59" MESSAGE="You must enter a value for seconds (0-59)" VALIDATE="integer" REQUIRED="Yes">
</pre>
<p>
<input type="Submit" name="" />
<input type="RESET">
</cfform>
CreateODBCDateTime

Description
Creates an ODBC date-time object.

Returns
A date/time object, in ODBC timestamp format.

Category
Date and time functions

Function syntax
CreateODBCDateTime(date)

See also
CreateDateTime, CreateODBCDate, CreateODBCTime, Now

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))#
<li>Formatted with CreateODBCDate and DateFormat: #DateFormat(CreateODBCDate(yourDate), "mmmm d, yyyy")#
<li>Formatted with CreateODBCDateTime and DateFormat: #DateFormat(CreateODBCDateTime(yourDate), "d/m/yy")#
</ul>
</cfoutput>
</cfif>
<CFORM ACTION="createodbcdatetime.cfm" METHOD="POST">
<p>Enter a 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" RANGE="1,12"
   MESSAGE="Enter a month (1-12)" VALIDATE="integer" REQUIRED="Yes">
Day <cfinput type="text" name="day" value="8" range="1,31"
   MESSAGE="Enter a day of the month (1-31)" VALIDATE="integer"
   REQUIRED="Yes">
Hour <cfinput type="text" name="hour" value="16" range="0,23"
   MESSAGE="You must enter an hour (0-23)" VALIDATE="integer"
   REQUIRED="Yes">
Minute<cfinput type="text" name="minute" value="12" range="0,59"
   MESSAGE="You must enter a minute value (0-59)" VALIDATE="integer"
   REQUIRED="Yes">
Second<cfinput type="text" name="second" value="0" range="0,59"
   MESSAGE="You must enter a seconds value (0-59)" VALIDATE="integer"
   REQUIRED="Yes">
</pre>
<p><INPUT TYPE="Submit" NAME=""> <INPUT TYPE="RESET"></p>
</CFORM>
CreateODBCTime

Description

Creates an ODBC time object.

Returns

A time object, in ODBC timestamp format.

Category

Date and time functions

Function syntax

CreateODBCTime(date)

See also

CreateODBCDateTime, CreateTime

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

<h3>CreateODBCTime Example</h3>
<cfif IsDefined("form.hour")>
Your time value, created with CreateTime...
<cfset yourTime = CreateTime(form.hour, form.minute, form.second)>
<cfoutput>
<ul>
<li>Formatted with CreateODBCTime: #CreateODBCTime(yourTime)#
<li>Formatted with TimeFormat: #TimeFormat(yourTime)#
</ul></cfoutput>
</cfif>
</cfform>
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

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

```cfml
<h3>CreateTime Example</h3>
<cffif IsDefined("FORM.hour")>
  Your time value, presented using CreateTime time function:
  <cfset yourTime = CreateTime(FORM.hour, FORM.minute, FORM.second)>
  <cfoutput>
    Formatted with timeFormat: #TimeFormat(yourTime)#
    Formatted with timeFormat and hh:mm:ss: #TimeFormat(yourTime, 'hh:mm:ss')#
  </cfoutput>
</cffif>
</cfml>
```

```cfml
<cfform action="createtime.cfm" method="post">
  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" />
</cfform>
```

CreateTime 435
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
CreateDateDateTime, DateAdd, DateConvert

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 = "cfsnippets" cachedWithin = "#CreateTimeSpan(0, 6, 0, 0)#">
SELECT   PARKNAME, REGION, STATE
FROM     Parks
ORDER by ParkName, State
</cfquery>
</cfoutput>
<tr><td colspan=2 bgcolor=f0f0f0><b><i>Park Name</i></b></td><td bgcolor=f0f0f0><b><i>Region</i></b></td><td bgcolor=f0f0f0><b><i>State</i></b></td></tr>
<tr><td valign=top bgcolor=ffffed><b>#GetParks.CurrentRow#/</b></td><td valign=top><font size=-1>#ParkName#</font></td><td valign=top><font size=-1>#Region#</font></td><td valign=top><font size=-1>#State#</font></td></tr>
</table>

<tr><td colspan=4><cfif (StartRow + MaxRows) LTE GetParks.RecordCount>
<a href="cfquery.cfm?startrow=#Evaluate(StartRow + MaxRows)#">See next #MaxRows# rows/A</a>
</cfif></tr>
</table>
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-xxxxxx-xxxxxxxxxx (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>
<p>This example uses CreateUUID to generate a UUID when you submit the form. You can submit the form more than once. </p>
<!---- Checks whether the form was submitted; if so, creates UUID. --->
<cfif IsDefined("Form.CreateUUID") Is True>
<hr>
<p>Your new UUID is: <cfoutput>#CreateUUID()#</cfoutput></p>
</cfif>
<form action = "createuuid.cfm">
<p><input type = "Submit" name = "CreateUUID"> </p>
</form>
DateAdd

Description
Adds units of time to a date.

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)</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">

<!--- if numbers passed, then use those --->
<cfif IsDefined("form.value")>
  <cfset value = form.value>
</cfif>

DateAdd
<cfif IsDefined("form.type")>
  <cfset type = form.type>
</cfif>

<cfquery name="GetMessages" datasource="cfsnippets">
SELECT UserName, Subject, Posted
FROM Messages
</cfquery>

<p>This example uses DateAdd to determine when a message in
the database will expire. 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>
</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">
</cfform>
**DateCompare**

**Description**
Performs a full date/time comparison of two dates.

**Returns**
- -1, if `date1` is less than `date2`
- 0, if `date1` is equal to `date2`
- 1, if `date1` is greater than `date2`

**Category**
Date and time functions

**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>
</tbody>
</table>
| datePart  | Optional. String. Precision of the comparison.  
|           | • s Precise to the second (default)  
|           | • n Precise to the minute  
|           | • h Precise to the hour  
|           | • d Precise to the day  
|           | • m Precise to the month  
|           | • yyyy Precise to the year |

**Usage**
When passing a date/time object as a string, you must enclose it in quotation marks. Otherwise, it is interpreted as a numeric representation of a date/time object.

**Example**
```
<p>The DateCompare function compares two date/time values.
<cfif IsDefined("FORM.date1")>
  <cfif IsDate(FORM.date1) and IsDate(FORM.date2)>
    <cfset comparison = DateCompare(FORM.date1, FORM.date2, FORM.precision)>
    <!--- switch on the variable to give various responses --->
    <cfswitch expression = #comparison#>
      <cfcase value = "-1">
        <h3><cfoutput>#DateFormat(FORM.date1)# #TimeFormat(FORM.date1)##</cfoutput> (Date 1) is earlier than <cfoutput>#DateFormat(FORM.date2)# #TimeFormat(FORM.date2)#</cfoutput> (Date 2)</h3>
        <i>The dates are not equal</i>
      </cfcase>
      <cfcase value = "0">
        <h3><cfoutput>#DateFormat(FORM.date1)# #TimeFormat(FORM.date1)##</cfoutput> (Date 1) is equal to <cfoutput>#DateFormat(FORM.date2)# #TimeFormat(FORM.date2)#</cfoutput> (Date 2)</h3>
      </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>
    </cfswitch>
</cfif>
</cfif>
```
Chapter 3: ColdFusion Functions

442

<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>

<cfcase value = "1">
  <h3><cfoutput>%DateFormat(FORM.date1)% %TimeFormat(FORM.date1)%</cfoutput> (Date 1) is later than <cfoutput>%DateFormat(FORM.date2)% %TimeFormat(FORM.date2)%</cfoutput> (Date 2)</h3>
  <i>The dates are not equal!</i>
</cfcase>
<cfdefaultcase>
  <h3>This is the default case</h3>
</cfdefaultcase>
</cfswitch>

<cfelse>
  <h3>Enter two valid date values</h3>
</cfif>
</cfif>
</form>

<form action = "datecompare.cfm">
  <hr size = "2" color = "#0000A0">
  <p>Date 1
  <br><input type = "Text" name = "date1" value = "%DateFormat(Now())% %TimeFormat(Now())%">
</cfoutput"></p>
  <p>Date 2
  <br><input type = "Text" name = "date2" value = "%DateFormat(Now())% %TimeFormat(Now())%">
</cfoutput"></p>
  <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>
</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><cfoutput>The current date and time: #curDate#. </cfoutput></p>
<cfset utcDate = DateConvert("local2Utc", curDate)>
<cfoutput>
  <p>The current date and time converted to UTC time: #utcDate#.</p>
</cfoutput>
<!--- This code checks whether form was submitted. If so, the code generates the CFML date with the CreateDateTime function. --->
<cfif IsDefined("FORM.submit")>
  <cfset yourDate = CreateDateTime(FORM.year, FORM.month, FORM.day, FORM.hour, FORM.minute, FORM.second)>
  <cfoutput>Your date value, presented as a ColdFusion date/time string: #yourdate#. </cfoutput></cfif>
<cfset yourUTC = DateConvert("local2utc", yourDate)>
<p><cfoutput>Your date and time value, converted to Coordinated Universal Time (UTC): #yourUTC#. </cfoutput></p>
```
Your UTC date and time, converted back to local date and time:
#DateConvert("utc2local", yourUTC)#.

Type the date and time, and press Enter to see the conversion.

Enter year, month and day in integer format for date value to view:

<table>
<thead>
<tr>
<th>Year</th>
<th>Month</th>
<th>Day</th>
<th>Hour</th>
<th>Minute</th>
<th>Second</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;input type=&quot;Text&quot; name=&quot;year&quot; value=&quot;1998&quot; validate=&quot;integer&quot; required=&quot;Yes&quot;&gt;</td>
<td>&lt;input type=&quot;Text&quot; name=&quot;month&quot; value=&quot;6&quot; range=&quot;1,12&quot; message=&quot;Enter a month (1-12)&quot; validate=&quot;integer&quot; required=&quot;Yes&quot;&gt;</td>
<td>&lt;input type=&quot;Text&quot; name=&quot;day&quot; value=&quot;8&quot; range=&quot;1,31&quot; message=&quot;Enter a day of the month (1-31)&quot; validate=&quot;integer&quot; required=&quot;Yes&quot;&gt;</td>
<td>&lt;input type=&quot;Text&quot; name=&quot;hour&quot; value=&quot;16&quot; range=&quot;0,23&quot; message=&quot;You must enter an hour (0-23)&quot; validate=&quot;integer&quot; required=&quot;Yes&quot;&gt;</td>
<td>&lt;input type=&quot;Text&quot; name=&quot;minute&quot; value=&quot;12&quot; range=&quot;0,59&quot; message=&quot;You must enter a minute value (0-59)&quot; validate=&quot;integer&quot; required=&quot;Yes&quot;&gt;</td>
<td>&lt;input type=&quot;Text&quot; name=&quot;second&quot; value=&quot;0&quot; range=&quot;0,59&quot; message=&quot;You must enter a value for seconds (0-59)&quot; validate=&quot;integer&quot; required=&quot;Yes&quot;&gt;</td>
</tr>
</tbody>
</table>

<input type="Submit" name="submit" value="Submit"> <input type="RESET">
**DateDiff**

**Description**
Determines the integer number of units by which `date1` is less than `date2`.

**Returns**
A number of units, of type `datepart`.

**Category**
Date and time functions

**Function syntax**
`DateDiff(datepart, date1, date2)`

**See also**
`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 `w` and `ww` attributes; 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 <code>yyyy</code> requests a date difference in whole years.</td>
</tr>
<tr>
<td></td>
<td>• <code>yyyy</code>: Years</td>
</tr>
<tr>
<td></td>
<td>• <code>q</code>: Quarters</td>
</tr>
<tr>
<td></td>
<td>• <code>m</code>: Months</td>
</tr>
<tr>
<td></td>
<td>• <code>y</code>: Days of year (same as <code>d</code>)</td>
</tr>
<tr>
<td></td>
<td>• <code>d</code>: Days</td>
</tr>
<tr>
<td></td>
<td>• <code>w</code>: Weekdays (same as <code>ww</code>)</td>
</tr>
<tr>
<td></td>
<td>• <code>ww</code>: Weeks</td>
</tr>
<tr>
<td></td>
<td>• <code>h</code>: Hours</td>
</tr>
<tr>
<td></td>
<td>• <code>n</code>: Minutes</td>
</tr>
<tr>
<td></td>
<td>• <code>s</code>: 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 `DateDiff` function determines the number of complete `datepart` units between the two dates; for example, if the `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 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="t">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>.
      <cfelse>
        <p>The two dates are equal! Try changing one of the values ...
      </cfelseif>
    </cfif>
  </cfif>
  <cfelse>
    <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
</pre>
</form>
```
<input type="Text" name="date1" value="<CFOUTPUT>#DateFormat(Now())#</CFOUTPUT>">
Date 1
<input type="Text" name="date2" value="<CFOUTPUT>#DateFormat(Now())#</CFOUTPUT>">
Date 2
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 LSDateFormat.

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, LSDateFormat, LSParseDateTime, LSTimeFormat, TimeFormat, ParseDateTime

History
ColdFusion MX: Added support for the following mask attribute 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.  
|           | • ddd: Day of the week as a three-letter abbreviation.  
|           | • dddd: Day of the week as its full name.  
|           | • m: Month as digits; no leading zero for single-digit months.  
|           | • mm: Month as digits; leading zero for single-digit months.  
|           | • mmm: Month as a three-letter abbreviation.  
|           | • mmmm: Month as its full name.  
|           | • y: Year as last two digits; no leading zero for years less than 10.  
|           | • yy: Year as last two digits; leading zero for years less than 10.  
|           | • yyyy: Year represented by four digits.  
|           | • gg: Period/era string. Ignored. Reserved.  
|           | 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 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 result set values. For more information and examples, see TechNote 22183, "ColdFusion Server (5 and 4.5.x) with Oracle: Formatting Date and Time Query Results," on our website at www.coldfusion.com/Support/KnowledgeBase/SearchForm.cfm.

Example

```cfml
<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:
<cfoutput>
<ul>
  <li>#DateFormat(todayDate)#+</li>
  <li>#DateFormat(todayDate, "mmm-dd-yyyy")#</li>
  <li>#DateFormat(todayDate, "mmmm d, yyyy")#</li>
  <li>#DateFormat(todayDate, "mm/dd/yyyy")#</li>
  <li>#DateFormat(todayDate, "d-mmm-yyyy")#</li>
  <li>#DateFormat(todayDate, "ddd, mmmm dd, yyyy")#</li>
  <li>#DateFormat(todayDate, "short")#</li>
  <li>#DateFormat(todayDate, "medium")#</li>
  <li>#DateFormat(todayDate, "long")#</li>
  <li>#DateFormat(todayDate, "full")#</li>
</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>
<dl>
<dt>year</dt><dd>#DatePart("yyyy", todayDate)#</dd>
<dt>quarter</dt><dd>#DatePart("q", todayDate)#</dd>
<dt>month</dt><dd>#DatePart("m", todayDate)#</dd>
<dt>day of year</dt><dd>#DatePart("y", todayDate)#</dd>
</dl>
</cfoutput>
<ul>
  <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("m", todayDate)#</li>
  <li>second: #DatePart("s", todayDate)#</li>
</ul>
</cfoutput>
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**
```coldfusion
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 `CreateDate` function or `Now` function as the `date` parameter of this function; for example: `#Day(CCreateDate(2001, 3, 3))#`

**Example**
```coldfusion
<h3>Day Example</h3>
<cfif IsDefined("FORM.year")>
<p>More information about your date:
<cfset yourDate = CreateDate(FORM.year, FORM.month, FORM.day)>
<cfoutput>
<p>Date: #DateFormat(yourDate)#. <br>
It is #DayOfWeekAsString(DayOfWeek(yourDate))#, day #DayOfWeek(yourDate)# in the week. <br>
This is day #Day(yourDate)# in the month of #MonthAsString(Month(yourDate))#, which has #DaysInMonth(yourDate)# days. <br>
We are in week #Week(yourDate)# of #Year(YourDate)#. <br>
This is a leap year #IsLeapYear(Year(yourDate))#<cfelse>This is not a leap year</cfif>
</cfoutput>
</cfif>
```

452 Chapter 3: ColdFusion Functions
DayOfWeek

Description
Determines 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>
  <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)# (#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></h3>```
DayOfWeekAsString

Description
Determines the day of the week, in a date, as a string function.

Returns
The day of the week, as a string that corresponds to day_of_week.

Category
Date and time functions, String functions

Function syntax
DayOfWeekAsString(day_of_week)

See also
Day, DayOfWeek, DayOfMonth, DaysInMonth, DaysInYear, FirstDayOfMonth

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
<h3>DayOfWeekAsString Example</h3>
<cfif IsDefined("FORM.year")>
More information about your date:
<cfset yourDate = CreateDate(FORM.year, FORM.month, FORM.day)>

<cfoutput>
<p>Your date, #DateFormat(yourDate)#.
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>
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

```cfm
<h3>DayOfYear Example</h3>
<cfif IsDefined("FORM.year")>
  <cfset yourDate = CreateDate(FORM.year, FORM.month, FORM.day)>
  <cfoutput>
    <p>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.
        <br>We are in week #Week(yourDate)# of #Year(yourDate)#
        day #DayOfYear(yourDate)# of #DaysInYear(yourDate)#.
        <cfif IsLeapYear(Years(yourDate))>
          This is a leap year
        <cfelse>This is not a leap year</cfif>
  </cfoutput>
</cfif>
</cfm>
```
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 theCreateDate 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)#.
        <cfif IsLeapYear(Year(yourDate))>This is a leap year</cfif>
        <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

```cftmpl
<h3>DaysInYear Example</h3>
<cfif IsDefined("FORM.year")>
  More information about your date:
  <cfset yourDate = CreateDate(FORM.year, FORM.month, FORM.day)>
  <cfoutput>
    <p>Your date, #DateFormat(yourDate)#.
    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
Postpones evaluation of a string as an expression, when it is passed as a parameter to the IIf or Evaluate functions. Escapes any double quotation marks in the parameter and wraps the result in double quotation marks.

This function is especially useful with the IIf function, to prevent the function from evaluating a string that is to be output.

This applies to expressions that are not surrounded by pound signs. (If pound signs surround any part of an expression, a ColdFusion function evaluates that part first, regardless of whether the DE function is present.)

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

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
Consider this example:

```cfc
<cfoutput>
<condition> <true expression> <false expression>
  IIf( 1 eq 2, DE("#Var1#"). DE("#Var2#"))
</cfoutput>
```

ColdFusion evaluates whatever is surrounded by pounds in the expression before executing it. So, although this expression is never true (because Var1 does not exist), the expression fails with an 'Error Resolving Parameter' error, because ColdFusion evaluates #Var1#' and #Var2#' before executing either expression.

This example returns 'Var2':

```cfc
IIf( 1 eq 2, DE('Var1'). DE('Var2'))
```

The following example uses IIF to alternate table-row background colors, white and gray. It uses the DE function to prevent ColdFusion from evaluating the color strings.

```cfc
<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>
  </cfloop>
</table>
```

For more information and code examples, see Chapter 4, “Using Expressions and Pound Signs,” in Developing ColdFusion MX Applications.

Example

<!--- This example shows the use of DE and Evaluate --->
<h3>DE Example</h3>
<cfif IsDefined("FORM.myExpression")>
<h3>The Expression Result</h3>
<cftry>
<!--- Evaluate the expression --->
<cfset myExpression = Evaluate(FORM.myExpression)>
<!--- Use DE to output the value of the variable, unevaluated --->
<cfoutput>
<I>The value of the expression #Evaluate(DE(FORM.MyExpression))# is #myExpression#.</I>
</cfoutput>
<!--- specify the type of error for which we are searching --->
<cfcatch type = "Any">
<!--- the message to display --->
<h3>Sorry, there's been an <B>Error</B>. Try a simple expression, such as "2+2".</h3>
<cfoutput>
<!--- and the diagnostic message from ColdFusion Server --->
<p>#cfcatch.message#</cfoutput>
</cfcatch>
</cftry>
</cfif>
DecimalFormat

Description
Converts a number to a decimal-formatted string.

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
<h3>DecimalFormat Function</h3>
<p>Returns a number to two decimal places.</p>
<p>
<cfloop FROM = 1 TO = 20 INDEX = "counter">
  <cfoutput>
    #counter# * Square Root of 2:
    #DecimalFormat(Evaluate(counter * sqr(2))#
  </cfoutput>
</cfloop>
DecrementValue

Description
Decrement the integer part of a number.

Returns
Integer part of number, decremented by one.

Category
Mathematical functions

Function syntax
DecrementValue(number)

See also
IncrementValue

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>number</td>
<td>Number to decrement</td>
</tr>
</tbody>
</table>

Example
<h3>DecrementValue Example</h3>
<p>Returns the integer part of a number decremented by one.</p>
<p>DecrementValue(0):</p>
<cfoutput>#DecrementValue(0)#</cfoutput>
<p>DecrementValue("1"): </p>
<cfoutput>#DecrementValue("1")#</cfoutput>
<p>DecrementValue(123.35):</p>
<cfoutput>#DecrementValue(123.35)#</cfoutput>
Decrypt

Description
Decrypts a string that is encrypted with the Encrypt function.

Returns
String, unencrypted.

Category
Other functions, String functions

Function syntax
Decrypt(encrypted_string, seed)

See also
Duplicate, Encrypt

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>encrypted_string</td>
<td>String or a variable that contains one. String to decrypt</td>
</tr>
<tr>
<td>seed</td>
<td>String. The 32-bit key that was used to encrypt the string.</td>
</tr>
</tbody>
</table>

Example
<!--- This example shows the use of Encrypt and Decrypt --->
<p>This function encrypts/decrypts a string. Enter a string and a key.
<cfif IsDefined("FORM.myString")>
  <cfset string = FORM.myString>
  <cfset key = FORM.myKey>
  <cfset encrypted = encrypt(string, key)>
  <cfset decrypted = decrypt(encrypted, key)>
  <cfoutput>
    <h4><B>The string:</B></h4> #string# <br>
    <h4><B>The key:</B></h4> #key#<br>
    <h4><B>Encrypted:</B></h4> #encrypted#<br>
    <h4><B>Decrypted:</B></h4> #decrypted#<br>
  </cfoutput>
</cfif>
<form action = "encrypt.cfm">
  <p>Input your key: <p><input type = "Text" name = "myKey" value = "foobar">
  <p>Enter string to encrypt: <p><textarea name = "myString" cols = "40" rows = "5" WRAP = "VIRTUAL">This string will be encrypted (try typing some more)</textarea>
  <input type = "Submit" value = "Encrypt my String">
</form>
DeleteClientVariable

Description
Deletes a client variable. (To test for the existence of a variable, use IsDefined.)

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.
<!---
<cfset client.somevar = ">
<cfset client.user_id = ""
<p>Client variable list:<cfoutput>#GETCLIENTVARIABLESLIST()#</cfoutput>
<cfif ListFindNoCase(GETCLIENTVARIABLESLIST(), "User_ID") is not 0>

<cfset temp = DeleteClientVariable("User_ID")>
<p>Was variable "User_ID" Deleted? <cfoutput>#temp#</cfoutput>
</cfif>
<p>Amended Client variable list:<cfoutput>#GETCLIENTVARIABLESLIST()#</cfoutput>
--->

---
DirectoryExists

Description
Determines whether a directory exists.

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

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>
      Your directory exists. Directory name: #yourDirectory#
    </cfoutput>
  </cfif>
  <cfelse>
    <p>Your directory does not exist.</p>
  </cfif>
</cfif>
</cfif>
DollarFormat

Description
Formats a string in U.S. format. (For other currencies, use LSCurrencyFormat or LSEuroCurrencyFormat.)

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 --->
<html>
<head>
<title>DollarFormat Example</title>
</head>
<body>
<h3>DollarFormat Example</h3>
<cfloop from = 8 to = 50 index = counter>
  <cfset full = counter>
  <cfset quarter = Evaluate(counter + (1/4))>
  <cfset half = Evaluate(counter + (1/2))>
  <cfset threefourth = Evaluate(counter + (3/4))>
  <cfoutput>
  <pre>
  bill#DollarFormat(full)##DollarFormat(quarter)##DollarFormat(half)##DollarFormat(threefourth)##
  18% tip#DollarFormat(Evaluate(full * (18/100)))##DollarFormat(Evaluate(quarter * (18/100)))##
  #DollarFormat(Evaluate(half * (18/100)))##
  #DollarFormat(Evaluate(threefourth * (18/100)))##
  </pre>
  </cfoutput>
</cfloop>
</body>
</html>
Duplicate

Description
Returns a clone, also known as a deep copy, of a variable. There is no reference to the original variable.

Returns
A clone of a variable.

Category
Structure functions, System functions

Function syntax
Duplicate(<variable_name>)

See also
StructCopy, other Structure functions

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:** With this function, you cannot duplicate a COM, CORBA, or JAVA object returned from the cfobject tag or the CreateObject function. If an array element or structure field is a COM, CORBA, or JAVA object, you cannot duplicate the array or structure.

Example

```<h3>Duplicate Example</h3>
<cfset s1 = StructNew()>
<cfset s1.nested = StructNew()>
<cfset s1.nested.item = "original">
<cfset copy = StructCopy(s1)>
<cfset clone = Duplicate(s1)>
<cfset s1.nested.item = "modified">
<cfset s1.nested.item = "modified">
<cfoutput>
<p>The copy contains the modified value: #copy.nested.item#</p>
<p>The duplicate contains the original value: #clone.nested.item#</p>
</cfoutput>```
Encrypt

Description
Encrypts a string. Uses a symmetric key-based algorithm, in which the same key is used to encrypt and decrypt a string. The security of the encrypted string depends on maintaining the secrecy of the key. Uses an XOR-based algorithm that uses a pseudo-random 32-bit key, based on a seed passed by the user as a function parameter.

Returns
String; can be much longer than the original string.

