Adobe ColdFusion Documentation

September 2014

1. Developing Adobe ColdFusion Applications . . . 11
1.1 Changes in ColdFusion . . . 12
1.1.1 Replacement of JRun with Tomcat . . . 15
1.1.2 Security enhancements . . . 16
1.1.3 ColdFusion WebSocket . . . 17
1.1.4 ColdFusion closures . . . 18
1.1.5 Enhanced Java integration . . . 19
1.1.6 ColdFusion ORM search for indexing and search . . . 20
1.1.7 Solr enhancements . . . 21
1.1.8 Scheduler enhancements . . . 22
1.1.9 Integration with Microsoft Exchange Server 2010 . . . 23
1.1.10 Lazy loading across client and server in ColdFusion . . . 24
1.1.11 Web service enhancements . . . 25
1.1.12 RESTful Web Services in ColdFusion . . . 26
1.1.13 Media Player enhancements . . . 55
1.1.14 Displaying geolocation . . . 56
1.1.15 Client-side charting . . . 57
1.1.16 Caching enhancements . . . 58
1.1.17 Server update using ColdFusion Administrator . . . 59
1.1.18 Secure Profile for ColdFusion Administrator . . . 60
1.2 Introduction to application development using ColdFusion . . . 61
1.2.1 Using the Developing ColdFusion Applications guide . . . 62
1.2.2 About Adobe ColdFusion documentation for Developers . . . 63
1.3 Introducing ColdFusion . . . 64
1.3.1 About Internet applications and web application servers . . . 65
1.3.1.1 About web application servers . . . 66
1.3.1.2 About web pages and Internet applications . . . 68
1.3.2 About ColdFusion . . . 69
1.3.2.1 ColdFusion Administrator . . . 70
1.3.2.2 Elements of ColdFusion . . . 71
1.3.2.3 Saving ColdFusion pages . . . 73
1.3.2.4 Testing ColdFusion pages . . . 74
1.3.3 About JEE and the ColdFusion architecture . . . 75
1.3.3.1 About ColdFusion and the JEE platform . . . 76
1.4 The CFML Programming Language . . . 77
1.4.1 Elements of CFML . . . 78
1.4.1.1 CFML Basics . . . 79
1.4.1.2 Comments . . . 80
1.4.1.3 Tags . . . 81
1.4.1.4 Functions . . . 95
1.4.1.5 ColdFusion components . . . 98
1.4.1.6 Constants . . . 99
1.4.1.7 Variables . . . 100
1.4.1.8 Expressions . . . 101
1.4.1.9 Data types . . . 102
1.4.1.10 Flow control . . . 103
1.4.1.11 Character case . . . 107
1.4.1.12 Special characters . . . 108
1.4.1.13 Reserved words in ColdFusion . . . 109
1.4.1.14 cfscript tag . . . 110
1.4.1.15 Elvis operator . . . 111
1.4.2 Using ColdFusion Variables . . . 112
1.4.2.1 Creating variables . . . 113
1.4.2.2 Data types- Developing guide . . . 114
1.4.2.3 Using periods in variable references . . . 126
1.4.2.4 Data type conversion . . . 129
1.4.2.5 About scopes . . . 135
1.4.2.6 Ensuring variable existence . . . 143
1.4.2.7 Validating data . . . 146
1.4.2.8 Passing variables to custom tags and UDFs . . . 147
1.4.3 Using Expressions and Number Signs . . . 148
1.4.3.1 Expressions-Developing guide . . . 149
1.4.3.2 Using number signs . . . 157
1.4.3.3 Dynamic expressions and dynamic variables . . . 161
1.4.4 Using Arrays and Structures . . . 175
1.4.4.1 About arrays . . . 176
1.4.4.2 Basic array techniques . . . 178
1.4.4.3 Populating arrays with data . . . 185
1.4.4.4 Array functions-Developing guide . . . 188

1.4.4.5 About structures . . . 190
1.4.4.6 Creating and using structures . . . 195