Category
Other functions, String functions

Function syntax
Encrypt(string, seed)

See also
Decrypt

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>seed</td>
<td>String. Seed used to generate 32-bit encryption key. Can be any combination of any number of characters.</td>
</tr>
</tbody>
</table>

Example

This function allows for the encryption and decryption of a string. Try it by entering a string and a key to see the results.

```
<cfif IsDefined("FORM.myString")>
  <cfset string = FORM.myString>
  <cfset key = FORM.myKey>
  <cfset encrypted = encrypt(string, key)>
  <cfset decrypted = decrypt(encrypted, key)>
  <cfoutput>
    <h4><B>The string:</B></h4> #string# <br>
    <h4><B>The key:</B></h4> #key#<br>
    <h4><B>Encrypted:</B></h4> #encrypted#<br>
    <h4><B>Decrypted:</B></h4> #decrypted#<br>
  </cfoutput>
</cfif>

<form action = "encrypt.cfm" method="post">
  <p>Input your key:
  <p><input type = "Text" name = "myKey" value = "foobar">
  <p>Input your string to be encrypted:
  <p><textarea name = "myString" cols = "40" rows = "5" WRAP = "VIRTUAL">
  This string will be encrypted (try typing some more)
  </textarea>
  <input type = "Submit" value = "Encrypt my String">
</form>

Encrypt 467
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, it must be escaped.

This function is useful for forming one variable from multiple variables. For example, to reference a column of the query qNames with a variable, var, using an index value to traverse rows, you could use the following code:

```coldfusion
<cfset var=Evaluate("qNames.#colname#[index]")>
```

For more information, see Chapter 4, “Using Expressions and Pound Signs,” in Developing ColdFusion MX Applications.

Example

```
<!--- This shows the use of DE and Evaluate --->
<h3>Evaluate Example</h3>
<cfif IsDefined("FORM.myExpression")>
<h3>The Expression Result</h3>
<cftry>
<!--- Evaluate the expression --->
<cfset myExpression = Evaluate(FORM.myExpression)>
<!--- Use DE to output the value of the variable, unevaluated --->
<cfoutput>
The value of the expression #Evaluate(DE(FORM.MyExpression))# is #myExpression#.</cfoutput>
</cfoutput>
```

...
Exp

Description
Calculates the exponent whose base is e that represents number. The constant e equals 2.71828182845904, the base of the natural logarithm. This function is the inverse of log, the natural logarithm of number.

Returns
The constant e, raised to the power of number.

Category
Mathematical functions

Function syntax
Exp(number)

See also
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
<cfif IsDefined("FORM.Submit")>
  <cfoutput>
    <p>Your number, #FORM.number#<br>
    #FORM.number# raised to the E power: #exp(FORM.number)#
  </cfif FORM.number LTE 0>
    <br>You must enter a positive real number to see its natural logarithm
  <cfelse>br>
    The natural logarithm of #FORM.number#: #log(FORM.number)#
  </cfif>
  </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.
<input type = "Text" name = "number" message = "You must enter a number" validate = "float" required = "No">
<input type = "Submit" name = "Submit">
ExpandPath

Description

Creates an absolute, platform-appropriate path that is equivalent to the value of `relative_path`, appended to the base path. This function (despite its name) can accept an absolute or relative path in the `relative_path` attribute.

The base path is the currently executing page's directory path. It is stored in `pageContext.getServletContext()`.

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` attribute: this function can now accept an absolute or relative path in the `relative_path` attribute. To resolve a path, this function uses virtual mappings that are defined in the ColdFusion Administrator. This function does not reliably use virtual mappings that are defined in IIS, Apache, or other Web servers.

Parameters

<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, (<code>\</code> and <code>.</code>) to convert to an absolute path. Can include forward 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("*.cfa")`
- `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>

```cfc
<cfset thisPath=ExpandPath("*.cfa")>
<cfset thisDirectory=GetDirectoryFromPath(thisPath)>
<cfoutput>
```
The current directory is: #GetDirectoryFromPath(thisPath)#

<cfif IsDefined("form.yourFile")>
<cfif form.yourFile is not ">
<cfset yourFile = form.yourFile>
<cfif FileExists(ExpandPath(yourFile))>
<p>Your file exists in this directory. You entered the correct file name. #GetFileFromPath(#thisPath#/yourfile#)##
<CFELSE>
<p>Your file was not found in this directory:
<br>Here is a list of the other files in this directory:
<!--- use CFDIRECTORY to give the contents of the snippets directory, order by name and size --->
<CFDIRECTORY DIRECTORY="#thisDirectory#" NAME="myDirectory" SORT="name ASC, size DESC">
<!--- Output the contents of the CFDIRECTORY as a CFTABLE --->
<CFTABLE QUERY="myDirectory">
<CFCOL HEADER="NAME:" TEXT="#Name#">
<CFCOL HEADER="SIZE:" TEXT="#Size#">
</CFTABLE>
</cfif>
</cfif>
<cfelse>
<h3>Please enter a file name</h3>
</CFIF>
</cfoutput>

<FORM action="expandpath.cfm" METHOD="post">
<h3>Enter the name of a file in this directory <I> (try expandpath.cfm)</I>
<INPUT TYPE="Text" NAME="yourFile">
<INPUT TYPE="Submit" NAME="">
</form>
**FileExists**

**Description**
Determines 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>

**Example**

```
<h3>FileExists Example</h3>

<cfset thisPath = ExpandPath("*.**")>
<cfset thisDirectory = GetDirectoryFromPath(thisPath)>
<cfoutput>
The current directory is: #GetDirectoryFromPath(thisPath)#
<cfif IsDefined("FORM.yourFile")>
<cfif FORM.yourFile is not ">
<cfset yourFile = FORM.yourFile>
<cfif FileExists(ExpandPath(yourfile))>
<p>Your file exists in this directory. You entered the correct file name, #GetFileFromPath("#thisPath#/yourfile")#</p>
<cfelse>
```
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>
<br>
#findnocase("the",stringToSearch)#<br>
#findnocase("the",stringToSearch,5)#<br>
#findnocase("no such substring",stringToSearch)#<br>
<br>
#findoneof("aeiou",stringToSearch)#<br>
#findoneof("aeiou",stringToSearch,4)#<br>
#findoneof("@%^()",stringToSearch)#<br>
</cfoutput>
FindNoCase

Description
Finds the first occurrence of a substring in a string, from a specified start position. If substring is not in string, returns zero. The search is case-insensitive.

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
<cfset stringToSearch = "The quick brown fox jumped over the lazy dog.">
#find("the",stringToSearch)#<br>
#find("the",stringToSearch,35)#<br>
#find("no such substring",stringToSearch)#<br>
<br>
#findnocase("the",stringToSearch)#<br>
#findnocase("the",stringToSearch,5)#<br>
#findnocase("no such substring",stringToSearch)#<br>
<br>
#findoneof("aeiou",stringToSearch)#<br>
#findoneof("aeiou",stringToSearch,4)#<br>
#findoneof("@%^*()",stringToSearch)#<br>
FindOneOf

Description
Finds the first occurrence of *any one of a set of characters* in a *string*, from a specified start position. The search is case-sensitive.

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. String 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>
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
<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
\texttt{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

\texttt{\textless h3>Fix Example\textgreater }
\texttt{\textless 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.\textgreater }
\texttt{\textless cfoutput>}
\texttt{\textless p>The fix of 3.4 is \#Fix(3.4)\#\textgreater }
\texttt{\textless p>The fix of 3 is \#Fix(3)\#\textgreater }
\texttt{\textless p>The fix of 3.8 is \#Fix(3.8)\#\textgreater }
\texttt{\textless p>The fix of -4.2 is \#Fix(-4.2)\#\textgreater }
\texttt{\textless /cfoutput>}

Fix 477
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**

```cfoutput
<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>
```
GetAuthUser

**Description**
Gets the name of an authenticated user.

**Returns**
The name of an authenticated user.

**Category**
Authentication functions

**Function syntax**
GetAuthUser(

**See also**
IsUserInRole, cflogin, cfloginuser, Chapter 16, "Securing Applications," in Developing ColdFusion MX Applications

**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**

```cml
<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

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Req/Opt</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tagname</td>
<td>Required</td>
<td></td>
<td>Ancestor tag name for which to return data</td>
</tr>
<tr>
<td>instancenumber</td>
<td>Optional</td>
<td>1</td>
<td>Number of ancestor levels to jump before returning data</td>
</tr>
</tbody>
</table>

**Example**
```
<!--- This example shows the use of GetBaseTagData function. Typically used in custom tags.--->
...
<cfif trim(inCustomTag) neq "">
  <cfoutput>
    Running in the context of a custom tag named #inCustomTag#.<p>
  </cfoutput>
  <!--- Get the tag instance data --->
  <cfset tagData = GetBaseTagData(inCustomTag)>
  <!--- Find the tag's execution mode --->
  Located inside the 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
GetBaseTagData

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,MYZAGNAME..." --->
<cfset ancestorList = GetBaseTagList()>
  <br><br>Dump of GetBaseTagList output:<br>
  <cfdump var="#ancestorList#"><br><br>
  <!--- Output current tag name --->
  <!--- 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**

```html
<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 --->
<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(). This example requires the existence of an Application.cfm file and that client management be in effect.
</p>
<cfset client.somevar = "">
<cfset client.user_id = "">
<p>Client variable list:<cfoutput>#GetClientVariablesList()#</cfoutput></p>
<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></p>
</cfif>
<p>Amended Client variable list:<cfoutput>#GetClientVariablesList()#</cfoutput></p>
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: <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

```cfset thisPath = ExpandPath("*.*")>
<cfset thisDirectory = GetDirectoryFromPath(thisPath)>
<cfoutput>
The current directory is: #GetDirectoryFromPath(thisPath)#
</cfoutput>
<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>
    </cfif>
  </cfif>
</cfif>
<cfelse>
<p>Your file was not found in this directory:</p>
<cfoutput>
Here is a list of the other files in this directory:
</cfoutput>
<cfoutput>
<!-- use cfdirectory show directory, order by name & size -->
</cfoutput>
<cfoutput>
<!-- Output the contents of the cfdirectory as a CFTABLE -->
</cfoutput>
<cfoutput>
</cfoutput>
<cfelse>
<p>Please enter a file name</p>
</cfif>
```

</cfif>
</cfoutput>
<form action="getdirectoryfrompath.cfm" METHOD="post">
</form>
</cfset>

<cfset thisPath = ExpandPath("*.")>
<cfset thisDirectory = GetDirectoryFromPath(thisPath)>
<cfoutput>
The current directory is: #GetDirectoryFromPath(thisPath)#
</cfoutput>
<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>
    </cfif>
  </cfif>
</cfif>
<cfelse>
<p>Your file was not found in this directory:
Here is a list of the other files in this directory:
</p>
<cfoutput>
<!-- use cfdirectory show directory, order by name & size -->
</cfoutput>
<cfoutput>
<!-- Output the contents of the cfdirectory as a CFTABLE -->
</cfoutput>
<cfoutput>
</cfoutput>
<cfelse>
<p>Please enter a file name</p>
</cfif>
```
<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, URLDecode, 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
<!--- 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 & "<br>");
WriteOutput("URL.YYY is " & URL.yyy & "<br>");
theEncoding = GetEncoding("URL");
WriteOutput("The URL variables were decoded using '" & theEncoding & "' encoding.");

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>
  <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

```cffile
<h3>GetFileFromPath Example</h3>
<cfset thisPath = ExpandPath("*.*")>
<cfset thisDirectory = GetDirectoryFromPath(thisPath)>
<cfoutput>
The current directory is: #GetDirectoryFromPath(thisPath)#
<cfif IsDefined("FORM.yourFile")>
<cfif FORM.yourFile is not ">"
<cfset yourFile = FORM.yourFile>
<cfif FileExists(ExpandPath(yourFile))>
<p>Your file exists in this directory. You entered the correct file name, #GetFileFromPath(#thisPath#/#yourfile#)#
<cfelse>
<p>Your file was not found in this directory:
<br>Here is a list of the other files in this directory:
<!--- use cfdirectory to give the contents of the snippets directory, order by name and size --->
<cfdirectory
directory = "/thisDirectory/"
name = "myDirectory"
sort = "name ASC, size DESC"
ificar undefined message="#Name#">
<!--- 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>
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()

Parameters

<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
The structure returned by this function contains the following entries:

Note: To determine whether data is binary, use IsBinary(x.content). To convert data to a string value, if it can be displayed as a string, use toString(x.content).

The following example shows how this function can return HTTP header information.

Example
```cfset x = GetHttpRequestData()```
```cfoutput```
```<table cellpaddling = "2" cellspacing = "2">```
```<tr>```
```<td><b>HTTP Request item</b></td>```
```<td><b>Value</b></td></tr>```
```<cfloop collection = #x.headers# item = "http_item">```
```<tr>```
```<td>#http_item#</td>```
```<td>#StructFind(x.headers, http_item)#</td></tr>```
```</cfloop>```
```<tr>```
```<td>request_method</td>```
```<td>#x.method#</td></tr>```
```<tr>```
```<td>server_protocol</td>```
```<td>#x.protocol#</td></tr>```
```</table>```
<b>http_content --- #x.content#</b>
</cfoutput>
GetHttpTimeString

**Description**
Gets the current time, in the Universal Time code (UTC).

**Returns**
The time, as a string, according to the HTTP standard described in RFC 1123.

**Category**
Date and time functions, International functions

**Function syntax**
GetHttpTimeString(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**

```cfoutput
  #GetHttpTimeString("#Now()#")#<br>
</cfoutput>```
GetK2ServerDocCount

Description
This function is deprecated.
Determines the number of documents that can be searched by the ColdFusion registered K2 Server. This function is used primarily by the ColdFusion Verity and K2Server Administrator pages, and requires significant processing time. Avoid using it in production applications. This function uses Verity K2Server Release K2.2.0.

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.

Gets the maximum number of documents that the ColdFusion registered K2 Server is permitted to return from a search. This function is used primarily by the ColdFusion Verity and K2Server Administrator pages. This function uses Verity K2Server Release K2.2.0.

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.

The restricted version of K2 Server version that is installed with ColdFusion has the following document search limits:

- For ColdFusion MX Evaluation: 250,000
- For ColdFusion MX Developer: 10,000
- For ColdFusion MX Professional: 125,000
- For ColdFusion MX Enterprise: 250,000
- For ColdFusion MX with Macromedia Spectra: 750,000

K2Broker with ColdFusion MX has no limit.

Example
<cfoutput>GetK2ServerDocCountLimit =
*$@GetK2ServerDocCountLimit()@$</cfoutput>
GetLocale

Description
Gets the current geographic/language locale value.

To set the default display format of date, time, number, and currency values in a ColdFusion application session, you use the SetLocale function.

Returns
The current locale value, as a string.

Category
Display and formatting functions, International functions, System functions

Function syntax
GetLocale()

See also
SetLocale

History
ColdFusion MX: Changed behavior: this function determines whether a locale value is set. (The value is set with the SetLocale function.)

- If the locale value is present, the function now returns it.
- If the 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 supported, the function returns this value
  - If the default locale value is not supported, the function returns English (US). (The code is "en_us"). (When ColdFusion is started, it stores the supported locale values in the variable Server.ColdFusion.SupportedLocales.) ColdFusion sets the locale in the JVM to this value; it persists until the server is restarted or it is reset with the SetLocale function.

Usage
This function does not access a web browser’s Accept-Language HTTP header setting.

Example
<h3>GetLocale Example</h3>
<p>The locale for this system is <cfoutput>#GetLocale()#</cfoutput>
GetMetaData

Description
Gets metadata (the methods, properties, and parameters of a component) associated with an object that is deployed on the ColdFusion server. This functionality, called introspection, lets applications dynamically determine how to use a component.

Returns
Key-value pairs, as a component descriptor data structure or as structured XML

Category
System functions

Function syntax
GetMetaData(object)

or, if used within a ColdFusion component:

GetMetaData(this)

History
ColdFusion MX: Added this function.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>Reference to an object; use this attribute to call a function from a CFML page</td>
</tr>
<tr>
<td>this</td>
<td>Reference to an object; use this attribute to call a function from a component.</td>
</tr>
</tbody>
</table>

Usage
The this scope is available at runtime to the component body and to the invoked method’s function body. It is used to read and write variables that are present during the life of the component.

Component metadata contains at least the following keys:

- name: the component name
- path: an absolute path to the component
- extends: ancestor component metadata
- functions: an array of metadata for each component function

Other component attributes are returned as additional keys.

Function metadata contains at least the following keys:

- name: the function name
- parameters: an array of argument metadata

Other function attributes are returned as additional keys.

Argument metadata contains at least the following key:

- name: the argument name

Other argument attributes are returned as additional keys.
Property metadata contains at least the following key:

- name: the property name

Other property attributes are returned as additional keys.
GetMetricData

Description
   Gets server performance metrics.

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>
</table>
| mode      | perf_monitor | Returns internal data, in a structure.  
To receive data, you must enable PerfMonitor in ColdFusion Administrator before executing the function.  
On Windows, this data is otherwise displayed in the Windows PerfMonitor. |
| simple_load |            | Returns an integer value that is computed from the state of the server's internal queues. Indicates the overall server load. |
| prev_req_time |        | Returns the time, in milliseconds, that it took the server to process the previous request. |
| avg_req_time |        | Returns the average time, in milliseconds, that it takes the server to process a request.  
Changing the setting to 0 prevents the server from calculating the average and removes overhead associated with gathering data.  
Default: 120 seconds. |

Usage
   If mode = "perf_monitor", the function returns a structure with these data fields:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>InstanceName</td>
<td>The name of the ColdFusion server. Default: 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>
| ReqRunning      | Number of HTTP requests currently running.  
In the ColdFusion Administrator, you can set the maximum number of requests that run concurrently. |
Example

```coldfusion
<!--- This example gets and displays metric data from Windows NT PerfMonitor --->
<cfset pmData = GetMetricData( "PERF_MONITOR" ) >
<cfoutput>
  Current PerfMonitor data is:<p>
  InstanceName:#pmData.InstanceName#<p>
  PageHits:#pmData.PageHits#<p>
  ReqQueued: #pmData.ReqQueued#<p>
  DBHits: #pmData.DBHits#<p>
  ReqRunning: #pmData.ReqRunning#<p>
  BytesIn: #pmData.BytesIn#<p>
  BytesOut: #pmData.BytesOut#<p>
  AvgQueueTime: #pmData.AvgQueueTime#<p>
  AvgReqTime: #pmData.AvgReqTime#<p>
  AvgDBTime: #pmData.AvgDBTime#<p>
</cfoutput>
```

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ReqTimedOut</td>
<td>Number of HTTP requests that timed out while in the staging queue or during processing.</td>
</tr>
<tr>
<td>BytesIn</td>
<td>Number of bytes in HTTP requests to ColdFusion MX</td>
</tr>
<tr>
<td>BytesOut</td>
<td>Number of bytes in HTTP responses from ColdFusion MX</td>
</tr>
<tr>
<td>AvgQueueTime</td>
<td>For the last two HTTP requests (current and previous), the average length of time the request waited in the staging queue.</td>
</tr>
<tr>
<td>AvgReqTime</td>
<td>For the last two HTTP requests (current and previous), the average length of time the server required to process the request</td>
</tr>
<tr>
<td>AvgDBTime</td>
<td>For the last two HTTP requests (current and previous), the average length of time the server took to process CFQueries in the request.</td>
</tr>
<tr>
<td>cachepops</td>
<td>This parameter is deprecated. ColdFusion automatically sets its value to -1.</td>
</tr>
</tbody>
</table>

Field Description

- **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.
- **cachepops**: 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");
```

For more information, see your Java Server Pages (JSP) documentation.

**Example**

```cfs
cfset pc = GetPageContext()
<cfset pc.setAttribute("name","John Doe")>
<cfoutput>
name: #variables.name#<br>
Language of the current locale is #pc.getRequest().getLocale().getDisplayLanguage()#</cfoutput>.
```

```
<cfset pc.setAttribute("name", "John Doe")>
<cfoutput>
name: #variables.name#<br>
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**

\[ \text{GetProfileString}(\text{iniPath, section, entry}) \]

**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")>
  <h4>Boot Loader</h4>
  <!--- If no entry in an initialization file, nothing displays --->
  <p>Timeout is set to: <cfoutput>#timeout#</cfoutput>.</p>
</cfif>

<form action = "getprofilestring.cfm">
  <table cellspacing = "2" cellpadding = "2" border = "0">
    <tr>
      <td>Full Path of Init File</td>
      <td><input type = "Text" name = "IniPath" value = "C:\myboot.ini"> </td>
    </tr>
  </table>
</form>
</tr></table>
</form>
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
<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>

GetTempDirectory

505


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 directory dir</td>
</tr>
</tbody>
</table>

Example

<h3>GetTempFile Example</h3>
<p>The temporary directory for this ColdFusion Server is <cfoutput>#GetTempDirectory()#</cfoutput>.</p>
<p>We have created a temporary file called: <cfoutput>#GetTempFile(GetTempDirectory(),"testFile")#</cfoutput>
GetTemplatePath

Description
This function is deprecated. Use the GetBaseTemplatePath function instead.
Gets the absolute path of an application's base page.

History
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

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
<!--- Setup timing test --->
<cfset iterationCount = 1000>
<!--- Time an empty loop with this many iterations --->
<cfset tickBegin = GetTickCount()>
<cfloop Index = i From = 1 To = #iterationCount#></cfloop>
<cfset tickEnd = GetTickCount()>
<cfset loopTime = tickEnd - tickBegin>

<!--- Report --->
<cfoutput>Loop time (#iterationCount# iterations) was: #loopTime# milliseconds</cfoutput>
GetTimeZoneInfo

Description

Gets local time zone information for the computer on which it is called, relative to Universal Time Coordinated (UTC). UTC is the mean solar time of the meridian of Greenwich, England, used as the basis for calculating standard time throughout the world.

ColdFusion stores date and time values as date-time objects: real numbers on a universal time line. It converts an object to a time zone when it formats an object; it converts a date/time value from a time zone to a real number when it parses a value.

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>
Total offset in seconds is #info.utcTotalOffset#. 
Offset in hours is #info.utcHourOffset#. 
Offset in minutes minus the offset in hours is #info.utcMinuteOffset#. 
Is Daylight Savings Time in effect? #info.isDSTOn#. 
</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,ten:, eleven:,twelve:,thirteen," & chr(32) & chr(9) & chr(10) & ",four">  
<cfoutput> 
#mystring#<br><br><br>
</cfoutput>
```


The output is as follows:

four,
.five. nine. zero::
nine. ten:: eleven:: twelve:: thirteen,
.four

The GetToken function recognizes explicit spaces, tabs, or newline characters as the parameter delimiters (To specify a space character, the code is chr(32); a tab character, chr(9); and a newline character, chr(10).

In the example string mystring, there is:
- A forced space between the substrings "four," and ".five"
- A literal space between "five," and "nine"
- A literal space between "ten:," and "eleven:"
- A forced space between "thirteen," and ",four"

In the following call against mystring, no spaces are specified in delimiters (it is omitted), so the function uses the space character as the string delimiter:

```cfoutput>
GetToken(mystring, 3) is : #GetToken(mystring, 3)#
</cfoutput><br>
```

The output of this code is as follows:

GetToken(mystring, 3) is : nine.zero::

The function finds the third delimiter, and returns the substring just before it that is between the second and third delimiter. This substring is "nine.zero::".

**Example C:** Consider the following code:

```cfset mystring2 = "four,"
   & #chr(9)# & #chr(10)# & ".five.nine.zero::"
   & #chr(10)# & "nine.ten:: eleven:: twelve:: thirteen."
   & #chr(9)# & #chr(10)# & ",four"
```

```cfoutput>
mystring2@<br>
</cfoutput>
```

The output is as follows:

four,
.five.nine.zero::
nine.ten:: eleven:: twelve:: thirteen,
.four

The following is a call against mystring2:

```cfoutput>
GetToken(mystring2, 2) is : #GetToken(mystring2, 2)#
</cfoutput>
```

The output is as follows:

GetToken(mystring2, 2) is : .five.nine.zero::
The function finds the second delimiter, and returns the substring just before it that is between the first and second delimiter. This substring is ",five,nine,zero;:"

Example

```coldfusion
<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" is: #GetToken(FORM.yourString, FORM.returnElement, yourDelimiter)#
    </cfoutput>
  </cfif>
</cfif>
```

...
Hash

Description
Converts a variable-length string to a 32-byte, hexadecimal string, using the MD5 algorithm. (It is not possible to convert the hash result back to the source string.)

Returns
32-byte, hexadecimal string

Category
Conversion functions, Other functions, String functions

Function syntax
Hash(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
The result is useful for comparison and validation. For example, a developer can store the hash of a password in a database without exposing the password. The developer can check the validity of the password with the following code:

```<cfif hash(form.password) is not myQuery.passwordHash>   <cflocation url = "unauthenticated.cfm"></cfif>```

Example

```<h3>Hash Example</h3>
<cfquery name = "CheckPerson" datasource = "UserData">
  SELECT PasswordHash
  FROM SecureData
  WHERE UserID = <cfqueryparam value = "#UserID#" cfsqltype = "CF_SQL_CHARVAR">
</cfquery>
<cfif Hash(form.password) is not checkperson.passwordHash>
  <cflocation url = "unauthenticated.cfm">
<cfelse>
...
</cfif>```
Hour

Description
Gets the current hour of the day.

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
<!--- 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. Returns are removed from *string*. Special characters (`>`, `<`, `*`, `&`) are escaped.

**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 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**
```html
<!--- 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\nPrevious line was blank!!!">

<cfoutput>
<h3>The text without processing</h3>
#testString#\n
<h3>Using HTMLCodeFormat</h3>
#HTMLCodeFormat(testString)#

<h3>Using HTMLEditFormat</h3>
#HTMLEditFormat(testString)#
</cfoutput>
```

---

**HTMLCodeFormat** 515
**HTMLEditFormat**

**Description**
Replaces special characters in a string with their HTML-escaped equivalents.

**Returns**
HTML-escaped string `string`. Returns are removed from `string`. Special characters (for example, `<` `&`) are escaped.

**Category**
Display and formatting functions

**Function syntax**
`HTMLEditFormat(string [, version ])`

**See also**
`HTMLCodeFormat`

**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>• -1: The latest implementation of HTML</td>
</tr>
<tr>
<td></td>
<td>• 2.0: HTML 2.0 (Default)</td>
</tr>
<tr>
<td></td>
<td>• 3.2: HTML 3.2</td>
</tr>
</tbody>
</table>

**Usage**
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
<!--- 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">
<cfoutput>
<h3>The text without processing</h3>
#testString#<br>
<h3>Using HTMLCodeFormat</h3>
#HTMLCodeFormat(testString)#
<h3>Using HTMLEditFormat</h3>
#HTMLEditFormat(testString)#
</cfoutput>
```

Previous line was blank!!!
**IIf**

**Description**
Evaluates a Boolean conditional dynamic expression. Depending on whether the expression is true or false, dynamically evaluates one of two string expressions and returns the result. This function is convenient for incorporating a *cfif* tag in-line in HTML.

For general conditional processing, see *cfif*. For error handling, see *cftry*. For more information, see *Developing ColdFusion MX Applications*.

**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**
```coldfusion
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:

```coldfusion
<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:

```coldfusion
IIf(y is 0, DE("Error"), x/y)
```

If `y = 0`, this generates an error, because the third expression is the value of `x/0` (invalid expression).

ColdFusion evaluates `string_expression1` and `string_expression2`. To return the string itself, use the `DE` function.
**Note:** If you use pound signs (#) in string_expression1 or string_expression2, ColdFusion evaluates the part of the expression in pound signs first. If you misuse the pound signs, you can cause unexpected results from the IIf function. For example, if you use pound signs around the whole expression in string_expression1, and if there is an undefined variable in string_expression1, the function might fail, with the error 'Error Resolving Parameter.'

If a variable is undefined, ColdFusion throws an error when it processes this function. The following example shows this problem:

```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 pound 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 pound signs around `LocalVar` in the following code cause it to fail with the error message 'Error Resolving Parameter', because ColdFusion never evaluates the original condition `IsDefined("LocalVar")`.

Here is another example:

```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 pound 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 (True: run expression 1; False: run expression 2).</p>
<p>The result of the expression</p>
<p>
#IIf(Hour(Now()) GTE 12, DE("It is afternoon or evening"), DE("It is morning"))
```

The output is:

```
It is afternoon or evening
```

```coldfusion
<cfoutput>
  #IIf(Hour(Now()) GTE 12, DE("It is afternoon or evening"), DE("It is morning"))#
</cfoutput>
```
</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
InputBaseN(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 radix.</td>
</tr>
<tr>
<td>radix</td>
<td>Base of the number represented by string, in the range 2 — 36.</td>
</tr>
</tbody>
</table>

Example
<h3>InputBaseN Example</h3>

<p>FormatBaseN converts a number to a string in the base specified by Radix.</p>
<p>
<cfoutput>
<br>FormatBaseN(10,2): #FormatBaseN(10,2)#
<br>FormatBaseN(1024,16): #FormatBaseN(1024,16)#
<br>FormatBaseN(125,10): #FormatBaseN(125,10)#
<br>FormatBaseN(10.75,2): #FormatBaseN(10.75,2)#
</cfoutput>

<h3>InputBaseN Example</h3>
<p>InputBaseN returns the number obtained by converting a string, using the base specified by Radix.</p>
<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>
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

<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 to insert.</td>
</tr>
<tr>
<td>string</td>
<td>A string or a variable that contains one. String into which to insert substring.</td>
</tr>
<tr>
<td>position</td>
<td>Integer or variable; position in string after which to insert substring.</td>
</tr>
</tbody>
</table>

Example

```
<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>
</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>
```
Int

Description
Calculates the closest integer that is smaller than number.

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

<h3>Int Example</h3>
<p>Int returns the closest integer smaller than a number.</p>
<p>Int(11.7) : <cfoutput>#Int(11.7)#</cfoutput></p>
<p>Int(-11.7) : <cfoutput>#Int(-11.7)#</cfoutput></p>
<p>Int(0) : <cfoutput>#Int(0)#</cfoutput></p>
**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**

```plaintext
IsArray(value [, number ])
```

**See also**
Array functions

**History**
ColdFusion MX:
- Changed behavior: if the `value` attribute 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**
This function evaluates a Java array object, such as vector object, as having one dimension.

**Example**

```cfml
<h3>IsArray Example</h3>

<!--- Make an array --->
<cfset MyNewArray = ArrayNew(1)>
<!--- set some elements --->
<cfset MyNewArray[1] = "element one">
<cfset MyNewArray[2] = "element two">
<cfset MyNewArray[3] = "element three">
<!--- is it an array? --->
<cfoutput>
<p>Is this an array? #IsArray(MyNewArray)#
<p>It has #ArrayLen(MyNewArray)# elements.
<p>Contents: #ArrayToList(MyNewArray)#
</cfoutput>
```
IsAuthenticated

Description
This function is obsolete. Use the newer security tools; see “Authentication functions” on page 367 and Chapter 16, “Securing Applications,” in Developing ColdFusion MX Applications.

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 “Authentication functions” on page 367 and Chapter 16, “Securing Applications,” in Developing ColdFusion MX Applications.

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>#htmleditformat(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.<br></cfif>
<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, 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>
  <cfelse>
    <h3>The expression <cfoutput>#DE(FORM.theTestValue)#</cfoutput> is not Boolean</h3>
  </cfif>
</cfif>

<form action = "isBoolean.cfm">
  <p>Enter an expression, and discover whether it can be evaluated to a Boolean value.

  <input type = "Text" name = "TheTestValue" value = "1">
  <input type = "Submit" value = "Is it Boolean?" name = ">
</form>
IsCustomFunction

Description
Determines whether a 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 quotes. 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>
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; otherwise, False. 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
CreateDateTime, IsNumericDate, 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

```
<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()#" />
</p>
<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; otherwise, No.

**Example**
```html
<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.
Returns False for an array or structure element referenced using bracket notation. For example, IsDefined("myArray[3]") always returns False, even if the array element myArray[3] has a value.

Category
Decision functions

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:

```cfif IsDefined("form.myVariable")
if(structKeyExists(form,"myVariable"))
```

Example

```<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>
</cfif>
<cfelse>
<p>During the first time through this template, the variable "form.myString" has not yet been defined, so we do not try to evaluate it.</p>
</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.

Determines whether the K2Server version is K2 Broker. For more information, see `GetK2ServerDocCountLimit` on page 495. This function is used primarily by the ColdFusion Verity and K2Server Administrator pages. This function uses Verity K2Server Release K2.2.0.

**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**
```coldfusion
<cfoutput>IsK2ServerABroker = #IsK2ServerABroker()#$</cfoutput>
```
IsK2ServerDocCountExceeded

**Description**
This function is deprecated.
Determines whether the number of documents that can be searched by the ColdFusion registered K2 Server exceed the limit. Depends on the K2Server platform limit; see GetK2ServerDocCountLimit on page 495.
This function is used primarily by the ColdFusion Verity and K2Server Administrator pages. This function uses Verity K2Server Release K2.2.0.

**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.

Determines whether the K2Server is running and available to perform a search. This function is used primarily by the ColdFusion Verity and K2Server Administrator pages. This function uses Verity K2Server Release K2.2.0.

**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
Determined whether a year is a leap year.

Returns
True, if year is a leap year; otherwise, False.

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)>
  </cfif>
  <h3>The year value #DE(FORM.theTestValue)# is a Leap Year</h3>
  <cfelse>
   <h3>The year value #DE(FORM.theTestValue)# is not a Leap Year</h3>
  </cfif>
</cfif>

<form action="isLeapYear.cfm">
<p>Enter a year value, and find out whether it is a Leap Year.</p>
<input type = "Text" name = "TheTestValue" value = 
    #Year(Now())#</input>
<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; otherwise, False.

**Category**

Decision functions

**Function syntax**

```
IsNumeric(string)
```

**See also**

IsBinary

**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 `<cfoutput>#DE(FORM.theTestValue)#</cfoutput>` can be converted to a number</h3>
  </cfif>
  <cfelse>
    <h3>The string `<cfoutput>#DE(FORM.theTestValue)#</cfoutput>` cannot be converted to a number</h3>
  </cfelse>
</cfif>

<form action = "isNumeric.cfm">
  <p>Enter a string, and find out whether it can be evaluated to a numeric value.</p>
  <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; otherwise, False.

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>
</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>
<input type="Text" name="TheTestValue" value="<CFOUTPUT>#Now()#</CFOUTPUT>">
<input type="Submit" value="Is it a Date?" name="">
</form>
**IsObject**

**Description**
Determines whether a value is an object.

**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**
```cfml
<!--- 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">
  <cffile IsObject(getColor)>
    <!-- Invoke the myFunction method --->
    <cfinvoke
      component="#getColor#" method="myFunction"
      returnVariable="myColor">
    </cfinvoke>
  </cffile>
  <cfif IsDefined("myColor")>
    <!-- Output the returned variable --->
    The value of myColor = <cfoutput>#myColor#</cfoutput><p>
  </cfif>
</cfobject>
```

IsObject 541
IsProtected

Description
This function is obsolete. Use the newer security tools; see “Authentication functions” on page 367 and Chapter 16, “Securing Applications,” in Developing ColdFusion MX Applications

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</p>
</cfif>
<!--- make a query on the snippets datasource --->
<cfquery name = "GetEmployees" datasource = "cfsnippets">
SELECT *
FROM employees
</cfquery>

<p>After the query is run, GetEmployees contains a number of rows
that look like this (display limited to three rows):
<cfoutput QUERY = "GetEmployees" MaxRows = "3">
<pre>#Emp_ID# #FirstName# #LastName#</pre>
</cfoutput>
<cfif IsQuery(GetEmployees)>
GetEmployees is no longer a simple value, but the name of a query
</cfif>
```
IsSimpleValue

Description
Determines the type of a value.

Returns
True, if value is a string, number, Boolean, or date/time value; False, otherwise.

Category
Decision functions

Function syntax
IsSimpleValue(value)

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
<!--- Shows an example of IsQuery and IsSimpleValue --->
<h3>IsSimpleValue Example</h3>
<!--- define a variable called "GetEmployees" --->
<cfparam name = "GetEmployees" default = "#Now()#">

<p>Before the query is run, the value of GetEmployees is
<cfoutput>#GetEmployees#</cfoutput>
<cfif IsSimpleValue(GetEmployees)>
<p>GetEmployees is currently a simple value</p>
<cfif>
<!--- make a query on the snippets datasource --->
<cfquery name = "GetEmployees" datasource = "cfsnippets">
SELECT *
FROM employees
</cfquery>

<p>After the query is run, GetEmployees contains a number of rows
that look like this (display limited to three rows):
<cfoutput QUERY = "GetEmployees" MaxRows = "3">
<pre>#Emp_ID# #FirstName# #LastName#
</pre>
</cfoutput>
<cfif IsQuery(GetEmployees)>
GetEmployees is no longer a simple value, but the name of a query
</cfif>
IsStruct

Description
Determines whether a variable is a structure.

Returns
True, if variable is a ColdFusion structure or is a Java object that implements the java.lang.Map interface. Returns False if the object in variable is a user-defined function (UDF).

Category
Decision functions, Structure functions

Function syntax
IsStruct( variable )

See also
Structure functions

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. --->
<p>This file is similar to addemployee.cfm, which is called by StructNew, StructClear, and StructDelete. It is an example of a custom tag used to add employees. Employee information is passed through the employee structure (the EMPINFO attribute). In UNIX, you must also add the Emp_ID.
<!---
<cfswitch expression = "#ThisTag.ExecutionMode#">
  <cfcase value = "start">
    <cfif IsStruct(attributes.EMPINFO)>
      <cfoutput>Error. Invalid data.</cfoutput>
      <cfexit method = "ExitTag">
    <cfelse>
      <cfquery name = "AddEmployee" datasource = "cfsnippets">
        INSERT INTO Employees (FirstName, LastName, Email, Phone, Department) VALUES
        <cfoutput>
          ("#StructFind(attributes.EMPINFO, "firstname")#", "#StructFind(attributes.EMPINFO, "lastname")#", "#StructFind(attributes.EMPINFO, "email")#", "#StructFind(attributes.EMPINFO, "phone")#", "#StructFind(attributes.EMPINFO, "department")#"
        </cfoutput>
      </cfquery>
      <cfoutput><hr>Employee Add Complete</cfoutput>
    <cfelse>
      <cfquery name = "AddEmployee" datasource = "cfsnippets">
        INSERT INTO Employees (FirstName, LastName, Email, Phone, Department) VALUES
        <cfoutput>
          ("#StructFind(attributes.EMPINFO, "firstname")#", "#StructFind(attributes.EMPINFO, "lastname")#", "#StructFind(attributes.EMPINFO, "email")#", "#StructFind(attributes.EMPINFO, "phone")#", "#StructFind(attributes.EMPINFO, "department")#"
        </cfoutput>
      </cfquery>
      <cfoutput><hr>Employee Add Complete</cfoutput>
    </cfelse>
  </cfif>
</cfswitch>
</cfcase>
</cfswitch> --->

IsStruct 547
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
Authentication functions, Decision functions

Function syntax
IsUserInRole("role_name")

See also
GetAuthUser, cfflogin, cffloginuser, Chapter 16, “Securing Applications,” in Developing ColdFusion MX Applications

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
<cffif IsUserInRole("Admin")>
    <cffoutput>Authenticated user is an administrator</cffoutput>
<cffelse IsUserInRole("User")>
    <cffoutput>Authenticated user is a user</cffoutput>
</cffif>
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)

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
<cfset packet="
<wddxPacket version='1.0'>
<header></header>
data>
<struct>
<var name='ARRAY'>
<array length='3'>
<string>one</string>
<string>two</string>
</array>
</var>
<var name='NUMBER'>
<string>5</string>
</var>
<var name='STRING'>
<string>hello</string>
</var>
</struct>
</data>
</wddxPacket>"
</hr>
<xmp>
<cfoutput>#packet#</cfoutput>
</xmp>
IsWDDX() returns #IsWDDX(packet)#<br>
</cfoutput>
IsXmlDoc

Description
Determines whether a function parameter is an Extended Markup language (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
IsXmlElem, IsXmlRoot, XmlChildPos, XmlNew, XmlParse, XmlSearch, XmlTransform

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>
**IsXmlElem**

**Description**
Determines whether a function parameter is an Extended Markup language (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**
```coldfusion
IsXmlElem(value)
```

**See also**
IsXmlDoc, IsXmlRoot, XmlChildPos, XmlNew, XmlParse, XmlSearch, XmlTransform

**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>
**IsXmlRoot**

**Description**
Determines whether a function parameter is the root element of an Extended Markup language (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**
IsXmlDoc, IsXmlElem, XmlChildPos, XmlNew, XmlParse, XmlSearch, XmlTransform

**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>
**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>
<tbody>
<tr>
<td>type</td>
<td>Data type to which to convert variable:</td>
</tr>
<tr>
<td></td>
<td>• boolean</td>
</tr>
<tr>
<td></td>
<td>• int</td>
</tr>
<tr>
<td></td>
<td>• long</td>
</tr>
<tr>
<td></td>
<td>• float</td>
</tr>
<tr>
<td></td>
<td>• double</td>
</tr>
<tr>
<td></td>
<td>• string</td>
</tr>
<tr>
<td>variable</td>
<td>A ColdFusion variable that holds a scalar or string type</td>
</tr>
</tbody>
</table>

**Usage**

Use after creating a Java object with the `cfobject` tag, before calling one of its methods. If the method takes more than one overloaded argument, you must call `JavaCast` for each one. Use `JavaCast` only when a method is overloaded (because its arguments can take more than one data type, not because the method can take a variable number of arguments).

`JavaCast` cannot be used to cast between complex objects, nor to cast to a super-class. Use this function's result only on calls to Java objects. Because there is not a one-to-one correspondence between internally stored ColdFusion types and Java scalar types, some conversions cannot be performed.

**Example**

The 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>
</cfobject>

<!--- ColdFusion can treat this as a string or a real number --->
<cfset x = 33>

Perform an explicit cast to an int and call fooMethod:
<cfset myInt = JavaCast("int", x)>
<cfoutput>#obj.fooMethod(myInt)#</cfoutput>

Perform an explicit cast to a string and call fooMethod:
<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

**Parameter** | **Description**
--- | ---
string | A string or a variable that contains one.

Usage

Escapes special JavaScript characters, so you can put arbitrary strings safely into JavaScript.

Example

```coldfusion
<!--- 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

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>LCase Example</h3>

```cfdm
<cfif IsDefined("FORM.sampleText")>
  <cfif FORM.sampleText is not ">
    <p>Your text. \#FORM.sampleText\# returned in lowercase is \#LCase(FORM.sampleText)\#</p>
  </cfif>
  <cfelse>
    <p>Please enter some text.</p>
  </cfif>
</cfif>

<form action="lcase.cfm">
  <p>Enter your text. Press "submit" to see it returned in lowercase:</p>
  <p><input type="Text" name="SampleText" value="SAMPLE"></p>
  <input type="Submit" name="" value="submit"></form>
```

LCase 557
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
<h3>Left Example</h3>

```cf
<!--- if len is 0, then err --->
<cfif IsDefined("Form.MyText")>
<cfif Len(form.myText) is not 0>
<p>Your string: #form.myText#</p>
<p>Your changed string, showing only the leftmost characters: #Left(Form.myText, form.removeChars)#</p>
<cfif Len(form.myText) LTE form.RemoveChars>
<p>Your original string: #form.myText#</p>
<p>Your string only has #Len(form.myText)# characters. You cannot output the #form.removeChars# leftmost characters of this string because it is not long enough</p>
<cfelse>
<p>Please enter a string</p>
</cfif>
</cfif>
<form action="#CGI.ScriptName#" method="POST">
<p>Type in some text</p>
<p>How many characters from the left do you want to show?</p>
<select name="RemoveChars">
<option value="1">1</option>
<option value="3" selected>3</option>
<option value="5">5</option>
<option value="7">7</option>
</select>
</form>
```
<option value="9">9</option>
<input type="Submit" name="Remove characters"/>
</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
<h3>Len Example</h3>

<cfif IsDefined("Form.MyText")>
  <!--- if len is 0, then err --->
  <cfif Len(FORM.myText) is not 0>
    <p>Your string, <cfoutput>"#FORM.myText#"</cfoutput>, has <cfoutput>#{Len(FORM.myText)#} characters.</cfoutput></p>
  <cfelse>
    <p>Please enter a string of more than 0 characters.</p>
  </cfif>
</cfif>

<form action = "len.cfm">
  <p>Type in some text to see the length of your string.</p>
  <br><input type = "Text" name = "MyText">
  <br><input type = "Submit" name = "Remove characters">
</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 [, delimiter])
```

**See also**
ListPrepend, ListInsertAt, ListGetAt, ListLast, ListSetAt

**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. Default: 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`.

To add an element to the beginning or end of a list, Macromedia recommends that you do so with code such as the following, rather than with the `ListAppend` or `ListPrepend` functions:

```coldfusion
cfset MyValue = "another element"
cfif listLen(myList) is 0>
   cfset myList = MyValue
<cfelse>
   cfset myList = myList & ", " & MyValue
</cfif>
```

The following table shows examples of `ListAppend` processing:

<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(&quot;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**
```
<!--- First, query to get some values for our list elements—–>
```
<cfquery name = "GetParkInfo" datasource = "cfsnippets">
SELECT PARKNAME.CITY, STATE
FROM PARKS WHERE PARKNAME LIKE 'AL%'
</cfquery>
<cfset temp = ValueList(GetParkInfo.ParkName)>
<cfoutput>
<p>The original list: #temp#</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

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. Default: 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.
</p><cfset temp = ValueList(GetParkInfo.ParkName)>
<cfoutput>
The original list: #temp#
</cfoutput>
<cfset temp2 = ListChangeDelims(Temp, "|:P|", ",")>
<cfoutput>
After executing the statement
<br><strong>ListChangeDelims(Temp, "|:P|", ",")</strong>,
the updated list: #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
ListContains(list, substring [ , delimiters ])

See also
ListContainsNoCase, ListFind

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. Default: 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
<!--- 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: #aList#</p> 
<p><strong>ListContains</strong> checks for substring "wo" in the list elements: 
<cfoutput> 
<p>Substring "wo" is in #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;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbs

564 Chapter 3: ColdFusion Functions
<cfoutput>
The string "two" is in <b>element #ListFind(aList, "two")#</b> of the list.
</cfoutput>
ListContainsNoCase

Description
Determines the index of the first list element that contains a specified substring.

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
ListContains, ListFindNoCase

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. Default: 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="cfsnippets">
   SELECT PARKNAME,CITY,STATE
   FROM Parks
   WHERE PARKNAME LIKE '#form.letter#'
</cfquery>
<cfset tempList = #ValueList(GetParkInfo.City)#>
<cfif ListContainsNoCase(tempList, form.yourCity) is not 0>
   There are parks in your city!
<cfelse>
   Sorry, there were no parks found for your city.
   Try searching under a different letter.
</cfif>
</cfif>
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
ListGetAt, ListSetAt, ListLen

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. Default: 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:
<cfset temp2 = ListDeleteAt(temp, "3")>

To specify another delimiter, you code as follows:
<cfset temp2 = ListDeleteAt(temp, "3", ";")>

ColdFusion ignores empty list elements; thus, the list "a,b,c,,,d" has four elements.

Example
<!--- First, query to get some values for our list elements--->
<CFQUERY NAME="GetParkInfo" DATASOURCE="cfsnippets">
    SELECT PARKNAME,CITY,STATE
    FROM Parks
    WHERE PARKNAME LIKE 'CHI%'
</CFQUERY>
<CFSET temp = ValueList(GetParkInfo.ParkName)>
<CFSET deleted_element = ListGetAt(temp, "3", ",")>
<CFOUTPUT><p>The original list: #temp#</p></CFOUTPUT>
<!--- Delete the third element from the list --->
<CFSET temp2 = ListDeleteAt(Temp, "3")>
<CFOUTPUT>
The changed list: #temp2#

This list element: #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
ListContains, ListFindNoCase

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 element.</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. Default: 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">  
<p>Pick a search type:  
<select name="type">  
<option value="ListFind" selected>Case-Sensitive  
<option value="ListFindNoCase">Case-Insensitive  
</select>  
<input type="Submit" name="" value="Search Employee List">  
</form>  
<!--- wait to have a string for searching defined --->
<cfif IsDefined("form.myString") and IsDefined("form.type")>
<cfquery name="SearchEmpLastName" datasource="cfsnippets">  
SELECT  FirstName, RTrim(LastName) AS LName, Phone, Department  
FROM    Employees  
</cfquery>  
<cfset myList = ValueList(SearchEmpLastName.LName)>
<cfif form.type is "ListFind">
<cfset temp = ListFind(myList, form.myString)>
<cfif temp is 0>
<h3>An employee with that exact last name was not found</h3>
<cfelse>
<cfoutput>
<p>Employee #ListGetAt(ValueList(SearchEmpLastName.FirstName), temp)#
#ListGetAt(ValueList(SearchEmpLastName.LName), temp)#, of the
#ListGetAt(ValueList(SearchEmpLastName.Department), temp)# Department,
can be reached at #ListGetAt(ValueList(SearchEmpLastName.Phone), temp)#.
<p>This was the first employee found under this case-sensitive last name search.
</cfoutput>
</cfelse>
<cfelse>
<cfset temp = ListFindNoCase(myList, form.myString)>
<cfif temp is 0>
<h3>An employee with that exact last name was not found</h3>
<cfelse>
<cfoutput>
<p>Employee #ListGetAt(ValueList(SearchEmpLastName.FirstName), temp)#
#ListGetAt(ValueList(SearchEmpLastName.LName), temp)#, of the
#ListGetAt(ValueList(SearchEmpLastName.Department), temp)# Department,
can be reached at #ListGetAt(ValueList(SearchEmpLastName.Phone), temp)#.
<p>This was the first employee found under this case-insensitive last name search.
</cfoutput>
</cfelse>
</cfelse>
</cfif>
**ListFindNoCase**

**Description**
Determine 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**
ListContains, ListFind

**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. Default: 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**
```html
<!--- Uses ListFind and ListFindNoCase to see if a substring exists in a list -->
<form action="./listfind.cfm" method="POST">
  <p>Try changing the case in Leary's last name:<br><input type="Text" size="25" name="myString" value="Leary"></p>
  <p>Pick a search type:<br>    <select name="type">
      <option value="ListFind" selected>Case-Sensitive
      <option value="ListFindNoCase">Case-Insensitive
    </select>
  </p>
  <input type="Submit" name="" value="Search Employee List">
</form>

<!--- wait to have a string for searching defined --->
<cfif IsDefined("form.myString") and IsDefined("form.type")>
  <cfquery name="SearchEmpLastName" datasource="cfsnippets">
    SELECT FirstName, RTrim(LastName) AS LName, Phone, Department
    FROM Employees
  </cfquery>
  <cfset myList = ValueList(SearchEmpLastName.LName)>
  <!--- Is this case-sensitive or case-insensitive searching --->
```
<cfif form.type is "ListFind">
  <cfset temp = ListFind(myList, form.myString)>
  <h3>An employee with that exact last name was not found</h3>
</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>
  <cfoutput>
    Employee #ListGetAt(ValueList(SearchEmpLastName.FirstName), temp)#
    #ListGetAt(ValueList(SearchEmpLastName.LName), temp)#, of the
    #ListGetAt(ValueList(SearchEmpLastName.Department), temp)# Department,
    can be reached at #ListGetAt(ValueList(SearchEmpLastName.Phone), temp)#.
    <p>This was the first employee found under this case-sensitive last name
    search.</p>
  </cfoutput>
</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

**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. Default: 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**
```
<!-- ListFirst Example -->
<h3>ListFirst Example</h3>
<!-- Find a list of users who wrote messages -->
<cfquery name = "GetMessageUser" datasource = "cfsnippets">
  SELECT Username, Subject, Posted
  FROM Messages
</cfquery>
<cfset temp = ValueList(GetMessageUser.Username)>
<p>Before editing the list, it is:
<cfoutput>#ValueList(GetMessageUser.Username)#</cfoutput>. (Users who posted more than once are listed more than once.)
<p>The first user in the list is:
<cfoutput>#ListFirst(temp)#</cfoutput>
<p>The rest of the list is:
<cfoutput>#ListRest(temp)#</cfoutput>. (Users who posted more than once are listed more than once.)
<p>The last user in the list is:
<cfoutput>#ListLast(temp)#</cfoutput>
```
**ListGetAt**

**Description**
Gets a list element at a specified position.

**Returns**
Index of the list element at position `position`.

**Category**
List functions

**Function syntax**
```
ListGetAt(list, position [., delimiters])
```

**See also**
ListFirst, ListLast, ListQualify, ListSetAt

**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. Default: 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:

```cf
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:

```cf
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**

```cf
<!--- Find a list of users who wrote messages --->
<cfquery name = "GetMessageUser" datasource = "cfsnippets">
    SELECT Username, Subject, Posted
    FROM Messages
</cfquery>
<cfset temp = ValueList(GetMessageUser.Username)>

<!--- loop through the list and show it with ListGetAt --->
<h3>This list of usernames who have posted messages numbers #ListLen(temp)# users.</h3>
<ul>
    <cfoutput>#ListGetAt(temp, #LoopIndex#)#</cfoutput>
</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**
```
ListInsertAt(list, position, value [, delimiters ])
```

**See also**
ListDeleteAt, ListAppend, ListPrepend, ListSetAt

**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. Default: 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
<!--- This example shows ListInsertAt --->
<cfquery name = "GetParkInfo" datasource = "cfsnippets"> 
SELECT PARKNAME,CITY,STATE 
FROM PARKS 
WHERE PARKNAME LIKE 'DE%' 
</cfquery>  
<cfset temp = ValueList(GetParkInfo.ParkName)>  
<cfset insert_at_this_element = ListGetAt(temp, "3", ",")>  
<cfoutput>  
<p>The original list: #temp#</p>  
</cfoutput>  
<cfset temp2 = ListInsertAt(Temp, "3", "my Inserted Value")>  
```

576   Chapter 3: ColdFusion Functions
**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**
ListGetAt, ListFirst

**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. Default: comma.</td>
</tr>
</tbody>
</table>

If this parameter contains more than one character, ColdFusion processes each occurrence of each character as a delimiter; you cannot specify a multicharacter delimiter.