1.4.4.7 Structure examples . . . 204
1.4.4.8 Structure functions - Developing guide . . . 211
1.4.5 Extending ColdFusion Pages with CFML Scripting . . . 213
1.4.5.1 About CFScript . . . 214
1.4.5.2 Language Enhancements in ColdFusion 9 . . . 216
1.4.5.3 What is supported in CFScript . . . 218
1.4.5.4 The CFScript language . . . 221
1.4.5.5 Using CFScript statements . . . 224
1.4.5.6 Defining components and functions in CFScript . . . 235
1.4.5.7 Import and new operations using cfimport . . . 241
1.4.5.8 Handling exceptions . . . 243
1.4.5.9 CFScript example . . . 245
1.4.5.10 Using closures . . . 249
1.4.6 Using Regular Expressions in Functions . . . 257
1.4.6.1 About regular expressions . . . 258
1.4.6.2 Regular expression syntax . . . 260
1.4.6.3 Using backreferences . . . 269
1.4.6.4 Returning matched subexpressions . . . 272
1.4.6.5 Regular expression examples . . . 276
1.4.6.6 Types of regular expression technologies . . . 278
1.4.7 ColdFusion Language Enhancements . . . 279
1.4.8 Built-in functions as first class citizen . . . 281
1.5 Building Blocks of ColdFusion Applications . . . 282
1.5.1 Creating ColdFusion Elements . . . 283
1.5.1.1 About CFML elements that you create . . . 284
1.5.1.2 Including pages with the cfinclude tag . . . 285
1.5.1.3 User-defined functions with ColdFusion Elements . . . 287
1.5.1.4 About user-defined functions-Developing guide . . . 288
1.5.1.5 Using ColdFusion components . . . 289
1.5.1.6 Using custom CFML tags . . . 290
1.5.1.7 Using CFX tags . . . 292
1.5.1.8 Selecting among ColdFusion code reuse methods . . . 293
1.5.2 Writing and Calling User-Defined Functions . . . 295
1.5.2.1 About user-defined functions . . . 296
1.5.2.2 Creating user-defined functions . . . 297
1.5.2.3 Calling user-defined functions . . . 304
1.5.2.4 Working with arguments and variables in functions . . . 305
1.5.2.5 Using UDFs effectively . . . 313
1.5.2.6 Handling errors in UDFs . . . 319
1.5.2.7 A user-defined function example . . . 325
1.5.3 Building and Using ColdFusion Components . . . 327
1.5.3.1 About ColdFusion components . . . 328
1.5.3.2 Creating ColdFusion components . . . 330
1.5.3.3 Building ColdFusion components . . . 331
1.5.3.4 CFC variables and scope . . . 343
1.5.3.5 Using ColdFusion components-Developing guide . . . 346
1.5.3.6 Passing parameters to methods . . . 355
1.5.3.7 Using CFCs effectively . . . 358
1.5.3.8 ColdFusion component example . . . 366
1.5.3.9 Implicit constructor for CFC . . . 369
1.5.3.10 Method chaining for CFC methods . . . 372
1.5.3.11 CFC Implicit notation . . . 373
1.5.4 Creating and Using Custom CFML Tags . . . 374
1.5.4.1 Creating custom tags . . . 375
1.5.4.2 Passing data to custom tags . . . 379
1.5.4.3 Managing custom tags . . . 385
1.5.4.4 Executing custom tags . . . 386
1.5.4.5 Nesting custom tags . . . 390
1.5.5 Building Custom CFXAPI Tags . . . 395
1.5.5.1 What are CFX tags? . . . 396
1.5.5.2 Before you begin developing CFX tags in Java . . . 397
1.5.5.3 Writing a Java CFX tag . . . 399
1.5.5.4 ZipBrowser example . . . 403
1.5.5.5 Approaches to debugging Java CFX tags . . . 405