**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**

```cfml
<h3>ListFirst, ListLast, and ListRest Example</h3>
<cfquery name = "GetMessageUser" datasource = "cfsnippets">
  SELECT Username, Subject, Posted 
  FROM Messages
</cfquery>
<cfset temp = ValueList(GetMessageUser.Username)>
<p>Before editing the list, it is:<b>&nbsp;</b>
<cfoutput>@ValueList(GetMessageUser.Username)@</cfoutput>.  
<p>(Users who posted more than once are listed more than once.)
<cfset myList = "one hundred, two hundred, three hundred"
<cfset MyValue = #trim(ListLast(myList)#
<p>The first user in the list is: <cfoutput>@ListFirst(temp)@</cfoutput>
<p>The rest of the list is:<b>&nbsp;</b><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: <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**
ListAppend, ListDeleteAt, ListInsertAt, ListPrepend

**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. Default: 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**

```html
<h3>ListLen Example</h3>
<!--- Find a list of users who wrote messages --->
<cfquery name = "GetMessageUser" datasource = "cfsnippets">
  SELECT Username, Subject, Posted
  FROM  Messages
</cfquery>
<cfset temp = ValueList(GetMessageUser.Username)>
<!--- loop through the list and show it with ListGetAt --->
<h3>This is a list of usernames who have posted messages</h3>
<cfoutput>@ListLen(temp)@</cfoutput> users.</h3>
<ul>
<cfloop From = "1" TO = "@ListLen(temp)@" INDEX = "Counter">
  <cfoutput>Username @ListGetAt(temp, Counter)@</cfoutput>
</cfloop>
</ul>
```
ListPrepend

Description
Inserts an element at the beginning of a list.

Returns
A copy of the list, with value inserted at the first position.

Category
List functions

Function syntax
ListPrepend(list, value [, delimiters ])

See also
ListAppend, ListInsertAt, ListSetAt

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. Default: 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 delimiters contains more than one delimiter character, ColdFusion uses the first delimiter in the string; if delimiters is omitted, ColdFusion uses a comma.

ColdFusion ignores empty list elements; thus, the list "a,b,c,,d" has four elements.

If the delimiters parameter is the empty string (""), ColdFusion returns the contents of the value parameter.

To add an element to the beginning or end of a list, Macromedia recommends that you do so with code such as the following, rather than with the ListAppend or ListPrepend functions:

```cfml
<cfset MyValue = "another element">
<cfif listLen(myList) is 0>
  <cfset myList = MyValue>
<cfelse>
  <cfset myList = myList & ", " & MyValue>
</cfif>
```

Example
<!--- This example shows ListPrepend --->
<cfquery name = "GetParkInfo" datasource = "cfsnippets">
  SELECT PARKNAME,CITY,STATE FROM PARKS WHERE PARKNAME LIKE 'DE%'</cfquery>
<cfset temp = ValueList(GetParkInfo.ParkName)>
<cfset first_element = ListFirst(temp)>
<cfoutput><p>The original list: #temp#</cfoutput>
<!--- now, insert an element at position 1--->
<cfset temp2 = ListPrepend(Temp, "my Inserted Value")>
**ListQualify**

**Description**
Inserts a string at the beginning and end of list elements.

**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])`

**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 attribute.</td>
</tr>
<tr>
<td>delimiters</td>
<td>A string or a variable that contains one. Character(s) that separate list elements. Default: comma. If this parameter contains more than one character, ColdFusion uses 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 = "cfsnippets">
SELECT FirstName, LastName
FROM Employees
</cfquery>

<h3>ListQualify Example</h3>
<p>This example uses ListQualify to put the full names of the employees in the query within quotation marks.</p>
<cfset myArray = ArrayNew(1)>
<cfloop query = "GetEmployeeNames">
<cfset temp = ArrayAppend(myArray, "#FirstName# #LastName#")>
</cfloop>
```

582    Chapter 3: ColdFusion Functions
<!--- sort that array descending alphabetically --->
<cfset myAlphaArray = ArraySort(myArray, "textnocase")>

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

<cfoutput>
  <p>The contents of the unqualified list are as follows:</p>
  #myList#
</cfoutput>

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

<!--- output the array as a list --->
<cfoutput>
  <p>The contents of the qualified list are as follows:</p>
  #qualifiedList1#
</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>
  #qualifiedList2#
</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

ListFirst, ListGetAt, ListLast

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. Default: 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 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

```
Example

<h3>ListFirst, ListLast, and ListRest Example</h3>
<cfquery name = "GetMessageUser" datasource = "cfsnippets">
  SELECT Username, Subject, Posted
  FROM Messages
</cfquery>

<cfset temp = ValueList(GetMessageUser.Username)>
<p>Before editing the list, it is:&nbsp;
<cfoutput>#ValueList(GetMessageUser.Username)#</cfoutput>. (Users who posted more than once are listed more than once.)
<p>The first user in the list is:
<cfoutput>#ListFirst(temp)#</cfoutput>
<p>The rest of the list is:&nbsp;
<cfoutput>@ListRest(temp)#</cfoutput>. (Users who posted more than once are listed more than once.)
<p>The last user in the list is: <cfoutput>@ListLast(temp)#</cfoutput>
```
ListSetAt

Description
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

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. Default: 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 = "cfsnippets">
SELECT Username, Subject, Posted
FROM Messages
</cfquery>
<cfset temp = ValueList(GetMessageUser.Subject)>
<!--- loop through the list and show it with ListGetAt --->
<h3>This is a list of <cfoutput>#{ListLen(temp)#{}</cfoutput>
subjects posted in messages.</h3>

<cfset ChangedElement = ListGetAt(temp, 2)>
<cfset TempToo = ListSetAt(temp, 2, "I changed this subject", ",")>
<ul>
<cfloop From = "1" To = 
"#ListLen(temptoo)#" INDEX = "Counter">
   <cfoutput><li>(#Counter#) SUBJECT: #ListGetAt(temptoo, Counter)#</li></cfoutput>
</cfloop>
<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 ])`

**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.

For example, in a `textnocase`, `desc` sort of `d,a,a,b,A`, the following occurs:

- ColdFusion MX returns `d,b,A,a,a`
- Earlier ColdFusion releases return `d,b,a,a,A`

(In a `textnocase`, `asc` sort, all ColdFusion releases return `a,a,A,b,d`.)

**Parameters**

<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>
</tbody>
</table>
| sort_type | • numeric: sorts numbers  
• text: sorts text alphabetically, taking case into account (also known as case sensitive). All letters of one case precede the first letter of the other case:  
  - `aabzABZ`, if `sort_order = "asc"` (ascending sort)  
  - `ZBAzbaa`, if `sort_order = "desc"` (descending sort)  
• `textnocase`: sorts text alphabetically, without regard to case (also known as case-insensitive). A letter in varying cases precedes the next letter:  
  - `aAabBbBzzZ`, in an ascending sort; preserves original intra-letter order  
  - `ZzzBbBaAa`, in a descending sort; reverses original intra-letter order |
Usage
ColdFusion ignores empty list elements; thus, the list “a,b,c,,,d” has four elements.

Example

```html
<h3>ListSort Example</h3>

<!--- Find a list of users who wrote messages --->
<cfquery name = "GetMessageUser" datasource = "cfsnippets">
SELECT Username, Subject, Posted
FROM Messages
</cfquery>

<cfset myList = ValueList(GetMessageUser.UserName)>
<p>Here is the unsorted list. </p>
<cfoutput>#myList#</cfoutput>
<p>Here is the list sorted alphabetically:</p>
<cfset sortedList = ListSort(myList, "Text")>
<cfoutput>#sortedList#</cfoutput>
<p>Here is a numeric list that is to be sorted in descending order.</p>
<cfset sortedNums = ListSort("12,23,107,19,1.65","Numeric", "Desc")>
<cfoutput>#sortedNums#</cfoutput>
<p>Here is a list that must be sorted numerically, since it contains negative and positive numbers, and decimal numbers. </p>
<cfset sortedNums2 = ListSort("23.75,-34,471:100,-9745","Numeric", "ASC", ";,:")>
<cfoutput>#sortedNums2#</cfoutput>
<p>Here is a list to be sorted alphabetically without consideration of case.</p>
<cfset sortedMix = ListSort("hello;123,HELLO;jeans,-345.887;ColdFusion:coldfusion","TextNoCase", "ASC", ";,:")>
<cfoutput>#sortedMix#</cfoutput>
```

---

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| sort_order | • asc - ascending sort order. Default.
  - aabzABZ or aAaBbBzzZ, depending on value of sort_type, for letters
  - from smaller to larger, for numbers
  • desc - descending sort order.
  - ZBAzbaa or ZzzBbBaAa, depending on value of sort_type, for letters
  - from larger to smaller, for numbers |
| delimiters | A string or a variable that contains one. Character(s) that separate list elements. Default: comma. If this parameter contains more than one character, ColdFusion uses the first character in the string as the delimiter, and ignores the rest. |
ListToArray

Description
Copies the elements of a list to an array.

Returns
An array

Category
Array functions, Conversion functions, List functions

Function syntax
ListToArray(list [, delimiters ])

See also
ArrayToList

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. Default: 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 attribute as a separate delimiter. Therefore, if the attribute is ",+" ColdFusion will break the list at either a comma plus sign.

Example
<h3>ListToArray Example</h3>
<!--- Find a list of users who wrote messages --->
<cfquery name = "GetMessageUser" datasource = "cfsnippets">
SELECT Username, Subject, Posted
FROM Messages
</cfquery>
<cfset myList = ValueList(GetMessageUser.UserName)>
<p>My list is a list with #ListLen(myList)# elements.
<cfset myArrayList = ListToArray(myList)>
<p>My array list is an array with #ArrayLen(myArrayList)# 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**
ListValueCountNoCase

**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. Default: comma. If this parameter contains more than one character, ColdFusion processes each occurrence of each character as a delimiter.</td>
</tr>
</tbody>
</table>

**Example**

```cfc
<cfquery name = "SearchByDepartment" datasource = "cfsnippets">
SELECT Department FROM Employees</cfquery>
<h3>ListValueCount Example</h3>
<p>This example uses ListValueCount to count employees in a department.</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>
<input type = "Submit" name = "Submit" value = "Search Employee List">
</form>
<!--- wait to have a string for searching defined --->
```
<cfif IsDefined("FORM.Submit") and IsDefined("FORM.departmentName")>
  <cfset myList = ValueList(SearchByDepartment.Department)>
  <cfset numberInDepartment = ListValueCount(myList, FORM.departmentName)>

  <cfif numberInDepartment is 0>
    <h3>There are no employees in <cfoutput>#FORM.departmentName#</cfoutput></h3>
  <cfelseif numberInDepartment is 1>
    <cfoutput><p>There is only one person in #FORM.departmentName#.</p></cfoutput>
  <cfelse>
    <cfoutput><p>There are #numberInDepartment# people in #FORM.departmentName#.</p></cfoutput>
  </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**

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

**See also**
ListValueCount

**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. Default: comma. If this parameter contains more than one character, ColdFusion processes each occurrence of each character as a delimiter.</td>
</tr>
</tbody>
</table>

**Example**

```cfc
<cfquery name = "SearchByDepartment" datasource = "cfsnippets">
SELECT Department
FROM Employees
</cfquery>

<h3>ListValueCountNoCase Example</h3>
<p>This example uses ListValueCountNoCase to count employees in a department.

<form action = "listvaluecountnocase.cfm">
<p>Select a department:</p>
<select name = "departmentName">
<option value = "Accounting">Accounting</option>
<option value = "Administration">Administration</option>
<option value = "Engineering">Engineering</option>
<option value = "Sales">Sales</option>
</select>
<input type = "Submit" name = "Submit" value = "Search Employee List">
</form>
```
<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 #FORM.departmentName#</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>
  </cfelse>
</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</p>
<form action = "ljustify.cfm">
    <p><input type = "Text" value = "<cfoutput>#jString#</cfoutput>" size = 35 name = "justifyString"></p>
    <p><input type = "Submit" name = "" > <input type = "RESET" ></p>
</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
</cfif>
<cfelse>
<br>The natural logarithm of #FORM.number#: #log(FORM.number)#
</cfif>
<cfif FORM.number LTE 0>
<br>Enter a positive real number to get its logarithm to base 10
</cfif>
<cfelse>
<br>The logarithm of #FORM.number# to base 10: #log10(FORM.number)#
</cfif>
<cfif>
</cfoutput>
</cfblock>

<cfinput type = "Text" name = "number" message = "You must enter a number" validate = "float" required = "No">
</cfinput>

<cfform action = "log.cfm">
Enter a number to see its value raised to the E power, its natural logarithm, and the logarithm of number to base 10.
<cfinput type = "Text" name = "number" message = "You must enter a number"
   validate = "float" required = "No">
</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
<h3>Log10 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>You must enter a positive real number to see the natural logarithm of that number</cfif>
    <cfelse><br>The natural logarithm of #FORM.number#: #log(FORM.number)#</cfif>
    <cfif #FORM.number# LTE 0><br>You must enter a positive real number to see the logarithm of that number to base 10</cfif>
    <cfelse><br>The logarithm of #FORM.number# to base 10: #log10(FORM.number)#</cfif>
  </cfoutput>
</cfif>
<cfform action = "log10.cfm">
  Enter a number to find its value raised to the E power, its natural logarithm, and the logarithm of number to base 10.
  <cfinput type = "Text" name = "number" message = "You must enter a number" validate = "float" required = "No" />
  <input type = "Submit" name = "" />
</cfform>
LSCurrencyFormat

Description
Formats a number in a locale-specific currency format. 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
LSEuroCurrencyFormat, LSIsCurrency, LSParseCurrency, LSParseEuroCurrency, SetLocale

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>
</tbody>
</table>
| type      | • local: the currency format and currency symbol used in the locale.  
  - With JDK 1.3, the default for Euro Zone countries is their local currency.  
  - With JDK 1.4, the default for Euro Zone countries is the euro.  
  • international: the international standard currency format and currency symbol of the locale.  
  • none: the currency format used in the locale; no currency symbol |

Usage
This function uses Java standard locale formatting rules on all platforms.

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>Locale</td>
<td>Type = Local</td>
<td>Type = International</td>
<td>Type = None</td>
</tr>
<tr>
<td>-----------------------</td>
<td>------------------</td>
<td>----------------------</td>
<td>--------------</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></td>
<td>100,000.00 BF</td>
<td>EUR100,000.00</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></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></td>
<td>100,000.00 FB</td>
<td>BEF100.000.00</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></td>
<td>100 000.00 F</td>
<td>FRF100 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></td>
<td>öS 100,000.00</td>
<td>ATS100.000.00</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></td>
<td>100.000.00 DM</td>
<td>DEM100.000.00</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>EUR10.000.000</td>
<td>10.000.000</td>
</tr>
<tr>
<td></td>
<td>L. 10.000.000</td>
<td>ITL10.000.000</td>
<td>10.000.000</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>
<tr>
<td>Portuguese (Standard)</td>
<td>100.000.00 $</td>
<td>EUR100.000.00</td>
<td>100.000.00</td>
</tr>
<tr>
<td></td>
<td>R$100.000.00</td>
<td>BRC100.000.00</td>
<td>100,000.00</td>
</tr>
<tr>
<td>Spanish (Mexican)</td>
<td>$100,000.00</td>
<td>MXN100,000.00</td>
<td>100,000.00</td>
</tr>
<tr>
<td>Spanish (Modern)</td>
<td>100,000.00 €</td>
<td>EUR10.000.000</td>
<td>10.000.000</td>
</tr>
<tr>
<td></td>
<td>10 000.000 Pts</td>
<td>ESP10.000.000</td>
<td>10.000.000</td>
</tr>
<tr>
<td>Spanish (Standard)</td>
<td>100.000.00 €</td>
<td>ESP10.000.000</td>
<td>10.000.000</td>
</tr>
<tr>
<td></td>
<td>10 000.000 Pts</td>
<td>EUR10.000.000</td>
<td>10.000.000</td>
</tr>
<tr>
<td>Swedish</td>
<td>100,000.00 kr</td>
<td>SEK100,000.00</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

<h3>LSCurrencyFormat Example</h3>
<p>LSCurrencyFormat returns a currency value using the locale convention. Default value is "local."
</p> <!--- 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>
</p>
<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

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 attribute 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>
<tr>
<td>mask</td>
<td>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 dddd, mmmm, d, and yyyy Default: medium For more information on formats, see LSParseDateTime on page 615.</td>
</tr>
</tbody>
</table>

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 list = "#Server.Coldfusion.SupportedLocales#"
    index = "locale" delimiters = ",">
    <cfset oldlocale = SetLocale(locale)>
    <cfoutput><p><b><i>#locale#</i></b><br>
    #LSDateFormat(Now(), "mmm-dd-yyyy")#<br>
    #LSDateFormat(Now(), "mmmm d, yyyy")#<br>
    #LSDateFormat(Now(), "mm/dd/yyyy")#<br>
    #LSDateFormat(Now(), "d-mmm-yyyy")#<br>
    #LSDateFormat(Now(), "ddd, mmmm dd, yyyy")#<br>
    #LSDateFormat(Now(), "d/m/yy")#<br>
    #LSDateFormat(Now())#<br>
    <hr noshade>
    </cfoutput>
</cfloop>
```
LSEuroCurrencyFormat

Description
Formats a number in a locale-specific currency format.

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
LSParseEuroCurrency, LSCurrencyFormat, SetLocale

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>
<tr>
<td>type</td>
<td>• local: the currency format used in the locale. (Default.)</td>
</tr>
<tr>
<td></td>
<td>• international: the international standard currency format of the locale. For example, EUR10,00</td>
</tr>
<tr>
<td></td>
<td>• none: the currency format used in the locale; no currency symbol</td>
</tr>
</tbody>
</table>

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 676.
## 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 (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>월100,000</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>
<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>MXP100,000.00</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>ESP100,000.00</td>
<td>100,000.00</td>
</tr>
<tr>
<td>Swedish</td>
<td>100,000.00</td>
<td>SEK100,000.00</td>
<td>100,000.00</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>-1 234,56 €</td>
<td>($1,234.56)</td>
</tr>
</tbody>
</table>

Example

```cfc
<h3>LSEuroCurrencyFormat Example</h3>
<p>LSEuroCurrencyFormat returns a currency value using the locale convention. Default value is "local."</p>
 <!--- Loop through list of locales, show currency values for 100,000 units --->
<cfloop list="#Server.Coldfusion.SupportedLocales#" index="locale" delimiters="","">
  <cfset oldlocale = SetLocale(locale)>
  <cfoutput><p><B><I>#locale#</I></B><br>
  Local: #LSEuroCurrencyFormat(100000, "local")#<br>
  International: #LSEuroCurrencyFormat(100000, "international")#<br>
  None: #LSEuroCurrencyFormat(100000, "none")#<br>
  <Hr noshade>
</cfoutput>
</cfloop>
```
**LSIsCurrency**

**Description**
Determines whether a string is 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. Returns True for amounts in the local, international, or none currency formats.

**Category**
Display and formatting functions, Decision functions, International functions

**Function syntax**
```coldfusion
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 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` on page 597.

**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**
```coldfusion
<h3>LSIsCurrency Example</h3>

<cfif IsDefined("FORM.locale")>
  <!--- if locale is defined, set locale to that entry --->
  <cfset NewLocale = SetLocale(FORM.locale)>

  <p>Is the value "<cfoutput>#FORM.myValue#</cfoutput>" a proper currency value for "<cfoutput>#GetLocale()#</cfoutput>?<br/>
    <p>Answer: "<cfoutput>#LSIsCurrency(FORM.myValue)#</cfoutput>"</p>
</cfif>
```

---

**LSIsCurrency** 605
<p><form action = "LSIsCurrency.cfm">
<p>Select a locale for which you would like to check a currency value:
<!--- check the current locale for server --->
<cfset serverLocale = GetLocale()>

--- End of extracted text ---

**LSIsDate**

**Description**
Determines 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**

```cfc
LSIsDate(string)
```

**See also**
CreateDateTime, GetLocale, IsNumericDate, LSDateFormat, ParseDateTime, SetLocale

**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 True.)
- 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**

```cfc
<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">
  Select a locale for which you would like to check a date value:
```
<cfset serverLocale = GetLocale()>
<!--- check the current locale for server --->
<cfset serverLocale = GetLocale()>
**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

**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**
```html
<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>
  <p>Answer: <cfoutput>#LSIsNumeric(FORM.myValue)#</cfoutput></cfif>

<p><form action = "LSIsNumeric.cfm">
  Select a locale for which to check a numeric value:
  ...
```
**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

**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>_</td>
<td>(Underscore.) 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>
<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>
</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><em>$</em>_____</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><em>-</em>_____</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>Character</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>Puts space before positive number; minus sign before negative number.</td>
</tr>
<tr>
<td>.</td>
<td>Separates every third decimal place with a comma (or locale-appropriate symbol).</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. Default: right-justified.</td>
</tr>
<tr>
<td>$</td>
<td>Puts a dollar sign (or locale-appropriate symbol) before formatted number. First character of mask must be the dollar sign ($).</td>
</tr>
<tr>
<td>^</td>
<td>Separates left and right formatting.</td>
</tr>
</tbody>
</table>
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 $1.5543122344752 \times 10^{-14}$ to the converted number. For example, without adding the rounding factor, converting the string value $1.275$ to double with two digits of precision results in a value of $1.27499999999999999$, which would be rounded up to $1.27$. By adding the rounding factor, the conversion correctly results in a value of $1.28$.

If you round off a double, such as $1.99499999999999999999999999999$, where the last decimal is $10E-14$, the rounding factor can cause an incorrect result.

**Example**

```coldfusion
<cfoutput>
  <!--- loop through a list of locales and show number values --->
  <cfset oldlocale = SetLocale(locale)>
  <cfset locale = Server.Coldfusion.SupportedLocales>[0]">
  <cfoutput>
    #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>
</cfoutput>
```

<table>
<thead>
<tr>
<th>Number</th>
<th>Mask</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.21</td>
<td>C(<strong>&quot;)</strong></td>
<td>&quot;(3.21)&quot;</td>
</tr>
<tr>
<td>3.21</td>
<td>C__(&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 $1.5543122344752 \times 10^{-14}$ to the converted number. For example, without adding the rounding factor, converting the string value $1.275$ to double with two digits of precision results in a value of $1.27499999999999999$, which would be rounded up to $1.27$. By adding the rounding factor, the conversion correctly results in a value of $1.28$.

If you round off a double, such as $1.99499999999999999999999999999$, where the last decimal is $10E-14$, the rounding factor can cause an incorrect result.

**Example**

```coldfusion
<cfoutput>
  <!--- loop through a list of locales and show number values --->
  <cfset oldlocale = SetLocale(locale)>
  <cfset locale = Server.Coldfusion.SupportedLocales>[0]">
  <cfoutput>
    #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>
</cfoutput>
```
**LSParseCurrency**

**Description**
Converts a locale-specific currency string into a formatted number. Attempts conversion by comparing the string with each of 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.

See LSCurrencyFormat for a list of the local-specific formats used to parse the currency.

**Example**

```
<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.
```
<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")#<br>
Parsed local Currency: #LSParseCurrency(LSCurrencyFormat(123456.78, "local"))#<br>
International: #LSCurrencyFormat(123456.78999, "international")#<br>
Parsed International Currency: #LSParseCurrency(LSCurrencyFormat(123456.78999, "international"))#<br>
None: #LSCurrencyFormat(123456.78999, "none")#<br>
Parsed None formatted currency: #LSParseCurrency(LSCurrencyFormat(123456.78999, "none"))#<br>
<hr noshade></cfoutput></cfloop>
LSParseDateTime

Description
Converting 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

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>
<tr>
<td>mmm dd, yyyy h:mm:ss tt</td>
<td>Jan 30, 2002 7:02:12 AM</td>
</tr>
<tr>
<td>mmmmm dd, yyyy h:mm:ss tt zzz</td>
<td>January 30, 2002 7:02:23 AM PST</td>
</tr>
<tr>
<td>ddd, mmm dd, yyyy hh:mm:ss</td>
<td>Wed, Jan 30, 2002 07:02:12</td>
</tr>
<tr>
<td>dddid, mmmmm dd, yyyy h:mm:ss tt zzz</td>
<td>Wednesday, January 30, 2002 7:02:12 AM PST</td>
</tr>
</tbody>
</table>
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**

```
<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(), "mm/dd/yyyy")# #LSTimeFormat(Now())#<br>
#LSDateFormat(Now(), "d-mmm-yyyy")# #LSTimeFormat(Now())#<br>
#LSDateFormat(Now(), "ddd, mmmm dd, yyyy")# #LSTimeFormat(Now())#<br>
#LSDateFormat(Now(), "d/m/yy")# #LSTimeFormat(Now())#<br>
#LSDateFormat(Now())# #LSTimeFormat(Now())#<br>
Standard Date/Time: #LSParseDateTime(#LSDateFormat(Now())# #LSTimeFormat(Now())#)<br></cfoutput>
```
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
LSParseCurrency, LSEuroCurrencyFormat, SetLocale

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 602.

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 676.

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)>
    <cfoutput><p>Current Locale: <b><i>#locale#</i></b><br>
       Value in local currency: #LSEuroCurrencyFormat(123456, "local")#
       Parsed using LSParseEuroCurrency:
       #LSParseEuroCurrency(localCurrency)#</cfoutput></p>
</cfloop>
<cfset IntlCurrency = LSEuroCurrencyFormat(123456, "international")>
Value with International currency formatting: #IntlCurrency#
Parsed using LSParseEuroCurrency:
#LSParseEuroCurrency(IntlCurrency)#
<br>
<cfset Currency = LSEuroCurrencyFormat(123456, "none")>
Value with no currency formatting: #Currency#
Parsed using LSParseEuroCurrency:
#LSParseEuroCurrency(Currency)#
<br>
<hr noshade>
</cfoutput>
</cfloop>
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

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
<h3>LSParseNumber Example</h3>
LSParseNumber converts a locale-specific string to a number.
Returns the number matching the value of string.

<!--- loop through a list of locales and show number values --->
<cfloop LIST="#Server.Coldfusion.SupportedLocales#" index="locale" delimiters=".">
<cfset oldlocale = SetLocale(locale)>
<cfoutput><p><B><I>#locale#</I></B><br>
#LSParseNumber(LSNumberFormat(1234.5678, "_________.___"))#<br>
#LSParseNumber(LSNumberFormat(1234.5678, "$_________."))#<br>
#LSParseNumber(LSNumberFormat(1234.5678, "+_________."))#<br>
The actual number: #LSParseNumber(LSNumberFormat(1234.5678, "________."))#<br>
<hr noshade>
<cfset oldlocale = SetLocale(locale)>
</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

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 attribute 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 date/time object is in the range 100 AD–9999 AD.  
A string that is convertible to a time value |
| 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  
• l: Milliseconds  
• t: One-character time marker string, such as A or P.  
• tt: Multiple-character time marker string, such as AM or PM |

The following conform to Java locale-specific time encoding standards. Their exact formats depend on the locale:  
• short: includes hours, minutes; may include AM or PM  
• medium: includes hours, minutes; may include AM or PM  
• long: medium plus time zone  
• full: long, may also include an hour designator  
Default: short
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

Listing 3-2

```
<h3>LSTimeFormat Example</h3>
<p>LSTimeFormat returns a time value using the locale convention.

<!--- loop through a list of locales and show time values ---
<cfloop LIST = 
"Server.Coldfusion.SupportedLocales" index = "locale" delimiters = .">
<cfset oldlocale = SetLocale(locale)>
<cfoutput><p><B><I>#locale#</I></B><br>
#LSTimeFormat(Now())#<br>
#LSTimeFormat(Now(), 'hh:mm:ss')#<br>
#LSTimeFormat(Now(), 'hh:mm:ssst')#<br>
#LSTimeFormat(Now(), 'HH:mm:ss')#<br>
<hr noshade>
</cfoutput>
</cfloop>
```
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

<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">
  Type in some text, and it will be modified by LTrim to remove leading spaces from the left
  <input type = "Text" name = "myText" value = " TEST">
  <input type = "Submit" name = "">
</form>
Max

Description
Determines 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

```cfm
<h3>Max Example</h3>
<cfif IsDefined("FORM.myNum1")>
  <cfif IsNumeric(FORM.myNum1) and IsNumeric(FORM.myNum2)>
    <p>The maximum of the two numbers is <cfoutput>#Max(FORM.myNum1, FORM.myNum2)#</cfoutput>
    <p>The minimum of the two numbers is <cfoutput>#Min(FORM.myNum1, FORM.myNum2)#</cfoutput>
  </cfelse>
  <p>Please enter two numbers
</cfif>

<form action = "max.cfm">
  <h3>Enter two numbers, see the maximum and minimum of them</h3>
  Number 1 <input type = "Text" name = "MyNum1">
  Number 2 <input type = "Text" name = "MyNum2">
  <br><input type = "Submit" name = "" value = "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-quote or double-quote 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. (0 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 is 0, then err --->
<cfif Len(FORM.myText) is not 0>
   <cfif Len(FORM.myText) LTE FORM.RemoveChars>
      <p>Your string <cfoutput>#FORM.myText#</cfoutput> only has <cfoutput>#Len(FORM.myText)#</cfoutput> characters. You cannot output the <cfoutput>#FORM.removeChars#</cfoutput> middle characters of this string because it is not long enough
   <cfelse>
      <p>Your original string: <cfoutput>#FORM.myText#</cfoutput>
      <p>Your changed string, showing only the <cfoutput>#FORM.removeChars#</cfoutput> middle characters:
      <cfoutput>#Mid(Form.myText, FORM.removeChars, Form.countChars)#</cfoutput>
   </cfif>
</cfif>
</cfif>
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

```
<h3>Min Example</h3>
<cfif IsDefined("FORM.myNum1")>
  <cfif IsNumeric(FORM.myNum1) and IsNumeric(FORM.myNum2)>
    <p>The maximum of the two numbers is <cfoutput>#Max(FORM.myNum1, FORM.myNum2)#</cfoutput>
    <p>The minimum of the two numbers is <cfoutput>#Min(FORM.myNum1, FORM.myNum2)#</cfoutput>
  <cfelse>
    <p>Please enter two numbers
  </cfif>
</cfif>

<form action = "min.cfm">
  <h3>Enter two numbers, and see the maximum and minimum of the two numbers</h3>
  Number 1 <input type = "Text" name = "MyNum1">
  <br>Number 2 <input type = "Text" name = "MyNum2">
  <br><input type = "Submit" name = "" value = "See results">
</form>
```
Minute

Description
Extracts the minute value from a date/time object.

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

```cfhtml
<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)#. 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>
```
MonthAsString

Description
Determine the name of the month that corresponds to month_number.

Returns
A string: the name of a month.

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)></p>
<cfoutput>
<p>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)#. 
<cfif IsLeapYear(Year(yourDate))>
This is a leap year
<cfelse>This is not a leap year
</cfif>
</cfoutput>
</cfif>
Now

Description
Gets the current date and time of the computer running the ColdFusion server. The return value can be passed as a parameter to date functions such as DaysInYear or FirstDayOfMonth.

Returns
A date/time object; the current date and time of the computer running the ColdFusion server.

Category
Date and time functions

Function syntax
Now()

See also
CreateDateDateTime, DatePart

Example
<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.</td>
</tr>
<tr>
<td>9</td>
<td>Optional. Digit placeholder. (Shows decimal places more clearly than _)</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>
<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. Default: right-justified.</td>
</tr>
</tbody>
</table>
### Mask character | Meaning
--- | ---
$ | Puts a dollar sign before formatted number. First character of mask must be the dollar sign ($).
^ | Separates left and right formatting.

**Note:** If you do not specify a sign for the mask, positive and negative numbers do not align in columns. To put a plus sign or space before positive numbers and a minus sign before negative numbers, use the plus or minus sign, respectively.

#### Usage
This function uses Java standard locale formatting rules on all platforms.

The position of symbols in format masks determines where the codes take effect. For example, if you put a dollar sign at the far left of a format mask, ColdFusion displays a dollar sign at the left edge of the formatted number. If you separate the dollar sign on the left edge of the format mask by at least one underscore, ColdFusion displays the dollar sign just to the left of the digits in the formatted number.

These examples show how symbols determine formats:

<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><em>-</em>___.</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>
<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>
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 1.994999999999999999999999999999, 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**

```cfml
<h3>NumberFormat Example</h3>
<cfloop FROM = 1000 TO = 1020 INDEX = "counter">
  <cfset CounterRoot2 = counter * sqr(2)>

  <!--- Show result in default format, adding comma for thousands place; --->
  <cfoutput>
    <pre>#counter# * Square Root of 2: #NumberFormat(CounterRoot2, '_____.')#</pre>
  </cfoutput>
  <pre>#counter# * Square Root of 2: #NumberFormat(CounterRoot2)#</pre>
</cfloop>
```
**ParagraphFormat**

**Description**

Replaces characters in a string:
- Single newline characters (CR/LF sequences) with spaces
- Double newline characters with HTML paragraph tags (<p>)

**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**

```cfm
<h3>ParagraphFormat Example</h3>
<p>Enter text into this textarea, and see it returned as HTML.
<cfif IsDefined("FORM.myTextArea")>
<p>Your text area, formatted
<p><cfoutput>#ParagraphFormat(FORM.myTextArea)#</cfoutput>
</cfif>
</cfif>
<!--- use #Chr(10)##Chr(13)# to simulate a line feed/carriage return combination; i.e. a return --->
<form action = "paragraphformat.cfm">
<textarea name = "MyTextArea" cols = "35" rows = 8>
This is sample text and you see how it scrolls
<cfoutput>#Chr(10)##Chr(13)#</cfoutput>
From one line
<cfoutput>#Chr(10)##Chr(13)##Chr(10)##Chr(13)#</cfoutput>
to the next
</textarea>
<input type = "Submit" name = "Show me the HTML version">
</form>
```
ParameterExists

Description
This function is deprecated. 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 1990-1999.</td>
</tr>
</tbody>
</table>
| pop-conversion | • 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. |

**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.

**Note:** This function does not accept POP dates.

**Example**

```<h3>ParseDateTime Example</h3><cfif IsDefined("form.theTestValue")><cfif IsDate(form.theTestValue)>
   <h3>The expression <cfoutput>#DE(form.theTestValue)#</cfoutput> is a valid date</h3>
   <p>The parsed date/time is: <cfoutput>#ParseDateTime(form.theTestValue)#</cfoutput></p>
<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())#" #TimeFormat(Now())#</CFOUTPUT>">
<input type="Submit" value="Parse the Date" name=""></form>
**Pi**

**Description**
Gets the mathematical constant $\pi$, accurate to 15 digits.

**Returns**
The number 3.14159265358979.

**Category**
Mathematical functions

**Function syntax**
`Pi()`

**See also**
`ASin`, `Cos`, `Sin`, `Tan`

**Example**

```html
<h3>Pi Example</h3>
<!--- By default, ColdFusion displays only 11 significant digits.
Use NumberFormat to display all 15. --->
The Pi function Returns the number
`<cfoutput>
#NumberFormat(Pi(), "_._______________")#
</cfoutput>`
the mathematical constant pi, accurate to 15 digits.
```
PreserveSingleQuotes

Description
Prevents ColdFusion from automatically escaping single quotation mark characters that are contained in a variable. ColdFusion does not evaluate the argument.

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:
```<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>

```coldfusion
cfset List = "'Suisun', 'San Francisco', 'San Diego'"
cfquery name = "GetCenters" datasource = "cfsnippets"
   SELECT Name, Address1, Address2, City, Phone
   FROM Centers
   WHERE City IN (#PreserveSingleQuotes(List)#)
</cfquery>
<p>We found #GetCenters.RecordCount# records.</p>
```
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, array-name)

See also
QueryNew, QueryAddRow, QuerySetCell

History
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 that was created with QueryNew.</td>
</tr>
<tr>
<td>column-name</td>
<td>Name of the new column.</td>
</tr>
<tr>
<td>array-name</td>
<td>Name of an array whose elements are to populate the new column.</td>
</tr>
</tbody>
</table>

Usage
You can add columns to query objects such as queries retrieved with the cfquery tag or queries created with the QueryNew function. You cannot us the QueryAddColumn function on a cached query.

Useful for generating a query object from the arrays of output parameters that Oracle stored procedures can generate.

Example

```coldfusion
<h3>QueryAddColumn Example</h3>
<p>This example adds three columns to a query object and then populates the columns with the contents of three arrays.</p>
<p>After populating the query, the example shows, in tabular format, the contents of the columns.</p>
<!---- make a query --->
<cfset myQuery = QueryNew("")>

<!---- create an array --->
<cfset FastFoodArray = ArrayNew(1)>
<cfset FastFoodArray[1] = "French Fries">
<cfset FastFoodArray[2] = "Hot Dogs">
<cfset FastFoodArray[3] = "Fried Clams">
<cfset FastFoodArray[4] = "Thick Shakes">
```
<cfset nColumnNumber = QueryAddColumn(myQuery, "FastFood", FastFoodArray)>  
<!--- create a second array --->  
<cfset FineCuisineArray = ArrayNew(1)>  
<cfset FineCuisineArray[1] = "Lobster">  
<cfset FineCuisineArray[2] = "Flambe">  
<!--- add a second column to the query --->  
<cfset nColumnNumber2 = QueryAddColumn(myQuery, "FineCuisine",  
          FineCuisineArray)>  
<!--- create a third array --->  
<cfset HealthFoodArray = ArrayNew(1)>  
<cfset HealthFoodArray[1] = "Bean Curd">  
<cfset HealthFoodArray[2] = "Yogurt">  
<cfset HealthFoodArray[3] = "Tofu">  
<!--- add a third column to the query --->  
<cfset nColumnNumber3 = QueryAddColumn(myQuery, "HealthFood",  
          HealthFoodArray)>  
<table cellspacing = "2" cellpadding = "2" border = "0">  
<tr>  
<th align = "left">Fast Food</th>  
<th align = "left">Fine Cuisine</th>  
<th align = "left">Health Food</th>  
</tr>  
<cfoutput query = "myQuery">  
<tr>  
<td>#FastFood#</td>  
<td>#FineCuisine#</td>  
<td>#HealthFood#</td>  
</tr>  
</cfoutput>  
</table>  
<p><b>Note:</b> Because there are fewer elements in the Fine Cuisine  
and Health Food arrays, QueryAddColumn added padding to the  
corresponding columns in the query.</p>
QueryAddRow

Description
Adds a specified number of empty rows to a query.

Returns
The number of rows in the query

Category
Query functions

Function syntax
QueryAddRow(query [, number ])

See also
QueryAddColumn, QueryAddRow, QuerySetCell, QueryNew

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. Default: 1.</td>
</tr>
</tbody>
</table>

Example

<h3>QueryAddRow Example</h3>

<!--- start by making a query --->
<cfquery name = "GetCourses" datasource = "cfsnippets">
   SELECT Course_ID, Descript
   FROM Courses
</cfquery>
<p>The Query "GetCourses" has #GetCourses.RecordCount# records.</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 #GetCourses.RecordCount# records.</p>
<cfoutput query="GetCourses">
<PRE>#Course_ID# #Course_Number# #Descript#</PRE></cfoutput>
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)

See also
QueryAddColumn, QueryAddRow, QuerySetCell

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>columnlist</td>
<td>A string or a variable that contains one. Delimited list of column names, or an empty string.</td>
</tr>
</tbody>
</table>

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

Example
<h3>QueryNew Example</h3>
<p>We will construct a new query with two rows:
<cfset myQuery = QueryNew("name, address, phone")>
<cfset newRow = QueryAddRow(MyQuery, 2)>
<cfset temp = QuerySetCell(myQuery, "name", "Fred", 1)>
<cfset temp = QuerySetCell(myQuery, "address", "9 Any Lane", 1)>
<cfset temp = QuerySetCell(myQuery, "phone", "555-1212", 1)>
<cfset temp = QuerySetCell(myQuery, "name", "Jane", 2)>
<cfset temp = QuerySetCell(myQuery, "address", "14 My Street", 2)>
<cfset temp = QuerySetCell(myQuery, "phone", "555-1444", 2)>
<cfoutput query = "myQuery"/>
<pre>#name##address##phone#</pre>
To get any element in the query, we can output it individually
<cfoutput>
<p>#MyQuery.name[2]#'s phone number: #MyQuery.phone[2]#
</cfoutput>
QuerySetCell

Description
Sets a cell to a value. If no row number is specified, the cell on the last row is set.

Returns
True, if successful; False, otherwise.

Category
Query functions

Function syntax
QuerySetCell(query, column_name, value [, row_number ])

See also
QueryAddColumn, QueryAddRow, QueryNew

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. Default: 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 = "cfsnippets">
  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

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

<h3>QuotedValueList Example</h3>
<!--- use the contents of one query to create another dynamically --->
<cfset List = "'BIOL', 'CHEM'">
<!--- first, get the department IDs in our list --->
<cfquery name = "GetDepartments" datasource = "cfsnippets">
  SELECT Dept_ID FROM Departments
  WHERE Dept_ID IN (#PreserveSingleQuotes(List)#)
</cfquery>
<!--- now, select the courses for that department based on the quotedValueList produced from our previous query --->
<cfquery name = "GetCourseList" datasource = "cfsnippets">
  SELECT * FROM CourseList
  WHERE Dept_ID IN (#QuotedValueList(GetDepartments.Dept_ID)#)
</cfquery>
<!--- now, output the results --->
<cfoutput QUERY = "GetCourseList" >
<pre>#Course_ID##Dept_ID##CorNumber##CorName#</pre>
</cfoutput>
Rand

**Description**
Generates a random number.

**Returns**
A random decimal number, in the range 0 – 1.

**Category**
Mathematical functions

**Function syntax**
```
Rand()
```

**See also**
Randomize, RandRange

**Usage**
To ensure greater randomness, call the Randomize function before calling Rand.

**Example**
```
<h3>Rand Example</h3>
<p>Rand() returns a random number in the range 0 to 1.</p>
<p>Rand() returned: <cfoutput>#Rand()#</cfoutput></p>
<p><A HREF = "rand.cfm">Try again</A></p>
```
Randomize

Description
Seeds the ColdFusion random number generator with an integer number. Seeding the generator helps ensure that the Rand function generates highly random numbers.

Returns
A non-random decimal number, in the range 0 – 1.

Category
Mathematical functions

Function syntax
Randomize(number)

See also
Rand, RandRange

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>number</td>
<td>A number</td>
</tr>
</tbody>
</table>

Usage
Call this function before calling Rand. Although this function returns a decimal number, it is not a random number.

Example
<h3>Randomize Example</h3>
<p>Call Randomize to seed the random number generator. This helps to ensure the randomness of numbers generated by Rand.</p>
<cfif IsDefined("FORM.myRandomInt")>
  <cfif IsNumeric(FORM.myRandomInt)>
    <cfoutput><p><b>Seed value is #FORM.myRandomInt#</b></p>
    <cfset r = Randomize(FORM.myRandomInt)>
    <cfloop index = "i" from = "1" to = "10" step = "1">
      <cfoutput>Next random number is #Rand()#</cfoutput>
    </cfloop>
  </cfoutput>
</cfif>
<cfelse>
  <p>Please enter a number.</p>
</cfif>

<form action = "randomize.cfm">
  <p>Enter a number to seed the randomizer:</p>
  <input type = "Text" name = "MyRandomInt">
  <input type = "Submit" name = "">
</form>
RandRange

Description
Generates a random integer between two specified numbers. Requests for random integers that are greater than 100,000,000 result in non-random numbers, to prevent overflow during internal computations.

Returns
A random integer

Category
Mathematical functions

Function syntax
RandRange(number1, number2)

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 less than 100,000,000</td>
</tr>
</tbody>
</table>

Example
<h3>RandRange Example</h3>
<p>RandRange returns an integer between two specified integers.
<cfif IsDefined("FORM.myInt")>
  <p>RandRange returned:
  <cfoutput>#RandRange(FORM.myInt, FORM.myInt2)#</cfoutput>
</cfif>

<cfform action = "randRange.cfm"
  <p>Enter a number to seed the randomizer:
  <cfinput type = "Text" name = "MyInt" value = "1" RANGE = "1,100000000" message = "Please enter a value between 1 and 100,000,000" validate = "integer" required = "Yes">
  <cfinput type = "Text" name = "MyInt2" value = "500" RANGE = "1,100000000" message = "Please enter a value between 1 and 100,000,000" validate = "integer" required = "Yes">
  <p><input type = "Submit" name = "">
</cfform>
**REFind**

**Description**

Uses a regular expression (RE) to search a string for a pattern. The search is case sensitive.

For more information on regular expressions, including escape sequences, anchors, and modifiers, see Chapter 7, “Using Regular Expressions in Functions,” in Developing ColdFusion MX Applications.

**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>
This function finds the first occurrence of a regular expression in a string. To find the second and subsequent instances of the expression or of subexpressions in it, you call this function more than once, each time with a different start position. To determine the next start position, use the `returnsubexpressions` parameter, and add the value returned in the first element of the length array to the value in the first element of the position array.

### Usage

This function finds the first occurrence of a regular expression in a string. To find the second and subsequent instances of the expression or of subexpressions in it, you call this function more than once, each time with a different start position. To determine the next start position, use the `returnsubexpressions` parameter, and add the value returned in the first element of the length array to the value in the first element of the position array.

### Example

```cftags
<h3>REFind Example</h3>
<p>This example shows the use of the REFind function with and without the `returnsubexpressions` parameter set to True. If you do not use the `returnsubexpressions` parameter, REFind returns the position of the first occurrence of a regular expression in a string starting from the specified position. Returns 0 if no occurrences are found.</p>
<p>REFind("a+c+", "abcaaccdd")</p>
```
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: #st.pos[1]#. The value of st.len[1] is: #st.len[1]#.

Substring is #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:
```cfset st1 = REFind("([[:alpha:]][ ]+)(\1)\)*.testString,1,"TRUE")
<cfset st1 = REFind("([[:alpha:])+](\1)\)*.testString,1,"TRUE")
```

The number of elements in each array is #ArrayLen(st1.pos)#.

First whole expression match: position is #st1.pos[1]#; length is #st1.len[1]#; whole expression match is #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.

```cfloop index = "1" from = "2" to = "#ArrayLen(st1.pos)#"
<cfoutput>Position is #st1.pos[i]#; Length is #st1.len[i]#; Substring is #Mid(testString,st1.pos[i],st1.len[i])#</cfoutput>
</cfloop>"
**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 *Developing ColdFusion MX Applications*.

**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>
<tbody>
<tr>
<td><code>reg_expression</code></td>
<td>Regular expression for which to search. Case-insensitive. For more information, see Chapter 7, “Using Regular Expressions in Functions,” in <em>Developing ColdFusion MX Applications.</em></td>
</tr>
<tr>
<td><code>string</code></td>
<td>A string or a variable that contains one. String in which to search.</td>
</tr>
</tbody>
</table>
RefindNoCase

This function finds the first occurrence of a regular expression in a string. To find the second and subsequent instances of the expression or of subexpressions in it, you call this function more than once, each time with a different start position. To determine the next start position, use the `returnsubexpressions` parameter, and add the value returned in the first element of the length array to the value in the first element of the position array.

**Example**

This example demonstrates the use of the RefindNoCase function with and without the `returnsubexpressions` parameter set to True.