1.5.5.6 Developing CFX tags in C++ . . . 408
1.5.6 Using the member functions . . . 410
1.6 Developing CFML Applications . . . 419
1.6.1 Designing and Optimizing a ColdFusion Application . . . 420
1.6.1.1 About applications . . . 421
1.6.1.2 Elements of a ColdFusion application . . . 422

1.6.1.3 Structuring an application . . . 426
1.6.1.4 Defining the application and its event handlers in Application.cfc . . . 428
1.6.1.5 Migrating from Application.cfm to Application.cfc . . . 442
1.6.1.6 Using an Application.cfm page . . . 443
1.6.1.7 Optimizing ColdFusion applications . . . 448
1.6.2 Handling Errors . . . 477
1.6.2.1 About error handling in ColdFusion . . . 478
1.6.2.2 Understanding errors . . . 479
1.6.2.3 Error messages and the standard error format . . . 485
1.6.2.4 Determining error-handling strategies . . . 486
1.6.2.5 Specifying custom error messages with the cferror tag . . . 489
1.6.2.6 Logging errors with the cflog tag . . . 492
1.6.2.7 Handling runtime exceptions with ColdFusion tags . . . 494
1.6.3 Using Persistent Data and Locking . . . 513
1.6.3.1 About persistent scope variables . . . 514
1.6.3.2 Managing the client state . . . 517
1.6.3.3 Configuring and using client variables . . . 522
1.6.3.4 Configuring and using session variables . . . 528
1.6.3.5 Configuring and using application variables . . . 534
1.6.3.6 Using server variables . . . 536
1.6.3.7 Locking code with cflock . . . 538
1.6.3.8 Examples of cflock . . . 545
1.6.4 Using ColdFusion Threads . . . 550
1.6.4.1 About ColdFusion threads . . . 551
1.6.4.2 Creating and managing ColdFusion threads . . . 552
1.6.4.3 Using thread data . . . 556
1.6.4.4 Working with threads . . . 560
1.6.4.5 Using ColdFusion tools to control thread use . . . 563
1.6.4.6 Example: getting multiple RSS feeds . . . 564
1.6.5 Securing Applications . . . 567
1.6.5.1 ColdFusion security features . . . 568
1.6.5.2 About resource and sandbox security . . . 569
1.6.5.3 About user security . . . 571
1.6.5.4 Using ColdFusion security tags and functions . . . 577
1.6.5.5 Security scenarios . . . 582
1.6.5.6 Implementing user security . . . 586
1.6.5.7 Security enhancements in ColdFusion 10 . . . 603
1.6.6 Developing Globalized Applications . . . 609
1.6.6.1 Introduction to globalization . . . 610
1.6.6.2 About character encodings . . . 612
1.6.6.3 Locales . . . 614
1.6.6.4 Processing a request in ColdFusion . . . 616
1.6.6.5 Tags and functions for globalizing applications . . . 618
1.6.6.6 Handling data in ColdFusion . . . 622
1.6.7 Debugging and Troubleshooting Applications . . . 626
1.6.7.1 Configuring debugging in the ColdFusion Administrator . . . 627
1.6.7.2 Using debugging information from browser pages . . . 630
1.6.7.3 Controlling debugging information in CFML . . . 635
1.6.7.4 Using the cftrace tag to trace execution . . . 637
1.6.7.5 Using the cftimer tag to time blocks of code . . . 642
1.6.7.6 Using the Code Analyzer . . . 645
1.6.7.7 Troubleshooting common problems . . . 646
1.6.8 Using the ColdFusion Debugger . . . 648
1.6.8.1 About the ColdFusion Debugger . . . 649
1.6.8.2 Install and uninstall the ColdFusion Debugger . . . 650