- **If you do not use the `returnsubexpressions` parameter, RefindNoCase returns the position of the first occurrence of a regular expression in a string starting from the specified position. Returns 0 if no occurrences are found.**

```cfoutput>
<cfoutput>#RefindNoCase("a+c+", "abcaaccdd")#</cfoutput></p>
<cfoutput>#RefindNoCase("a+c+", "abcaaccdd")#</cfoutput></p>
<cfoutput>#RefindNoCase("[[[alpha:]]]+", "abcaaccdd")#</cfoutput></p>
<cfoutput>#RefindNoCase("[[\?\&\]rep = ", "report.cfm?rep = 1234&u = 5")#</cfoutput></p>
</cfoutput></p>
<!--- Set startPos to one; returnMatchedSubexpressions = True --->

- **If you do use the `returnsubexpressions` parameter, RefindNoCase returns the position and length of the first occurrence of a regular expression in a string starting from the specified position. The position and length variables are stored in a structure. To access position and length information, use the keys `pos` and `len`, respectively.**

```cfset teststring = "The cat in the hat hat came back!"
<cfoutput>
<cfoutput><b)#teststring#</b></cfoutput></p>
<cfoutput>#RefindNoCase("[[\[:alpha:\]]]+", testString,1,"True")#</cfoutput></p>
</cfoutput></p>
<p>When the string in which the function is to search is: [b] RefindNoCase("[[[alpha:]]]+", testString,1,"True")[/b]</p>
<p>This function returns a structure that contains two arrays: pos and len.</p>
<p>To create this structure you can use a CFSET statement. For example:</p>
```cfset st = RefindNoCase("[[[alpha:]]]+", testString,1,"True")&gt;
<cfset st = REFindNoCase("[[:alpha:]]+", testString, 1, "True")>
<p>
The number of elements in each array: #ArrayLen(st.pos)#.</p>
<p>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.</p>
<p>The value of st.pos[1] is: <cfoutput>#st.pos[1]#</cfoutput>.</p>
<p>The value of st.len[1] is: <cfoutput>#st.len[1]#</cfoutput>.</p>
<p>Substring is <b>[#Mid(testString, st.pos[1], st.len[1])#]</b></p>
<hr size = "2" color = "#0000A0">
<p>However, if you use parentheses to denote subexpressions in the regular expression, the first element contains the position and length of the first instance of the whole expression. The position and length of the first instance of each subexpression will be included in additional array elements.</p>
<p>For example:
<&lt;CFSET st1 = REFindNoCase("[[:alpha:]]+[ ]+(\1)", testString, 1, "True")&gt;
<cfset st1 = REFindNoCase("([[:alpha:]]+)[ ]+(\1)", testString, 1, "True")>
<p>The number of elements in each array is #ArrayLen(st1.pos)#.</p>
<p>First whole expression match; position is #st1.pos[1]#; length is #st1.len[1]#; whole expression match is <b>[#Mid(testString, st1.pos[1], st1.len[1])#]</b></p>
<p>Subsequent elements of the arrays provide the position and length of the first instance of each parenthesized subexpression therein.</p>
<cfloop index = "i" from = "2" to = "#ArrayLen(st1.pos)#">
<p>Position is #st1.pos[i]#; Length is #st1.len[i]#; Substring is <b>[#Mid(testString, st1.pos[i], st1.len[i])#]</b></p>
</cfloop><br>
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 cfoject 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();
ReleaseObject(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
<br/>
Returns a string with <i>count</i> characters removed from the start position. Returns 0 if no characters are found.
<br/>
<br/>
<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>
<cfelse>
<p>Your original string: #FORM.myString#
<p>Your modified string:#RemoveChars(FORM.myString, FORM.numChars, FORM.start)#
</cfoutput>
RepeatString

Description
Creates a string that contains a specified number of repetitions of the specified string.

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>
<ol>
  <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>
</ol>
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>
<tr>
<td>scope</td>
<td>• one: replace the first occurrence (default)</td>
</tr>
<tr>
<td></td>
<td>• all: replace all occurrences</td>
</tr>
</tbody>
</table>

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:
replace("The quick brown fox jumped over the lazy cow, dog, and cat.", ",", ",", ",","All")

Example
<h3>Replace Example</h3>

<p>The Replace function returns <i>string</i> with <i>substring1</i> replaced by <i>substring2</i> in the specified scope. This is a case-sensitive search.
</p>
<cif IsDefined("FORM.MyString")>
<p>Your original string. <cfoutput>#FORM.MyString#</cfoutput>
</p>
<cif>
<p>You wanted to replace the substring <cfoutput>#FORM.MySubstring1#</cfoutput> with the substring <cfoutput>#FORM.MySubstring2#</cfoutput>.
</p>
<p>The result: <cfoutput>#Replace(FORM.myString, FORM.MySubstring1, FORM.mySubString2)#</cfoutput></p>
</cif>
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.
<cffif IsDefined("FORM.MyString")>
<p>Your original string. <cfoutput>#FORM.MyString#</cfoutput>
</cffif>
<p>You wanted to replace the substring <cfoutput>#FORM.MySubstring1#</cfoutput> with the substring <cfoutput>#FORM.MySubstring2#</cfoutput>. The result: <cfoutput>#Replacelist(FORM.myString, FORM.MySubstring1, FORM.mySubString2)#</cfoutput>
</p></form>

<h3>ReplaceList Example Two</h3>
<cfsset stringtoreplace = "The quick brown fox jumped over the lazy dog.">
<cfoutpu}
#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: Replace the first occurrence (default)  
            • all: Replace all occurrences |

Example
<h3>ReplaceNoCase Example</h3>
<p>The ReplaceNoCase function returns <i>string</i> with <i>substring1</i> replaced by <i>substring2</i> in the specified scope. The search/replace is case-insensitive.</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> with the substring <cfoutput>#FORM.MySubstring2#</cfoutput>.</p>
<p>The result: <cfoutput>#ReplaceNoCase(FORM.myString, FORM.MySubstring1, FORM.mySubString2)#</cfoutput></p>
</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 attribute is set to one, returns a string with the first occurrence of the regular expression replaced by the value of substring.
If the scope attribute is set to all, returns a string with all occurrences of the regular expression replaced by the value of substring.
If the function finds no matches, it returns a copy of the string unchanged.

Category
String functions

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: replace the first occurrence (default) • all: replace all occurrences</td>
</tr>
</tbody>
</table>

Usage
For details on using regular expressions, see Chapter 7, “Using Regular Expressions in Functions,” in Developing ColdFusion MX Applications.
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")</cfoutput>`
- `<cfoutput>`REReplace("CABARET",*[A-Z]*,"G","ALL")</cfoutput>`
- `<cfoutput>`REReplace("I love jellies","jell(y|ies)","cookies")</cfoutput>`
- `<cfoutput>`REReplace("I love jelly","jell(y|ies)","cookies")</cfoutput>`
**REReplaceNoCase**

**Description**
Uses a regular expression to search a string for a string pattern and replace it with another. The search is case-insensitive.

**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**
REFind, REFindNoCase, 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 Developing ColdFusion MX Applications.</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: Replace the first occurrence of the regular expression. Default.</td>
</tr>
<tr>
<td></td>
<td>• all: Replace 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 Developing ColdFusion MX Applications.

**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>`
`<cfoutput>"<p>REReplaceNoCase("cabaret","C|B","G","ALL")</p>"
`<cfoutput>`
`<p>REReplaceNoCase("cabaret","[A-Z","G","ALL")</p>`
`<cfoutput>`
`<p>REReplaceNoCase("I LOVE JELLIES","jelly\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)#<cfoutput>`
This code outputs the following:
```
reverse(6*2) equals 21
```

Example

```<h3>Reverse Example</h3>
<p>Reverse returns your string with the positions of the characters reversed.
<cfif IsDefined("FORM.myString")>
  <p>Reverse returned:
  <cfoutput>#Reverse(FORM.myString)#</cfoutput>
</cfelse>
  <p>Please enter a string to be reversed.
</cfif>
</h3>
<form action = "reverse.cfm">
  <p>Enter a string to be reversed:
  <input type = "Text" name = "MyString">
</form>
```
Right

Description
Gets a specified number of characters from a string, beginning at the right.

Returns
- If the length of the string is greater than or equal to $\text{count}$, the rightmost $\text{count}$ characters of the string
- If $\text{count}$ is greater than the length of the string, the whole string
- If $\text{count}$ is greater than 1, and the string is empty, an empty string

Category
String functions

Function syntax
$\text{Right}(\text{string}, \text{count})$

See also
Mid, Left, 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 or a variable that contains one. Number of characters to return.</td>
</tr>
</tbody>
</table>

Example

```html
<h3>Right Example</h3>
<cif IsDefined("Form.MyText")>
    <!--- if len is 0, then err --->
    <cif Len(FORM.myText) is not 0>
        <cif Len(FORM.myText) LTE FORM.RemoveChars> only has $\text{len}(<output>FORM.myText</output>) characters. You cannot output the $\text{right}(<output>FORM.myText</output>, FORM.RemoveChars)$ rightmost characters of this string because it is not long enough
            </cifelse>
        <p>Your original string: <output>FORM.myText</output></p>
        <p>Your changed string, showing only the $\text{right}(<output>FORM.myText</output>, FORM.RemoveChars)$ rightmost characters: <output>$\text{right}(FORM.myText, FORM.removeChars)$</output></p>
    </cifelse>
    <p>Please enter a string</p>
</cif>
```

Right  669
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>
</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 = "<cfoutput>#jString#</cfoutput>" size = 35 name = "justifyString"></input>
<input type = "Submit" name = "" > <input type = "reset"></input>
</form>
Round

Description
Rounds a number to the closest integer.

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>

Example
<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(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>string</td>
<td>A string or a variable that contains one</td>
</tr>
</tbody>
</table>

Example

```<h3>RTrim Example</h3>
<cfif IsDefined("FORM.myText")>
<cfoutput>
Your string: "#FORM.myText#"
Your string: "#RTrim(FORM.myText)#" (right trimmed)
</cfoutput>
</cfif>
</form>

<form action = "Rtrim.cfm" method="post">
<p>Enter some text. It will be modified by Rtrim to remove spaces from the right.
</p><input type = "Text" name = "myText" value = "TEST   ">
<p><input type = "Submit" name = "">
</form>```
Second

Description
Extracts the ordinal for the second from a date/time object.

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>
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>
</table>
| scope_name | • url  
| | • form  
| charset | 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 |

**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. Before using URL or Form variables, call this function (typically, in the Application.cfm page) to set the encoding and avoid 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 provides general information on character encoding and the web, and has several useful links.
• 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 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

<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

**History**
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 the following locale names:

<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>
<tr>
<td>Chinese (Taiwan)</td>
<td>French (Standard)</td>
<td>Norwegian (Nynorsk)</td>
</tr>
<tr>
<td>Dutch (Belgian)</td>
<td>French (Swiss)</td>
<td>Portuguese (Brazilian)</td>
</tr>
<tr>
<td>Dutch (Standard)</td>
<td>German (Austrian)</td>
<td>Portuguese (Standard)</td>
</tr>
<tr>
<td>English (Australian)</td>
<td>German (Standard)</td>
<td>Spanish (Modern)</td>
</tr>
<tr>
<td>English (Canadian)</td>
<td>German (Swiss)</td>
<td>Spanish (Standard)</td>
</tr>
<tr>
<td>English (New Zealand)</td>
<td>Italian (Standard)</td>
<td>Swedish</td>
</tr>
<tr>
<td>English (UK)</td>
<td>Italian (Swiss)</td>
<td></td>
</tr>
<tr>
<td>English (US)</td>
<td>Japanese</td>
<td></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 in 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**

```
<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 <cfoutput>#GetLocale()#</cfoutput>  
The old locale was <cfoutput>#SetLocale("English (UK)")#</cfoutput>  
The locale is now <cfoutput>#GetLocale()#</cfoutput>  
```
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 #MyTimeout#</p>
<cfset code = SetProfileString(IniPath, Section, "timeout", MyTimeout)>
<p>Value returned from SetProfileString: #code#</p>
</cfif>
<cfelse>
<hr size = "2" color = "red">
<p>Timeout value should be greater than zero in order to provide time for user response.</p>
</cfelse>
</cfif>
The timeout value in your initialization file is already set to: #MyTimeout#.

Timeout is set to: #timeout#.

Default directory is: #default#.

<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>
<td></td>
</tr>
</table>
</form>
SetVariable

Description
This function is no longer required in well-formed ColdFusion pages.
Sets a variable in the name parameter to the value of the value parameter.

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
Before this function is called, the client variable must exist, and the cfapplication tag ClientManagement attribute must be set to "Yes".

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 pound signs (#); for example:

```coldfusion
<cfset DynamicVar2 = "ABD">
<cfset "#DynamicVar2#" = "Test Value2">
```

Also, the following lines are equivalent:

```coldfusion
<cfset "myVar#i#" = myVal>
SetVariable("myVar" & i, myVal)
```

For more information, see Chapter 4, “Using Expressions and Pound Signs,” in Developing ColdFusion MX Applications.

Example

```coldfusion
<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.
Sgn(14): <cfoutput>#Sgn(14)#</cfoutput>
Sgn(21-21): <cfoutput>#Sgn(21-21)#</cfoutput>
Sgn(-0.007): <cfoutput>#Sgn(-0.007)#</cfoutput>
```
**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**
\[ \text{Sin}(\text{number}) \]

**See also**
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 \( \frac{\pi}{180} \). To convert radians to degrees, multiply radians by \( \frac{180}{\pi} \).

**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**

```<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>
</cfif>
<!--- If input is not a number, show an error message --->
</cfif>
</h3>
<form action="#CGI.script_name#" method="post">
Enter an angle in degrees to get its sine:<br>
<br><input type = "Text" name = "sinNum" size = "15">
```

---
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

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 <cfoutput>#FORM.myString#</cfoutput></p>
<p>Your set of characters was <cfoutput>#FORM.mySet#</cfoutput></p>
<p>Your string up until one of the characters in the set is: <cfoutput>#SpanExcluding(FORM.myString, FORM.mySet)#</cfoutput></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 action = "spanexcluding.cfm">
<p>Enter a string: <input type = "Text" name = "myString" value = "Hey, you!"></p>
<p>And a set of characters: <br><input type = "Text" name = "mySet" value = "Ey"></p>
<br><input type = "Submit" name = ""></form>
SpanIncluding

Description
Gets characters from a string, from the beginning to a character that is not in a specified set of characters. The search is case-sensitive.

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

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**

```html
<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, Other 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
Structure functions

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 append.</td>
</tr>
<tr>
<td>struct2</td>
<td>Structure that contains the data to append to struct1.</td>
</tr>
</tbody>
</table>
| overwriteFlag | • Yes: values in struct2 overwrite corresponding values in struct1. Default.  
|             | • No        |

Usage
This function appends the fields and values of struct2 to struct1; struct2 is not modified. If struct1 already contains a field of struct2, overwriteFlag determines whether the value in struct2 overwrites it.

A structure's keys are unordered.

Example
<html>
<body>
<!-- Create a Name structure --->
<cfset nameCLK=StructNew()>
<cfset nameCLK.first="Chris">
<cfset nameCLK.middle="Lloyd">
<cfset nameCLK.last="Gilson">
<!-- Create an address struct --->
<cfset addrCLK=StructNew()>
<cfset addrCLK.street="17 Gigantic Rd">
<cfset addrCLK.city="Watertown">
<cfset addrCLK.state="MA">
<cfset addrCLK.zip="02472">
<!-- Create a Person structure --->
<cfset personCLK=StructNew()>
<cfset personCLK.name=#nameCLK#>
<cfset personCLK.addr=#addrCLK#>
<!-- Display the contents of the person struct before the Append --->
</body>
</html>
The person struct before the Append call:
<cfloop collection=#personCLK# item="myItem">
    #myItem#
</cfoutput>
</cfloop>

<!--- Merge the Name struct into the top-level person struct --->
<cfset bSuccess = StructAppend( personCLK, addrCLK )>

<!--- Display the contents of the person struct, after the Append --->
<p>
The person struct after the Append call:
<cfloop collection=#personCLK# item="myItem">
    #myItem#
</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**
Structure functions

**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**

```cfml
<!--- Shows StructClear function. Calls cf_addemployee custom tag which uses the addemployee.cfm file. --->
<body>
<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.
<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
Structure functions

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 466.

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></td>
</tr>
<tr>
<td>Binary</td>
<td></td>
</tr>
<tr>
<td>Base64</td>
<td>Reference</td>
</tr>
<tr>
<td>structure.array</td>
<td>Value</td>
</tr>
<tr>
<td>structure.nested_structure</td>
<td>Reference</td>
</tr>
</tbody>
</table>
Example

```coldfusion
// This script creates a structure that StructCopy copies by value. <br>
<cfscript>
    // Create elements.
    s = StructNew();
    s.array = ArrayNew(2);

    // Assign simple values to original top-level structure fields.
    s.number = 99;
    s.string = "hello tommy";

    // Assign values to original top-level array.
    s.array[1][1] = "one one";
    s.array[1][2] = "one two";
</cfscript>

<!--- Output original structure --->
<hr>
<b>Original Values</b><br>
<cfoutput>
    // Simple values <br>
    s.number = #s.number#<br>
    s.string = #s.string#<br>
    // Array value <br>
    s.array[1][1] = #s.array[1][1]#<br>
    s.array[1][2] = #s.array[1][2]#<br>
</cfoutput>

// Copy this structure to a new structure. <br>
<cfset copied = StructCopy(s)>

<cfscript>
    // Change the values of the original structure. <br>
    s.number = 100;
    s.string = "hello tommy (modified)";
    s.array[1][1] = "one one (modified)";
    s.array[1][2] = "one two (modified)";
</cfscript>

<!--- Output original structure --->
<hr>
<b>Modified Original Values</b><br>
<cfoutput>
    // Simple values <br>
    s.number = #s.number#<br>
    s.string = #s.string#<br>
    // Array value <br>
    s.array[1][1] = #s.array[1][1]#<br>
    s.array[1][2] = #s.array[1][2]#<br>
</cfoutput>

// Copy this structure to a new structure. <br>
<cfset copied = StructCopy(s)>

<cfscript>
    // Copy this structure to a new structure. <br>
    copied.number = #copied.number#<br>
    copied.string = #copied.string#<br>
    copied.array[1][1] = #copied.array[1][1]#<br>
    copied.array[1][2] = #copied.array[1][2]#<br>
</cfscript>
```

<table>
<thead>
<tr>
<th>Variable type</th>
<th>Assigned by</th>
</tr>
</thead>
<tbody>
<tr>
<td>structure.object</td>
<td>Reference</td>
</tr>
<tr>
<td>structure.query</td>
<td>Reference</td>
</tr>
</tbody>
</table>

---

<!--- This code shows assignment by-value and by-reference. --->

// This script creates a structure that StructCopy copies by value. <br>
<cfscript>
    // Create elements.
    s = StructNew();
    s.array = ArrayNew(2);

    // Assign simple values to original top-level structure fields.
    s.number = 99;
    s.string = "hello tommy";

    // Assign values to original top-level array.
    s.array[1][1] = "one one";
    s.array[1][2] = "one two";
</cfscript>

<!--- Output original structure --->
<hr>
<b>Original Values</b><br>
<cfoutput>
    // Simple values <br>
    s.number = #s.number#<br>
    s.string = #s.string#<br>
    // Array value <br>
    s.array[1][1] = #s.array[1][1]#<br>
    s.array[1][2] = #s.array[1][2]#<br>
</cfoutput>

// Copy this structure to a new structure. <br>
<cfset copied = StructCopy(s)>

<cfscript>
    // Change the values of the original structure. <br>
    s.number = 100;
    s.string = "hello tommy (modified)";
    s.array[1][1] = "one one (modified)";
    s.array[1][2] = "one two (modified)";
</cfscript>

<!--- Output original structure --->
<hr>
<b>Modified Original Values</b><br>
<cfoutput>
    // Simple values <br>
    s.number = #s.number#<br>
    s.string = #s.string#<br>
    // Array value <br>
    s.array[1][1] = #s.array[1][1]#<br>
    s.array[1][2] = #s.array[1][2]#<br>
</cfoutput>

// Copy this structure to a new structure. <br>
<cfset copied = StructCopy(s)>

<cfscript>
    // Copy this structure to a new structure. <br>
    copied.number = #copied.number#<br>
    copied.string = #copied.string#<br>
    copied.array[1][1] = #copied.array[1][1]#<br>
    copied.array[1][2] = #copied.array[1][2]#<br>
</cfscript>
copied.string = #copied.string#<br>
// Array value <br>
copied.array[1][1] = #copied.array[1][1]#<br>
copied.array[1][2] = #copied.array[1][2]#<br>
</cfoutput>

// This script creates a structure that StructCopy copies by reference.
<cfscript>
// Create elements.
s = StructNew();
s.nested = StructNew();
s.nested.array = ArrayNew(2);
// Assign simple values to nested structure fields.
s.nested.number = 99;
s.nested.string = "hello tommy";
// Assign values to nested array.
s.nested.array[1][1] = "one one";
s.nested.array[1][2] = "one two";
</cfscript>

<!--- Output original structure --->
<hr>
<b>Original Values</b><br>
<cfoutput>
// Simple values <br>
s.nested.number = #s.nested.number#<br>
s.nested.string = #s.nested.string#<br>
// Array values <br>
s.nested.array[1][1] = #s.nested.array[1][1]#<br>
s.nested.array[1][2] = #s.nested.array[1][2]#<br>
</cfoutput>

// Use StructCopy to copy this structure to a new structure. <br>
<cfset copied = StructCopy(s)>  
<!-- Use Duplicate to clone this structure to a new structure. <br>
<cfset duplicated = Duplicate(s)>  

<cfscript>
// Change the values of the original structure.
s.nested.number = 100;
s.nested.string = "hello tommy (modified)";
s.nested.array[1][1] = "one one (modified)";
s.nested.array[1][2] = "one two (modified)";
</cfscript>

<!--- Output modified original structure --->
<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>

<br>
<b>Copied structure values should reflect changes to original.</b><br>
<cfoutput>
// Simple values <br>
</cfoutput>

StructCopy  695
copied.nested.number = #copied.nested.number#<br>
copied.nested.string = #copied.nested.string#<br>
// Array values <br>
copied.nested.array[1][1] = #copied.nested.array[1][1]#<br>
copied.nested.array[1][2] = #copied.nested.array[1][2]#<br>
</cfoutput>

<hr>

<b>Duplicated structure values should remain unchanged.</b><br>
<cfoutput>
// Simple values <br>
duplicated.nested.number = #duplicated.nested.number#<br>
duplicated.nested.string = #duplicated.nested.string#<br>
// Array value <br>
duplicated.nested.array[1][1] = #duplicated.nested.array[1][1]#<br>
duplicated.nested.array[1][2] = #duplicated.nested.array[1][2]#<br>
</cfoutput>
StructCount

Description
Counts the keys in a structure.

Returns
A number; if structure does not exist, throws an exception.

Category
Structure functions

Function syntax
StructCount(structure)

See also
Structure functions

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 = "cfsnippets">
        INSERT INTO Employees
        (FirstName, LastName, Email, Phone, Department)
        VALUES
        <cfoutput>
        ("#StructFind(attributes.EMPINFO, "firstname")#",
        "#StructFind(attributes.EMPINFO, "lastname")#",
        "#StructFind(attributes.EMPINFO, "email")#",
        "#StructFind(attributes.EMPINFO, "phone")#",
        "#StructFind(attributes.EMPINFO, "department")#"
        )
        </cfoutput>
      </cfquery>
      <cfoutput><hr>Employee Add Complete</cfoutput>
      <p>#StructCount(attributes.EMPINFO)# columns added.</cfoutput>
    </cfif>
  </cfcase>
</cfswitch> --->

StructCount
StructDelete

Description
Removes an element from a structure.

Returns
Boolean value. The value depends on the `indicatenotexisting` parameter value.

Category
Structure functions

Function syntax
`StructDelete(structure, key [, indicatenotexisting])`

See also
Structure functions

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: return Yes if <code>key</code> exists; No if it does not.</td>
</tr>
<tr>
<td></td>
<td>• False: return Yes regardless of whether <code>key</code> exists. Default.</td>
</tr>
</tbody>
</table>

Example

```<h3>StructDelete Function</h3>
<!--- Delete the surrounding comments to make this page work --->
<p>This example uses the StructInsert and StructDelete functions.
<!--- Establish params for first time through --->
<cfparam name = "firstname" default = "Mary">
<cfparam name = "lastname" default = "Sante">
<cfparam name = "email" default = "msante@allaire.com">
<cfparam name = "phone" default = "777-777-7777">
<cfparam name = "department" default = "Documentation">

<cfif IsDefined("FORM.Delete")>
    <cfoutput>
        Field to be deleted: #form.field#
    </cfoutput>
</cfif>

<CFScript>
employee = StructNew();
StructInsert(employee, "firstname", firstname);
StructInsert(employee, "lastname", lastname);
StructInsert(employee, "email", email);
StructInsert(employee, "phone", phone);
StructInsert(employee, "department", department);
</CFScript>

Before deletion, employee structure looks like this:
<cfdump var="employee"/>
```
<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>
<br><br>
<form method="post" action="#CGI.Script_Name#">
<p>Select the field to be deleted:&nbsp;
<select name = "field">
<option value = "firstname">first name
<option value = "lastname">last name
<option value = "email">email
<option value = "phone">phone
<option value = "department">department
</select>
<input type = "submit" name = "Delete" value = "Delete">
</form>
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
Structure functions

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
<!---- 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.</p>
<!----
<cfswitch expression="#ThisTag.ExecutionMode#">
  <cfcase value="start">
    <cfif StructIsEmpty(attributes.EMPINFO)>
      <cfoutput>Error. No employee data was passed.</cfoutput>
      <cfexit method = "ExitTag">
    <cfelseif>
      <cfquery name = "AddEmployee" datasource = "cfsnippets">
      INSERT INTO Employees (FirstName, LastName, Email, Phone, Department)
      VALUES
      <cfoutput>
      ('#StructFind(attributes.EMPINFO, "firstname")#',
      '#StructFind(attributes.EMPINFO, "lastname")#',
      '#StructFind(attributes.EMPINFO, "email")#',
      '#StructFind(attributes.EMPINFO, "phone")#',
      '#StructFind(attributes.EMPINFO, "department")#')
      </cfoutput>
    </cfquery>
  </cfif>
</cfswitch>
</cfoutput><hr>Employee Add Complete</cfoutput>
</cfcase>
</cfswitch> --->

700 Chapter 3: ColdFusion Functions
**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**
```plaintext
StructFindKey(top, value, scope)
```

**See also**
Structure functions

**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 attribute 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**
```plaintext
<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
Structure functions

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 attribute 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

Structure functions

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>pathDesired</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

<!--- GetStruct() test --->
<cfset test = StructGet( "dog.myscope.test" )>
<cfset test.foo = 1>
<cfif NOT IsDefined("dog")>
    Dog is not defined<br>
</cfif>
<cfif NOT IsDefined("dog.myscope")>
    Dog.Myscope is not 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>
<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
Structure functions

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. Default: False.</td>
</tr>
</tbody>
</table>

Usage
A structure’s keys are unordered.

Example

```cfscript
employee = StructNew();
StructInsert(employee, "firstname", FORM.firstname);
StructInsert(employee, "lastname", FORM.lastname);
StructInsert(employee, "email", FORM.email);
StructInsert(employee, "phone", FORM.phone);
StructInsert(employee, "department", FORM.department);
</cfscript>
```
First name is #StructFind(employee, "firstname")#
Last name is #StructFind(employee, "lastname")#
Phone is #StructFind(employee, "phone")#
Department is #StructFind(employee, "department")#

<!---- Call the custom tag that adds employees --->
<CF_ADDEMPLOYEE EMPINFO = "#employee#">
</cif>

<form action = "structinsert.cfm">
 First Name:&nbsp;
<input name = "firstname" type = "text" hspace = "30" maxlength = "30">
 Last Name:&nbsp;
<input name = "lastname" type = "text" hspace = "30" maxlength = "30">
 EMail:&nbsp;
<input name = "email" type = "text" hspace = "30" maxlength = "30">
 Phone:&nbsp;
<input name = "phone" type = "text" hspace = "20" maxlength = "20">
 Department:&nbsp;
<input name = "department" type = "text" hspace = "30" maxlength = "30">
<input type = "submit" value = "OK">
</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**
Structure functions

**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**

```cftmpl
<!--- This example illustrates use of StructIsEmpty. --->
<cfswitch expression = "#ThisTag.ExecutionMode#">
  <cfcase value = "start">
    <cfif StructIsEmpty(attributes.EMPINFO)>
      <cfoutput>Error. No employee data was passed.</cfoutput>
      <cfexit method = "ExitTag">
    </cfif>
    <cfelse>
      <!--- Add the employee; In UNIX, you must also add the Emp_ID --->
      <cfquery name = "AddEmployee" datasource = "cfsnippets">
        INSERT INTO Employees
        (FirstName, LastName, Email, Phone, Department)
        VALUES
        ('
        'StructFind(attributes.EMPINFO, "firstname")',
        'StructFind(attributes.EMPINFO, "lastname")',
        'StructFind(attributes.EMPINFO, "email")',
        'StructFind(attributes.EMPINFO, "phone")',
        'StructFind(attributes.EMPINFO, "department")'
        )
      </cfquery>
      <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
Structure functions

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
<!--- 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>
   </cfif>
   <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>
<form action = "structkeyarray.cfm">
<table cellspacing = "2" cellpadding = "2" border = "0">
<tr>
   <td>First Name:</td>
   <td><input name = "firstname" type = "text"</td>
</tr>
</table>
</form>
<table>
<thead>
<tr>
<th>Field</th>
<th>Input Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>First Name</td>
<td>&lt;input type=&quot;text&quot; name=&quot;firstname&quot; value=&quot;&quot; hspace=&quot;30&quot; maxlength=&quot;30&quot;&gt;&lt;/input&gt;</td>
</tr>
<tr>
<td>Last Name</td>
<td>&lt;input type=&quot;text&quot; name=&quot;lastname&quot; value=&quot;&quot; hspace=&quot;30&quot; maxlength=&quot;30&quot;&gt;&lt;/input&gt;</td>
</tr>
<tr>
<td>Email</td>
<td>&lt;input type=&quot;text&quot; name=&quot;email&quot; value=&quot;&quot; hspace=&quot;30&quot; maxlength=&quot;30&quot;&gt;&lt;/input&gt;</td>
</tr>
<tr>
<td>Phone</td>
<td>&lt;input type=&quot;text&quot; name=&quot;phone&quot; value=&quot;&quot; hspace=&quot;20&quot; maxlength=&quot;20&quot;&gt;&lt;/input&gt;</td>
</tr>
<tr>
<td>Company</td>
<td>&lt;input type=&quot;text&quot; name=&quot;company&quot; value=&quot;&quot; hspace=&quot;30&quot; maxlength=&quot;30&quot;&gt;&lt;/input&gt;</td>
</tr>
</tbody>
</table>

After you submit the FORM, scroll down to see the array.

```html
<cfif NOT StructISEmpty(employee)>
  <hr size="2" color="#0000A0">
  <cfset keysToStruct = StructKeyArray(employee)>
  <cfloop index="i" from="1" to="#ArrayLen(keysToStruct)#">
    <p><cfoutput>Key#i# is #keysToStruct[i]#</cfoutput></p>
    <p><cfoutput>Value#i# is #employee[keysToStruct[i]]#</cfoutput></p>
  </cfloop>
</cfif>
```
StructKeyExists

Description
Determine 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
StructKeyExists(structure, "key")

See also
Structure functions

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:

```
<cfif IsDefined("Form.JediMaster")>
<cfif StructKeyExists(Form."JediMaster")>
```

A structure's keys are unordered.

Example
<!--- This example shows the use of StructKeyExists. --->
<p>This file is similar to addemployee.cfm, which is called by StructNew, StructClear, and StructDelete. To test, copy the &LT;CFELSEif&GT; statement to the appropriate place in addemployee.cfm. It is a custom tag to add employees. Employee information is passed through the employee structure (the EMPINFO attribute). In UNIX, you must also add the Emp_ID.

<cfswitch expression = "#ThisTag.ExecutionMode#">
<cfcase value = "start">
<cfoutput>Error. No employee data was passed.</cfoutput>
<cfexit method = "ExitTag">
<cfelseif NOT StructKeyExists(attributes.EMPINFO, "department")>
<cfscript>StructUpdate(attributes.EMPINFO, "department", "Unassigned");
</cfscript>
<cfexit method = "ExitTag">
<cfelse>

```
StructKeyList

Description
Extracts keys from a ColdFusion structure.

Returns
A list of keys; if structure does not exist, ColdFusion throws an exception.

Category
Structure functions

Function syntax
StructKeyList(structure [, delimiter])

See also
Structure functions

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. Default: 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>
  </cfif>
  <cfelseif Form.Submit is "Clear">
    <cfset rc = StructClear(employee)>
  </cfelseif>
</cfif>
</cfif>
<html>
<head>
<title>StructKeyList Function</title>
</head>
<body>
<h3>StructKeyList Function</h3>
<h3>Listing the Keys in the Employees Structure</h3>
<p>This example uses StructNew function to create structure "employee" that supplies employee information. The fields are filled with the contents of the following form.</p>
After you enter employee information into structure, example uses StructureKeyList 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.

| First Name: | <input name="firstname" type="text" value="" hspace="30" maxlength="30"> |
| Last Name: | <input name="lastname" type="text" value="" hspace="30" maxlength="30"> |
| EMail | <input name="email" type="text" value="" hspace="30" maxlength="30"> |
| Phone: | <input name="phone" type="text" value="" hspace="20" maxlength="20"> |
| Company: | <input name="company" type="text" value="" hspace="30" maxlength="30"> |
| | <input type="submit" name="submit" value="OK"> |

Here are the keys to the structure:

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
Structure functions

Parameters
None

Example
<!--- Shows StructNew. Calls CF_ADDEMPLOYEE, which uses the |
      addemployee.cfm file to add employee record to database. --->
<h1>Add New Employees</h1>
<cfparam name = "FORM.firstname" default = "">
<cfparam name = "FORM.lastname" default = "">
<cfparam name = "FORM.email" default = "">
<cfparam name = "FORM.phone" default = "">
<cfparam name = "FORM.department" default = "">
<cfif FORM.firstname EQ ">
<p>Please fill out the form.
<cfelse>
<cfoutput>
<cfscript>
employee = StructNew();
StructInsert(employee, "firstname", FORM.firstname);
StructInsert(employee, "lastname", FORM.lastname);
StructInsert(employee, "email", FORM.email);
StructInsert(employee, "phone", FORM.phone);
StructInsert(employee, "department", FORM.department);
</cfscript>
<p>First name is #StructFind(employee, "firstname")#
<p>Last name is #StructFind(employee, "lastname")#
<p>EMail is #StructFind(employee, "email")#
<p>Phone is #StructFind(employee, "phone")#
<p>Department is #StructFind(employee, "department")#
</cfoutput>
<!--- Call the custom tag that adds employees --->
<CF_ADDEMPLOYEE EMPINFO = "#employee#">
</cfif>
StructSort

Description
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
StructSort(base, sortType, sortOrder, pathToSubElement)

See also
Structure functions

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 lower-case letters precede the first upper-case 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. Default: 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
```cfs
<cfscript>
salaries = StructNew();
employees = StructNew();
departments = StructNew();
for ( i=1; i lt 6; i=i+1 )
{
    salary = 120000 - i*10000 ;
salaries["employee#i#"] = salary ;
    employee = StructNew();
    employee["salary"] = salary ;
    // employee.salary = salary ;
    employees["employee#i#"] = employee ;
}</cfscript>
```
StructSort 715

departments["department#{i#}"] = StructNew();
departments["department#{i#}"].boss = employee;
}
</cfscript>

<cfoutput>
<p>list of employees based on the salary (text search):  <br>
1) #ArrayToList( StructSort( salaries ) )#<br>
2) #ArrayToList( StructSort( salaries, "text", "ASC" ) )#<br>
3) #ArrayToList( StructSort( salaries, "textnocase", "ASC" ) )#<br>
4) #ArrayToList( StructSort( salaries, "text", "DESC" ) )#<br>
<p>list of employees based on the salary (numeric search):  <br>
5) #ArrayToList( StructSort( salaries, "numeric", "ASC" ) )#<br>
6) #ArrayToList( StructSort( salaries, "numeric", "DESC" ) )#<br>
<p>list of employees based on the salary (subfield search):  <br>
7) #ArrayToList( StructSort( employees, "numeric", "DESC", "salary" ) )#<br>
8) #ArrayToList( StructSort( employees, "text", "ASC", "salary" ) )#<br>
<p>list of departments based on the salary (sub-sub-field search):  <br>
9) #ArrayToList( StructSort( departments, "text", "ASC", "boss.salary" ) )#<br></cfoutput>

<!--- add an invalid element and test that it throws an error --->

<p><cfset employees[ "employee#4#" ] = StructNew()></p>
<cftry>
  <cfset temp = StructSort( employees, "text", "ASC", "salary" )>
  <cfoutput>We have a problem - this was supposed to throw an exception!</cfoutput>
</cftry>
<cfcatch type="any">
  <cfoutput>
    ERROR: <b>This error was expected!</b><br>
    #cfcatch.message# - #cfcatch.detail#<br>
  </cfoutput>
</cfcatch>
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
Structure functions

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. --->
<p>This file is similar to addemployee.cfm, which is called by StructNew, StructClear, and StructDelete. To test this file, copy the &LT;CFELSEIF&GT; statement to the appropriate place in addemployee.cfm. It is an example of a custom tag used to add employees. Employee information is passed through the employee structure (the EMPINFO attribute). In UNIX, you must also add the Emp_ID.\n
<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>

---
716 Chapter 3: ColdFusion Functions
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
\[ \text{Tan}(number) \]

See also
Atn, Cos, ACos, 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 \( \frac{\pi}{180} \). To convert radians to degrees, multiply radians by \( \frac{180}{\pi} \).

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 0.0000000000001. To test for an infinite value, check whether the value is more than 1E16.

Example

<!--- 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)#>
<!--- 0.0000000000001 is the function's precision limit. If absolute value of returned value is less, set result to 0 --->
<cfif Abs(tanValue) LT 0.0000000000001>
<cfset tanValue=0>
</cfif>
<cfoutput>
Tan(#FORM.tanNum#) = #tanValue#<br><br>
</cfoutput>
<cfelse>
<!--- If input is not a number, show an error message --->
<h4>You must enter a numeric angle in degrees.</h4>
</cfif>
</cfif>
<form action="#CGI.script_name#" method="post">
Enter an angle in degrees to get its tangent:
**TimeFormat**

**Description**
Formats a time value using US English time formatting conventions.

**Returns**
A custom-formatted time value. If no mask is specified, returns a time value using the \texttt{hh:mm tt} format. For international time formatting, see \texttt{LSTimeFormat}.

**Category**
Date and time functions, Display and formatting functions

**Function syntax**
\texttt{TimeFormat(time [, mask])}

**See also**
\texttt{CreateTime, Now, ParseDateTime, LSTimeFormat, DateFormat}

**History**
ColdFusion MX 6.1: Added the mask character \texttt{L} or \texttt{l} to represent milliseconds.

ColdFusion MX:
- Changed the way extra characters are processed: this function processes extra characters within the \texttt{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, \texttt{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, \texttt{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 \texttt{mask} attribute options: \texttt{short}, \texttt{medium}, \texttt{long}, and \texttt{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>
<tr>
<td>mask</td>
<td>Masking characters that determine the format:</td>
</tr>
<tr>
<td></td>
<td>• h: Hours; no leading zero for single-digit hours (12-hour clock)</td>
</tr>
<tr>
<td></td>
<td>• hh: Hours; leading zero for single-digit hours (12-hour clock)</td>
</tr>
<tr>
<td></td>
<td>• h: Hours; no leading zero for single-digit hours (24-hour clock)</td>
</tr>
<tr>
<td></td>
<td>• hh: Hours; leading zero for single-digit hours (24-hour clock)</td>
</tr>
<tr>
<td></td>
<td>• m: Minutes; no leading zero for single-digit minutes</td>
</tr>
<tr>
<td></td>
<td>• mm: Minutes; a leading zero for single-digit minutes</td>
</tr>
<tr>
<td></td>
<td>• s: Seconds; no leading zero for single-digit seconds</td>
</tr>
<tr>
<td></td>
<td>• ss: Seconds; leading zero for single-digit seconds</td>
</tr>
<tr>
<td></td>
<td>• t: Milliseconds</td>
</tr>
<tr>
<td></td>
<td>• t: One-character time marker string, such as A or P</td>
</tr>
<tr>
<td></td>
<td>• tt: Multiple-character time marker string, such as AM or PM</td>
</tr>
<tr>
<td></td>
<td>• short: equivalent to h:mm tt</td>
</tr>
<tr>
<td></td>
<td>• medium: equivalent to h:mm:ss tt</td>
</tr>
<tr>
<td></td>
<td>• long: medium followed by three-letter time zone; as in, 2:34:55 PM EST</td>
</tr>
<tr>
<td></td>
<td>• full: same as long</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.

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

```coldfusion
<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>
<ul>
<li>#TimeFormat(todayDate)#
<li>#TimeFormat(todayDate, "hh:mm:ss")#
<li>#TimeFormat(todayDate, "hh:mm:ss tt")#
<li>#TimeFormat(todayDate, "hh:mm:ss tt")#
</ul>
</cfoutput>
</body>
```
**ToBase64**

**Description**
Calculates the Base64 representation of a string or binary object. The Base64 format uses printable characters, allowing binary data to be sent in forms and e-mail, and stored in a database or file.

The Base64 representation of a string or binary object.

**Category**
Conversion functions, Other functions, String functions

**Function syntax**

```
ToBase64(string or binary_object[, encoding])
```

**See also**
- `cffile` for information about loading and reading binary data
- `cfwddx` for information about serializing and deserializing binary data
- `IsBinary` and `ToBinary` for checking for binary data and converting a Base64 object to binary format

**History**
ColdFusion MX: Added the `encoding` attribute.

**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`  
  - `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.  
  Default: the encoding of the page on which the function is called. See `cfcontent` on page 87.  
  For a binary object, this parameter is ignored. |

**Usage**
Base64 provides 6-bit encoding of data. The encoding process represents 24-bit groups of input bits as output strings of 4 encoded ASCII characters. Binary objects and, in some cases, 8-bit characters, cannot be transported over many internet protocols, such as HTTP and SMTP. Using Base64 safely converts the data into a format that is safe over any internet protocol.
Base64 lets you store binary objects in a database.

**Note:** To reverse Base64 encoding of a string, you can convert it to a binary object, then convert the binary object to a string, using the `toString` function.

**Example**

```coldfusion
<h3>ToBase64 Example</h3>
<!--- Initialize data. ---->
<cfset charData = "">
<!--- Create string of ASCII characters (32-255); concatenate them --->
<cfloop index = "data" from = "32" to = "255">
    <cfset ch = chr(data)>
    <cfset charData = charData & ch>
</cfloop>
<p>
The following string is the concatenation of all characters (32 to 255) from the ASCII table.<br>
<cfoutput>#charData#</cfoutput></p>
<!--- Create a Base64 representation of this string. ---->
<cfset data64 = toBase64(charData)>

<!--- Convert string to binary. ------->
<cfset binaryData = toBinary(data64)>
<!--- Convert binary back to Base64. ---->
<cfset another64 = toBase64(binaryData)>
<!--- Compare another64 with data64 to ensure that they are equal. ---->
<cfif another64 eq data64>
    <h3>Base64 representations are identical.</h3>
<cfelse>
    <h3>Conversion error.</h3>
</cfif>
```
ToBinary

Description
Calculates the binary representation of Base64-encoded data.

Returns
The binary representation of Base64-encoded data.

Category
Conversion functions, Other functions, String functions

Function syntax
ToBinary(string_in_Base64 or binary_value)

See also
• cffile for information about loading and reading binary data
• cfwddx for information about serializing and deserializing binary data
• IsBinary and ToBase64 for checking format and converting to Base64
• Len for determining the length of a binary object

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| string_in_Base64 or binary_value | A string or a variable that contains one:
  • In Base64 format to convert to binary
  • In binary format to test whether it is valid |

Usage
Base64 provides 6-bit encoding of 8-bit ASCII characters. From Base64 data, you can recreate the binary object that it represents, such as a GIF, JPG, or executable file.

Example

```<h3>ToBinary Example</h3>
<!---- Initialize data. ---->
<cfset charData = ">
<!---- Create a string of ASCII characters (32-255); concatenate them. ---->
<cfloop index = "data" from = "32" to = "255">
  <cfset ch = chr(data)>
  <cfset charData = charData & ch>
</cfloop>
<p>The following string is the concatenation of all characters (32 to 255) from the ASCII table.<br>
<cfoutput>\#charData\#</cfoutput></p>
<!----- Create a Base64 representation of this string. ----->
<cfset data64 = toBase64(charData)>

<!---- Convert string to binary. ---->
<cfset binaryData = toBinary(data64)>
<!---- Convert binary back to Base64. ---->
<cfset another64 = toBase64(binaryData)>
<!---- Compare another64 with data64 to ensure that they are equal. ---->
<cfif another64 eq data64>
  <h3>Base64 representation of binary data is identical to the Base64 representation of string data.</h3>
```
<cfelse>
  <h3>Conversion error.</h3>
</cfif>
**ToString**

**Description**
Converts a value to a string.

**Returns**
A string.

**Category**
Conversion functions, Other functions, String functions

**Function syntax**
```
ToString(any_value[, encoding])
```

**See also**
ToBase64, ToBinary

**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` attribute.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>any_value</td>
<td>Value to convert to a string</td>
</tr>
</tbody>
</table>
| encoding  | The character encoding (character set) of the string. The following list includes commonly used values:  
  - utf-8  
  - iso-8859-1  
  - windows-1252  
  - us-ascii  
  - shift_jis  
  - iso-2022-ja  
  - euc-ja  
  - euc-ko  
  - 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). Default: the encoding of the page on which the function is called. See cfcontent on page 87. |

**Usage**
This function can convert simple values and binary values that do not contain Byte zero. If this function cannot convert a value, it throws an exception. This function can convert an XML document object to a string representation.

**Note:** You can use this function to reverse Base64 encoding of a string. Convert the Base64 encoded object to a binary object, then use this function to convert the binary object to a string.
Example

<h3>ToString Example</h3>

<!-- Initialize data. ---->
<cfset charData = ""/>

<!-- Create string of ASCII characters (32-255) and concatenate them. ---->
<cfloop index = "data" from = "32" to = "255">
  <cfset ch = chr(data)>
  <cfset charData = charData & ch>
</cfloop>

The following string is the concatenation of characters (32 to 255)
from the ASCII table.
<cfoutput>#charData#</cfoutput>

<!-- Create a Base64 representation of this string. ---->
<cfset data64 = toBase64(#charData#)>

The following string is the Base64 representation of the string.
<cfoutput>#data64#</cfoutput>

<!-- Create a binary representation of Base64 data. ---->
<cfset dataBinary = toBinary(data64)>

<!-- Create the string representation of the binary data. ---->
<cfset dataString = ToString(dataBinary)>

The following is the string representation of the binary data.
<cfoutput>#dataString#</cfoutput>
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></cfoutput>
<form action = "trim.cfm">
  Type in some text, and it will be modified by trim to remove leading spaces from the left and right
  <p><input type = "Text" name = "myText" value = " TEST ">
  <p><input type = "Submit" name = "">
</form>```
UCase

Description
Converts the alphabetic characters in a string to uppercase.

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

```cftmpl
<h3>UCase Example</h3>
<cfif IsDefined("FORM.sampleText")>
  <cfif FORM.sampleText is not ">"
    <p>Your text. <cfoutput>#FORM.sampleText#</cfoutput>, returned in uppercase is <cfoutput>UCase(FORM.sampleText)</cfoutput>.
  </cfelse>
  <p>Please enter some text.
</cfif>
</cfif>
<form action = "ucase.cfm">
<p>Enter your sample text, and press "submit" to see the text returned in uppercase:
<p><input type = "Text" name = "SampleText" value = "sample">
<input type = "Submit" name = "" value = "submit">
</form>
```
**URLDecode**

**Description**
Decodes a URL-encoded string.

**Returns**
A copy of a string, decoded.

**Category**
Conversion functions, Other functions, String functions

**Function syntax**
```
URLDecode(urlEncodedString[, charset])
```

**See also**
URLEncodedFormat

**History**
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>urlEncodedString</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). Default: the character encoding of the URL scope. See `SetEncoding` on page 674.

**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.

```cfs
<cfscript>
  // Build string
  s = "";
  for (c = 1; c lte 256; c = c + 1)
  {
    s = s & chr(c);
  }
  // Encode string and display result
  enc = URLEncodedFormat(s);
  WriteOutput("Encoded string is: '#enc#'.<br>");
  // Decode and compare result with original
  dec = URLDecode(enc);
  if (dec neq s)
  {
    WriteOutput("Decoded is not the same as encoded.";
  }
  else
  {
    WriteOutput("All's quiet on the Western front.";
  }
</cfscript>
```
URLEncodedFormat

Description
Generates a URL-encoded string. For example, it replaces spaces with %20, and non-alphanumeric characters with equivalent hexadecimal escape sequences. Passes arbitrary strings within a URL (ColdFusion automatically decodes URL parameters that are passed to a page).

Returns
A copy of a string, URL-encoded.

Category
Conversion functions, Other functions, String functions

Function syntax
URLEncodedFormat(string [, charset])

See also
URLEncode

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>. Default: the character encoding of the response. See cfcontent on page 87.</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 = "urlencodedformat.cfm?myExample=<cfoutput>#URLEncodedFormat(s)#
</cfoutput>">Click me</A>
URLSessionFormat

Description
Depending on whether a client computer accepts cookies, this function does the following:

- If the client does not accept cookies: automatically appends all required client identification information to a URL
- If the client accepts cookies: does not append information

This function automatically determines which identifiers are required, and sends only the required information. It provides a more secure and robust method for supporting client identification than manually encoding the information in each URL, because it sends only required information, when it is required, and it is easier to code.

Returns
A URL; if cookies are disabled for the browser, client and session data are appended.

Category
Other functions

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")#>

Parameter Description
request_URL URL of a ColdFusion page
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

```html
<h3>Val Example</h3>
<cfif IsDefined("FORM.theTestValue")>
  <cfif Val(FORM.theTestValue) is not 0>
    <h3>The string <cfoutput>#{FORM.theTestValue}#</cfoutput> can be converted to a number:
      <cfoutput>#{Val(FORM.theTestValue)}#</cfoutput></h3>
  </cfif>
</cfif>
<form action = "val.cfm">
  <p>Enter a string, and determine whether its beginning can be evaluated to a numeric value.</p>
  <input type = "Text" name = "TheTestValue" value = "123Boy">
  <input type = "Submit" value = "Is the beginning numeric?" name = "">
</form>
```
ValueList

Description
Inserts a delimiter between each value in an executed query. ColdFusion does not evaluate the arguments.

Returns
A delimited list of the values of each record returned from an executed query.

Category
Other 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. Default: comma (,).</td>
</tr>
</tbody>
</table>

Example

<h3>ValueList Example</h3>

<!--- use the contents of a query to create another dynamically --->
<cfquery name = "GetDepartments" datasource = "cfsnippets">
    SELECT Dept_ID FROM Departments
    WHERE Dept_ID IN ('BIOL')
</cfquery>

<cfquery name = "GetCourseList" datasource = "cfsnippets">
    SELECT * FROM CourseList
    WHERE Dept_ID IN ('#ValueList(GetDepartments.Dept_ID)#')
</cfquery>

<cfoutput QUERY = "GetCourseList" >
<pre>Course_ID##Dept_ID##CorNumber##CorName</pre>
</cfoutput>
**Week**

**Description**
From a date/time object, determines the week number within the year.

**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**
```
<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>
```
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. Default: 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 wraptext 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

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

```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**
Extensibility functions, XML functions

**Function syntax**
XmlChildPos(elem, childName, N)

**See also**
cfxml, IsXmlDoc, XmlChildPos, XmlFormat, XmlNew, XmlParse, XmlSearch, XmlTransform

**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</td>
</tr>
<tr>
<td>N</td>
<td>Index of XML child element for which to search</td>
</tr>
</tbody>
</table>

**Usage**
The returned index can be used in the ArrayInsertAt and ArrayDeleteAt functions.

**Example**
This example searches the XML document object mydoc.employee.XmlChildren for the mydoc.employee.name[2] element:

```coldfusion
XmlChildPos(mydoc.employee. "name", 2)
```
**XmlElemNew**

**Description**
Creates an XML document object element.

**Returns**
An XML document object element.

**Category**
Extensibility functions, XML functions

**Function syntax**
```
XmlElemNew(xmlObj, childName)
```

**See also**
cfxml, IsXmlDoc, XmlChildPos, XmlFormat, XmlNew, XmlParse, XmlSearch, XmlTransform

**History**
ColdFusion MX: Added this function.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>xmlObj</td>
<td>The name of an XML object. An XML document or an element.</td>
</tr>
<tr>
<td>childName</td>
<td>The name of the element to create. This element becomes a child element of xmlObj in the tree.</td>
</tr>
</tbody>
</table>

**Example**
The following example creates and displays a ColdFusion document object. For more information on this example, see Chapter 31, “Using XML and WDDX,” in Developing ColdFusion MX Applications.

```cfc
<cfset testVar = True>
<cfscript>
MyDoc = XmlNew();
MyDoc.xmlRoot = XmlElemNew(MyDoc, "MyRoot");
if (testVar IS TRUE)
    MyDoc.MyRoot.XmlText = "The value of testVar is True.";
else
    MyDoc.MyRoot.XmlText = "The value of testVar is False.";
for (i = 1; i LTE 4; i = i + 1)
    MyDoc.MyRoot.XmlChildren[i] = XmlElemNew(MyDoc, "childNode");
    MyDoc.MyRoot.XmlChildren[i].XmlText = "This is Child node " & i & ";
</cfscript>
<cfdump var=#MyDoc#>
```
XmlFormat

Description
Escapes special XML characters in a string, so that the string is safe to use with XML.

Returns
A copy of string that is safe to use with XML.

Category
Extensibility functions, String functions, XML functions

Function syntax
XmlFormat(string)

See also
cfxml, IsXmlDoc, XmlChildPos, XmlChildPos, XmlNew, XmlParse, XmlSearch, XmlTransform

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
The characters that this function escapes include the following:
• Greater than symbol (>)
• Less than symbol (<)
• Single quotation mark (')
• Double quotation mark (")
• Ampersand symbol (&)

Example
<h3>XMLFormat</h3>
<p>This example shows how XMLFormat is used to escape special XML characters and make the use of XML with ColdFusion easy.</p>
<xml version = "1.0">
  <cfoutput>
    <someXML>
      <someElement someAttribute = "#XMLFormat("a quoted value")#>
        #XMLFormat("Body of element to be passed here.")#
      </someElement>
    </someXML>
  </cfoutput>
</xml>
**XmlNew**

**Description**
Creates an XML document object.

**Returns**
An empty XML document object.

**Category**
Extensibility functions, XML functions

**Function syntax**

```
XmlNew([caseSensitive])
```

**See also**
cfxml, IsXmlDoc, XmlChildPos, XmlChildPos, XmlFormat, XmlParse, XmlSearch, XmlTransform

**History**
ColdFusion MX: Added this function.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| caseSensitive | Determines how ColdFusion processes the case of XML document object component identifiers  
  • yes: maintains case  
  • no: ColdFusion ignores case. Default. |

**Usage**
An XML document object is represented in ColdFusion as a structure.

The `caseSensitive` attribute value determines whether identifiers whose characters are of varying case, but are otherwise the same, refer to different components. For example:

- If `caseSensitive = "no"`, the names `mydoc.employee.name[1]` and `mydoc.employee.NAME[1]` refer to the same element
- If `caseSensitive = "yes"`, these names refer to two distinct elements

The following example creates and displays a ColdFusion document object. For more information on this example, see Chapter 31, “Using XML and WDDX,” in Developing ColdFusion MX Applications.