1.6.8.3 Set up ColdFusion to use the Debugger . . . 651
1.6.8.4 About the Debug perspective . . . 653
1.6.8.5 Using the ColdFusion Debugger-Developing guide . . . 654
1.6.8.6 Viewing ColdFusion log files . . . 656
1.6.8.7 Using Scheduler . . . 657
1.6.9 Client-side CFML (for mobile development) . . . 662
1.6.10 Social Enhancements . . . 685
1.6.11 REST Enhancements in ColdFusion 11 . . . 688
1.6.12 Authentication through OAuth . . . 689
1.7 Accessing and Using Data . . . 690
1.7.1 Introduction to Databases and SQL . . . 691
1.7.1.1 What is a database? . . . 692
1.7.1.2 Using SQL . . . 695
1.7.2 Accessing and Retrieving Data . . . 704
1.7.2.1 Working with dynamic data . . . 705
1.7.2.2 Retrieving data . . . 706
1.7.2.3 Outputting query data . . . 709

1.7.2.4 Getting information about query results . . . 711
1.7.2.5 Enhancing security with cfqueryparam . . . 713
1.7.3 Updating Your Database . . . 715
1.7.3.1 About updating your database . . . 716
1.7.3.2 Inserting data . . . 717
1.7.3.3 Updating data . . . 723
1.7.3.4 Deleting data . . . 730
1.7.3.5 Database-related enhancements in ColdFusion 10 . . . 732
1.7.4 Using Query of Queries . . . 735
1.7.4.1 About recordsets . . . 736
1.7.4.2 About Query of Queries . . . 738
1.7.4.3 Query of Queries user guide . . . 744
1.7.5 Managing LDAP Directories . . . 763
1.7.5.1 About LDAP . . . 764
1.7.5.2 The LDAP information structure . . . 766
1.7.5.3 Using LDAP with ColdFusion . . . 769
1.7.5.4 Querying an LDAP directory . . . 771
1.7.5.5 Updating an LDAP directory . . . 778
1.7.5.6 Advanced topics . . . 790
1.7.6 Solr search support . . . 799
1.7.6.1 CFML tags used to configure Solr . . . 800
1.7.6.2 Solr search service . . . 801
1.7.6.3 Language support . . . 802
1.7.6.4 Solr search examples . . . 803
1.7.6.5 Term highlighting . . . 806
1.7.6.6 Configuring Solr memory . . . 808
1.7.6.7 Solr enhancements in ColdFusion 9.0.1 . . . 809
1.7.6.8 Solr enhancements in ColdFusion 10 . . . 810
1.8 ColdFusion ORM . . . 820
1.8.1 Introducing ColdFusion ORM . . . 821
1.8.1.1 ColdFusion ORM example . . . 822
1.8.2 Architecture . . . 826
1.8.3 Configure ORM . . . 827
1.8.3.1 Logging . . . 828
1.8.3.2 ORM settings . . . 830
1.8.4 Define ORM mapping . . . 836
1.8.4.1 Map a ColdFusion component . . . 837
1.8.4.2 Map the properties . . . 841
1.8.4.3 Define Relationships . . . 856
1.8.4.4 Advanced mapping . . . 874
1.8.4.5 Mapping CFCs without properties . . . 886
1.8.5 Working with objects . . . 887
1.8.5.1 Generated accessors . . . 888
1.8.5.2 Generated methods for relationships between CFCs . . . 889
1.8.5.3 Perform create, read, update, delete operations on ORM Objects . . . 891
1.8.6 Using queries . . . 896
1.8.6.1 ORMExecuteQuery(hql, [,unique] [, queryoptions]) . . . 897
1.8.6.2 ORMExecuteQuery(hql, params [,unique] [,queryOptions]) . . . 899
1.8.7 Transaction and concurrency . . . 902
1.8.7.1 Change of behavior in ColdFusion 9 Update 1 . . . 904
1.8.7.2 Optimistic locking . . . 905
1.8.8 Performance optimization . . . 907
1.8.8.1 Lazy Loading . . . 908
1.8.8.2 Caching . . . 911
1.8.9 ORM session management . . . 917