**Example**

```cfscript
<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 an XML document that is represented as a string variable into an XML document object.

Returns
An XML document object.

Category
Conversion functions, Extensibility functions, XML functions

Function syntax
XmlParse(xmlString [, caseSensitive ])

See also
cfxml, IsXmlDoc, XmlChildPos, XmlChildPos, XmlFormat, XmlNew, XmlSearch, XmlTransform

History
ColdFusion MX: Added this function.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>xmlString</td>
<td>An XML document object string</td>
</tr>
</tbody>
</table>
| caseSensitive | • yes: maintains the case of document elements and attributes  
|               | • no. Default.                                  |

Usage

The caseSensitive attribute value determines whether identifiers whose characters are of varying case, but are otherwise the same, refer to different components. For example:

- If caseSensitive = "no", the names mydoc.employee.name[1] and mydoc.employee.NAME[1] refer to the same element
- If caseSensitive = "yes", these names refer to two distinct elements

If the XML document is represented by a string variable, use the XmlParse tag directly on the variable. For example, if your application uses cfhttp action="get" to get the XML document, use the following code to create the XML document object:

```cfset myXMLDocument = XmlParse(cfhttp.fileContent)```

If the XML document is in a file, use the cffile tag to convert the file to a CFML variable, then use the XmlParse tag on the resulting variable. For example, if the XML document is in the file C:\temp\myxmldoc.xml, use the following code to convert the file to an XML document object:

```<cffile action="read" file="C:\temp\myxmldoc.xml" variable="XMLFileText">
<cfset myXMLDocument=XmlParse(XMLFileText)>```

**Note:** If the file is not encoded with the ASCII or Latin-1 character encoding, use the cffile tag charset attribute to specify the file's character encoding. For example, if the file is encoded in UTF, specify charset="UTF-8".
XmlSearch

Description
Uses an XPath language expression to search an XML document that is represented as a string variable.

Returns
An array of XML object nodes that match the search criteria.

Category
Extensibility functions, XML functions

Function syntax
XmlSearch(xmlDoc, xPathString)

See also
cfxml, IsXmlDoc, XmlChildPos, XmlChildPos, XmlFormat, XmlNew, XmlParse, XmlTransform

History
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
XPath is specified by the World-Wide Web Consortium. For detailed information on XPath, 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 the employeesimple.xml file, and displays the names.

```cfml
<cffile action="read"
  file="C:\inetpub\wwwroot\examples\employeesimple.xml"
  variable="myxml">
<cfscript>
  myxmldoc = XmlParse(myxml);
  selectedElements = XmlSearch(myxmldoc, "/employee/name/last");
  for (i = 1; i LTE ArrayLen(selectedElements); i = i + 1)
    writeoutput(selectedElements[i].XmlText & "<br>");
</cfscript>
```
XmlTransform

**Description**
Applies an Extensible Stylesheet Language Transformation (XSLT) to an XML document object that is represented as a string variable. An XSLT converts an XML document to another format or representation by applying an Extensible Stylesheet Language (XSL) stylesheet to it. For more information, see Chapter 31, “Using XML and WDDX,” in Developing ColdFusion MX Applications.

**Returns**
A string; an XML document after the XSLT is applied

**Category**
Conversion functions, Extensibility functions, XML functions

**Function syntax**
XmlTransform(xmlString | xmlObj, xslString)

**See also**
cfxml, IsXmlDoc, XmlChildPos, XmlChildPos, XmlFormat, XmlNew, XmlParse, XmlSearch

**History**
ColdFusion MX: Added this function.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>xmlString</td>
<td>A string that represents the XML document, or a parsed object representation of it.</td>
</tr>
<tr>
<td>xslString</td>
<td>XSLT transformation to apply.</td>
</tr>
</tbody>
</table>
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

```
<h3>Year 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)#). <br>
    <cfif IsLeapYear(Year(yourDate))>
      This is a leap year
    <cfelse>This is not a leap year
    </cfif>
  </cfoutput>
</cfif>
```
**YesNoFormat**

**Description**
Evaluates a number or Boolean value.

**Returns**
Yes, for a non-zero value; No, otherwise.

**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 values as "No".

<cfoutput>
  <ul>
  <li>YesNoFormat(1):#YesNoFormat(1)#</li>
  <li>YesNoFormat(0):#YesNoFormat(0)#</li>
  <li>YesNoFormat("1123"):#YesNoFormat("1123")#</li>
  <li>YesNoFormat("No"):#YesNoFormat("No")#</li>
  <li>YesNoFormat(True):#YesNoFormat(True)#
  </ul>
<cfoutput>
```
This chapter describes the CFXAPI classes and members.

Contents

C++ class overview .......................................................... 750
Deprecated class members ................................................. 750
CCFXException class ........................................................... 751
CCFXQuery class ............................................................... 753
CCFXRequest class ............................................................. 757
CCFXStringSet class ......................................................... 766
C++ class overview

A list of CFXAPI classes and members follows.

<table>
<thead>
<tr>
<th>Class</th>
<th>Member</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::GetName</td>
</tr>
<tr>
<td></td>
<td>CCFXQuery::GetRowCount</td>
</tr>
<tr>
<td></td>
<td>CCFXQuery::SetData</td>
</tr>
<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>

Deprecated class members

The following CFXAPI classes and members are deprecated. They do not work, and might cause an error, in later releases.

<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 members

<table>
<thead>
<tr>
<th>Function</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 exception that occur during processing.

Example
This code block shows how GetError and GetDiagnostics work with ThrowException and ReThrowException.
// Write output back to the user here...
pRequest->Write( "Hello from CFX_FOO2!" );
pRequest->ThrowException( "User Error", "You goof'd..." );

// Output optional debug info
if ( pRequest->Debug() )
{
    pRequest->WriteDebug( "Debug info..." );
}

// Catch 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 members

<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:

```c
// 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**

```cpp
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:

```cpp
// 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**

```cpp
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:

```c
CCFXQuery* pQuery = pRequest->GetQuery();
pRequest->Write( "The query name is: " ) ;
pRequest->Write( pQuery->GetName() ) ;
```

**CCFXQuery::GetRowCount**

**Syntax**
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:

```c
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>iRow</td>
<td>Row of data element to set (1-based)</td>
</tr>
<tr>
<td>iColumn</td>
<td>Column of data element to set (1-based)</td>
</tr>
<tr>
<td>lpszData</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:

```cpp
// First row
int iRow;
iRow = pQuery->AddRow();
pQuery->SetData( iCity, iRow, "Minneapolis" );
pQuery->SetData( iState, iRow, "MN" );
pQuery->SetData( iZip, iRow, "55345" );

// Second row
iRow = pQuery->AddRow();
pQuery->SetData( iCity, iRow, "St. Paul" );
pQuery->SetData( iState, iRow, "MN" );
pQuery->SetData( iZip, iRow, "55105" );
```
**CCFXRequest class**

Abstract class that represents a request made to a ColdFusion Extension (CFX). An instance of this class is passed to the main function of your extension DLL. The class provides interfaces that can be used by the custom extension for the following actions:

- Reading and writing variables
- Returning output
- Creating and using queries
- Throwing exceptions

**Class members**

```cpp
virtual BOOL AttributeExists ( LPCSTR lpszName )
CCFXRequest::AttributeExists checks whether the attribute was passed to the tag.

virtual LPCSTR GetAttribute ( LPCSTR lpszName )
CCFXRequest::GetAttribute gets the value of the passed attribute.

virtual CCFXStringSet* GetAttributeList()
CCFXRequest::GetAttributeList gets an array of attribute names passed to the tag.

virtual CCFXQuery* GetQuery()
CCFXRequest::GetQuery gets the query that was passed to the tag.

virtual LPCSTR GetSetting( LPCSTR lpszSettingName )
CCFXRequest::GetSetting
This member is deprecated. It might not work, and might cause an error, in later releases.

virtual void Write( LPCSTR lpszOutput )
CCFXRequest::Write writes text output back to the user.

virtual void SetVariable( LPCSTR lpszName, LPCSTR lpszValue )
CCFXRequest::SetVariable sets a variable in the template that contains this tag.

virtual CCFXQuery* AddQuery( LPCSTR lpszName, CCFXStringSet* pColumns )
CCFXRequest::AddQuery adds a query to the template that contains this tag.

virtual BOOL Debug()
CCFXRequest::Debug checks whether the tag contains the DEBUG attribute.

virtual void WriteDebug( LPCSTR lpszOutput )
CCFXRequest::WriteDebug writes text output into the debug stream.

virtual CCFXStringSet* CreateStringSet()
CCFXRequest::CreateStringSet allocates and returns a CCFXStringSet instance.

virtual void ThrowException( LPCSTR lpszError, LPCSTR lpszDiagnostics )
CCFXRequest::ThrowException throws an exception and ends processing of this request.

virtual void ReThrowException ( CCFXException* e )
CCFXRequest::ReThrowException re-throws an exception that has been caught.
```
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;
pQuery->AddRow();
pQuery->setData(iRow, iFirstName, "John");
pQuery->setData(iRow, iLastName, "Smith");
iRow = pQuery->AddRow();
pQuery->setData(iRow, iFirstName, "Jane");
pQuery->setData(iRow, iLastName, "Doe");
```

virtual void SetCustomData( LPVOID lpvData ) CCFXRequest::SetCustomData sets custom (tag specific) data to carry with a request.

virtual LPVOID GetCustomData() CCFXRequest::GetCustomData gets custom (tag specific) data for a request.
CCFXRequest::AttributeExists

Syntax
BOOL CCFXRequest::AttributeExists(LPCSTR lpszName)

Description
Checks whether the attribute was passed to the tag. Returns True if the attribute is available; False, otherwise.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lpszName</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, and throws an exception if the attribute was not passed:

```c
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:
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:
LPCSTR lpszDestination = pRequest->GetAttribute("DESTINATION") ;
pRequest->Write( "The destination is: " ) ;
pRequest->Write( lpszDestination ) ;
CCFXRequest::GetAttributeList

Syntax

CCFXStringSet* CCFXRequest::GetAttributeList(void)

Description

Gets an array of attribute names passed to the tag. To get the value of one attribute, use CCFXRequest::GetAttribute.

Returns

Returns an object of class CCFXStringSet class that contains a list of attributes passed to the tag. The memory allocated for the returned string set is freed automatically by ColdFusion after the request is completed.

Example

The following example gets the list of attributes and iterates over the list, writing each attribute and its value back to the user.

LPCSTR lpszName, lpszValue;
CCFXStringSet* pAttribs = pRequest->GetAttributeList();
int nNumAttribs = pAttribs->GetCount();
for( int i=1; i<=nNumAttribs; i++ )
{
    lpszName = pAttribs->GetString(i);
    lpszValue = pRequest->GetAttribute(lpszName);
    pRequest->Write(lpszName);
    pRequest->Write(" = ");
    pRequest->Write(lpszValue);
    pRequest->Write("<BR>" );
}

CCFXRequest::GetCustomData

Syntax

LPVOID CCFXRequest::GetCustomData(void)

Description

Gets the custom (tag specific) data for the request. This member is typically used from within subroutines of a tag implementation to extract tag data from a request.

Returns

Returns a pointer to the custom data, or NULL if no custom data has been set during this request using CCFXRequest::SetCustomData.

Example

The following example retrieves a pointer to a request specific data structure of hypothetical type MYTAGDATA:

void DoSomeGruntWork( CCFXRequest* pRequest )
{
    MYTAGDATA* pTagData = (MYTAGDATA*)pRequest->GetCustomData();
    ... remainder of procedure ...
}
CCFXRequest::GetQuery

Syntax

CCFXQuery* CCFXRequest::GetQuery(void)

Description

Retrieves a query that was passed to a tag. To pass a query to a custom tag, you use the QUERY attribute. This attribute should be set to the name of a query (created using the CFQUERY tag or another custom tag). The QUERY attribute is optional and should be used only by tags that process an existing data set.

Returns

Returns an object of the CCFXQuery class that represents the query passed to the tag. If no query was passed to the tag, NULL is returned. The memory allocated for the returned query is freed automatically by ColdFusion after the request is completed.

Example

The following example retrieves the query that was passed to the tag. If no query was passed, an exception is thrown:

CCFXQuery* pQuery = pRequest->GetQuery() ;
if ( pQuery == NULL )
{
    pRequest->ThrowException(
        "Missing QUERY parameter",
        "You must pass a QUERY parameter in "
        "order for this tag to work correctly." ) ;
}

CCFXRequest::ReThrowException

Syntax

void CCFXRequest::ReThrowException(CCFXException* e)

Description

Re-throws an exception that has been caught within an extension procedure. This function is used to avoid having C++ exceptions that are thrown by DLL extension code propagate back into ColdFusion. Catch ALL C++ exceptions that occur in extension code, and either re-throw them (if they are of the CCFXException class) or create and throw a new exception pointer using CCFXRequest::ThrowException.

Parameters

<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:

```c++
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**

```c++
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:

```c++
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:

BOOL bMessageSent;
...attempt to send the message...
if ( bMessageSent == TRUE )
{
    pRequest->SetVariable( "MessageSent", "Yes" );
}
else
{
    pRequest->SetVariable( "MessageSent", "No" );
}

CCFXRequest::ThrowException

Syntax
void CCFXRequest::ThrowException(LPCSTR lpszError,
                                 LPCSTR lpszDiagnostics)

Description
Throws an exception and ends processing of a request. Call this function when you encounter an error that does not allow you to continue processing the request. This function is almost always combined with the CCFXRequest::ReThrowException to protect against resource leaks in extension code.

Parameters

<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:
```
char buffError[512] ;
wprintf( buffError, "Unexpected Windows NT error number %ld 
" occurred while processing request."); GetLastErro() ;
```
pRequest->ThrowException( "Error occurred", buffError ) ;

CCFXRequest::Write
Syntax
```
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:
```
CHAR buffOutput[1024] ;
wprintf( buffOutput, "The destination is: %s", pRequest->GetAttribute("DESTINATION") );
pRequest->Write( buffOutput ) ;
```

CCFXRequest::WriteDebug
Syntax
```
void CCFXRequest::WriteDebug(LPCSTR lpszOutput)
```
Description
Writes text output into the debug stream. The text is only displayed to the end-user if the tag contains the DEBUG attribute. (For more information, see CCFXRequest::Debug.)

Parameters
<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:
```
if ( pRequest->Debug() )
{
   pRequest->WriteDebug( "Top secret debug info" ) ;
}
```
CCFXStringSet class

Abstract class that represents a set of ordered strings. Strings can be added to a set and can be retrieved by a numeric index (index values for strings are 1-based). To create a string set, use `CCFXRequest::CreateStringSet`.

Class members

<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:

```cpp
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:

```c
int nNumItems = pStringSet->GetCount() ;
for ( int i=1; i<=nNumItems; i++ )
  {
    pRequest->Write( pStringSet->GetString( i ) ) ;
    pRequest->Write( "<BR>" ) ;
  }
```

**CCFXStringSet::GetIndexForString**

**Syntax**

```c
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:

```c
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**

```c
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>iIndex</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:

```cpp
int nNumItems = pStringSet->GetCount() ;
for ( int i=1; i<=nNumItems; i++ )
{
    pRequest->Write( pStringSet->GetString( i ) ) ;
    pRequest->Write( "<BR>" ) ;
}
```
CHAPTER 5
ColdFusion Java CFX Reference

This chapter describes the Java interfaces available for building ColdFusion custom CFXs in Java.

Contents
Overview class libraries ......................................................... 770
CustomTag interface ............................................................. 770
Query interface ................................................................. 772
Request interface .............................................................. 777
Response interface ............................................................ 782
Debugging classes reference ............................................... 785
Overview class libraries

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>CustomTag interface</td>
<td>processRequest</td>
</tr>
<tr>
<td>Query interface</td>
<td>addRow, getColumnIndex, getColumns, getData, getName, getRowCount, setData</td>
</tr>
<tr>
<td>Request interface</td>
<td>attributeExists, debug, getAttribute, getAttributeList, getIntAttribute, getQuery, getSetting</td>
</tr>
<tr>
<td>Response interface</td>
<td>addQuery, setVariable, write, writeDebug</td>
</tr>
</tbody>
</table>

CustomTag interface

public abstract interface CustomTag

Interface for implementing custom tags.

Classes that implement this interface can be specified in the CLASS attribute of the Java CFX tag. For example, in a class MyCustomTag, which implements this interface, the following CFML code calls the MyCustomTag.processRequest method:

```cfm
<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**

CustomTag interface

**Syntax**

```java
public void processRequest(Request request, Response response)
```

**Throws**

Exception If an unexpected error occurs while processing the request.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>request</td>
<td>Parameters (attributes, query, and so on.) for this request</td>
</tr>
<tr>
<td>response</td>
<td>Interface for generating response to request (output, variables, queries, and so on)</td>
</tr>
</tbody>
</table>
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
public String[] getColumns()

Example
The following example retrieves the array of columns, then iterates over the list, writing each column name back to the user:

// Get the list of columns from the query
String[] columns = query.getColumns() ;
int nNumColumns = columns.length ;
// Print the list of columns to the user
response.write( "Columns in query: " ) ;
for( int i=0; i<nNumColumns; i++ )
{  
    response.write( columns[i] + " " ) ;
}

**getData**

**Description**
Retrieves a data element from a row and column of a query. Row and column indexes begin with 1. You can find the number of rows in a query by calling `getRowCount`. You can find the number of columns in a query by calling `getColumns`.

Returns the value of the requested data element.

**Category**
Query interface

**Syntax**
```java
public String getData(int iRow, int iCol)
```

**Throws**
`IndexOutOfBoundsException` If an invalid index is passed to the method.

**See also**
`setData`, `addColumn`

**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
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
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
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
getAttribute, 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.

debug

Description
Checks whether the tag contains the debug attribute. Use this method to determine whether to
write debug information for this request. For more information, see writeDebug.

Returns True if the tag contains the debug attribute; otherwise returns False.

Category
Request interface

Syntax
public boolean debug()

See also
writeDebug

Example
The following example checks whether the debug attribute is present, and if so, it writes a brief
default message:
if ( request.debug() )
{
    response.writeDebug( "debug info" ) ;
}

getAttribute

Description
Retrieves the value of a passed attribute. Returns an empty string if the attribute does not exist
(use attributeExists to test whether an attribute was passed to the tag). Use
getAttribute(String,String) to return a default value rather than an empty string.

Returns the value of the attribute passed to the tag. If no attribute of that name was passed to the
tag, an empty string is returned.

Category
Request interface

Syntax
public String getAttribute(String name)

See also
attributeExists,getAttributeList,getIntAttribute,getAttribute

Parameters

<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 ) ;
```

ggetAttributeList

Description
Retrieves a list of attributes passed to the tag. To retrieve the value of one attribute, use the
getAttribute member function.
Returns an array of strings containing the names of the attributes passed to the tag.

Category
Request interface

Syntax
```java
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
```
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

```java
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><code>addQuery(String name, String[] columns)</code></td>
<td>Adds a query to the calling template.</td>
</tr>
<tr>
<td>void</td>
<td><code>setVariable(String name, String value)</code></td>
<td>Sets a variable in the calling template.</td>
</tr>
<tr>
<td>void</td>
<td><code>write(String output)</code></td>
<td>Outputs text back to the user.</td>
</tr>
<tr>
<td>void</td>
<td><code>writeDebug(String output)</code></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 member functions `addRow` and `setData`.

Returns the Query that was added to the template.

**Category**

Response interface

**Syntax**

```java
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**
```java
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:
```java
boolean bMessageSent;
...
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:
```
response.write( "DESTINATION = " +
    request.getAttribute("DESTINATION") ) ;
```

writeDebug

**Description**
Writes text output into the debug stream. This text is displayed to the end-user only if the tag contains the debug attribute (check for this attribute using the `Request.debug` member function).

**Category**
Response interface

**Syntax**
```
public void writeDebug(String output)
```

**See also**
depth

**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:
```
if ( request.debug() )
{
    response.writeDebug( "debug info" ) ;
}
```
Debugging classes reference

The constructors and methods supported by the `DebugRequest`, `DebugResponse`, and `DebugQuery` classes are as follows. These classes also support the other methods of the `Request`, `Response`, and `Query` interfaces, respectively.

**DebugRequest**

// initialize a debug request with attributes
public DebugRequest(Hashtable attributes);

// initialize a debug request with attributes and a query
public DebugRequest(Hashtable attributes, Query query);

// initialize a debug request with attributes, a query, and settings
public DebugRequest(Hashtable attributes, Query query, Hashtable settings);

**DebugResponse**

// initialize a debug response
public DebugResponse();

// print the results of processing
public void printResults();

**DebugQuery**

// initialize a query with name and columns
public DebugQuery(String name, String[] columns)
   throws IllegalArgumentException;

// initialize a query with name, columns, and data
public DebugQuery(String name, String[] columns, String[][] data)
   throws IllegalArgumentException;
This chapter provides information about JavaScript objects and functions used to WDDX in a ColdFusion application.

**Contents**

- JavaScript object overview ........................................... 788
- WddxSerializer object ............................................... 789
- WddxRecordset object ............................................... 793
JavaScript object overview

These are the JavaScript objects and functions.

<table>
<thead>
<tr>
<th>Class</th>
<th>Members</th>
</tr>
</thead>
<tbody>
<tr>
<td>WddxSerializer object</td>
<td>serialize, serializeVariable, serializeValue, write</td>
</tr>
<tr>
<td>WddxRecordset object</td>
<td>addColumn, addRows, getField, getRowCount, setField, wddxSerialize</td>
</tr>
</tbody>
</table>

WDDX JavaScript objects are defined in the wddx.js file; this file is installed in the webroot/cfide/scripts directory.

To use these objects, you must put a JavaScript tag before the code that refers to the objects; for example:

```html
<script type="text/javascript" src="/CFIDE/scripts/wddx.js"></script>
```
WddxSerializer object

The WddxSerializer object includes functions that serialize any JavaScript data structure.

Functions

The only function that developers typically call is `serialize`.

<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 WddxRecordset 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, WddxSerializer 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 WddxSerializer 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);
    if (wddxPacket != null)
    {
        formField.value = wddxPacket;
    }
    else
    {
    }
}  ```
alert("Couldn't serialize data");
}
}

serializeVariable

Description
Serializes a property of a structure. If an object is not a string, number, array, Boolean, or date, WddxSerializer treats it as a structure.

Syntax
object.serializeVariable( name, obj )

Parameters

<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, or False if an error occurs.

Usage
This is an internal function; you do not typically call it.

Example
This example is from the WddxSerializer serializeValue function:

```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; or False if an error occurs.

Usage

This is an internal function; you do not typically call it.

Example

This example is from the WddxSerializer serialize function:

```javascript
... 
this.wddxPacket = "";
this.write("<wddxPacket version="1.0">header</wddxPacket>" );
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:
...
else if (typeof(obj) === "number")
{
    // Number value
    this.write("<number>" + obj + "</number>");
}
else if (typeof(obj) === "boolean")
{
    // Boolean value
    this.write("<boolean value='" + obj + "'/>");
}
...
**WddxRecordset object**

Includes functions that you call as needed when constructing a WDDX record set.

**Functions**

<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>
<tr>
<td><code>object.getField(row, col)</code></td>
<td>Returns the element in a row/column position.</td>
</tr>
<tr>
<td><code>object.getRowCount()</code></td>
<td>Indicates the number of rows in a WddxRecordset instance.</td>
</tr>
<tr>
<td><code>object.setField(row, col, value)</code></td>
<td>Sets the element in a row/column position.</td>
</tr>
<tr>
<td><code>object.wddxSerialize(serializer)</code></td>
<td>Serializes a record set.</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 `<>&` characters in string values are escaped as `&lt;&gt;&amp;` in HTML.

**Example**

```cftagscript
<!--- Create a simple query --->
<cfquery name = "q" datasource ="cfsnippets">
SELECT Message_Id, Thread_id, Username, Posted FROM messages
</cfquery>
<!--- Load the wddx.js file, which includes the dump function --->
<script type="text/javascript" src="/CFIDE/scripts/wddx.js"></script>
<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**

```cftagscript
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:

```javascript
// 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

```javascript
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:
// 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):

```java
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**
```javascript
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**
```javascript
object.setField( row, col, value )
```

**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>
<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:

```java
// 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**
```java
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; or False if an error occurs.

**Usage**
This is an internal function; you do not typically call it.

**Example**
This example is from the WddxSerializer serializeValue function:

```java
... 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);
    }
    ...
CHAPTER 7
ColdFusion ActionScript Functions

This chapter explains the syntax and usage of the two server-side ActionScript functions, CF.query and CF.http.

Contents
CF.query ................................................................. 800
CF.http ............................................................... 801
### CF.query

**Description**
Performs queries against ColdFusion data sources.

**Return value**
Returns a RecordSet object.

**Syntax**
```
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:

```
CF.query(
    datasource: "data source name",
    sql: "SQL stmts",
    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:

```
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 30, “Using Server-Side ActionScript,” of Developing ColdFusion MX Applications. 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

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. |
| 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  
• bgsound 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. |
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:

```javascript
CF.http({method:"method", url:"URL", username:"username", password:"password", resolveurl:"yes or no", params:arrayvar, path:"path", file:"filename"});
```

**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);
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>
</tbody>
</table>
You access these attributes using the \texttt{get} function:

\begin{verbatim}
function basicGet()
{
    url = "http://localhost:8100/";

    // Invoke with just the url. This is an HTTP GET.
    result = CF.http(url);
    return result.get("Filecontent");
}
\end{verbatim}

\textbf{Note:} For more information on using server-side ActionScript, see Chapter 30, "Using Server-Side ActionScript," of Developing ColdFusion MX Applications.

**Example**

The following examples show a number of the ways to use the \texttt{CF.http} function:

\begin{verbatim}
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";
\end{verbatim}

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 cfhttpparam 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 cfhttpparam 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");
}