1.8.9.1 Event Handling in CFC . . . 918
1.8.9.1.1 Event handling in a persistent CFC . . . 919
1.8.9.1.2 Event handling using an event handler CFC . . . 920
1.8.10 Autogenerating database schema . . . 921
1.8.10.1 Naming strategy . . . 922
1.8.10.2 Example . . . 924
1.8.10.3 Populating the database using script . . . 925
1.8.11 Support for multiple data sources for ORM . . . 926
1.8.11.1 Configuring the application to use multiple data sources . . . 927
1.8.11.2 Introduction-Multiple data sources for ORM . . . 929
1.8.11.3 Usage scenario . . . 930
1.8.11.4 ORM Function enhancements . . . 931
1.8.12 ColdFusion ORM search . . . 934
1.8.12.1 Indexing modes . . . 935
1.8.12.2 Indexing the persistent entity . . . 936
1.8.12.3 Other enhancements . . . 940
1.8.12.4 Performing the search . . . 941

1.8.12.5 Search based on indexed information . . . 942
1.9 ColdFusion and HTML 5 . . . 943
1.9.1 Using ColdFusion WebSocket . . . 944
1.9.1.1 WebSocket Enhancements . . . 945
1.9.1.2 Browsers and fallback . . . 954
1.9.1.3 ColdFusion and WebSocket . . . 955
1.9.1.4 How is WebSocket different from conventional communication models . . . 956
1.9.1.5 WebSocket communication models . . . 957
1.9.1.6 What is WebSocket . . . 958
1.9.1.7 Using WebSocket to broadcast messages . . . 959
1.9.1.8 Using WebSocket in point-to-point communication . . . 988
1.9.1.9 Error handling in ColdFusion WebSocket . . . 992
1.9.1.10 Using ColdFusion Administrator for WebSocket Configurations . . . 993
1.9.1.11 Browser support for WebSocket . . . 994
1.9.2 Media Player enhancements-Developing guide . . . 995
1.9.2.1 HTML 5 video playback capability . . . 996
1.9.2.2 Fallback plan for cfmediaplayer . . . 997
1.9.2.3 Specifying the playback type . . . 998
1.9.2.4 Specifying the video poster image, loop playback, and title using the tag cfmediaplayer . . . 999
1.9.2.5 Extending the media player capabilities . . . 100
1.9.2.6 Embedding subtitle files to an HTML 5 video . . . 100
1.9.2.7 Play list support for Flash videos . . . 100
1.9.2.8 Skinning . . . 100
1.9.2.9 Player controls . . . 100
1.9.2.10 Dynamic streaming . . . 100
1.9.3 Displaying geolocation - Developing guide . . . 100
1.9.3.1 cfinput-Developing guide . . . 101
1.9.3.2 cfmap-Developing guide . . . 101
1.9.3.3 cfmapitem-Developing guide . . . 101
1.9.4 Client-side charting-Developing guide . . . 101
1.9.4.1 Supported charts . . . 101
1.9.4.2 How client side charting works . . . 101
1.9.4.3 Limitations . . . 101
1.9.4.4 Charting examples . . . 101
1.10 Flex and AIR Integration in ColdFusion . . . 101
1.10.1 Using the Flash Remoting Service . . . 101
1.10.1.1 About using the Flash Remoting service with ColdFusion . . . 102
1.10.1.2 Configuring the Flash Remoting Gateway . . . 102
1.10.1.3 Using the Flash Remoting service with ColdFusion pages . . . 102
1.10.1.4 Using Flash with CFCs . . . 103
1.10.1.5 Using the Flash Remoting service with ColdFusion Java objects . . . 103
1.10.1.6 Handling errors with ColdFusion and Flash . . . 103
1.10.2 Using Flash Remoting Update . . . 103
1.10.2.1 Prerequisites for using Flash Remoting Update . . . 104
1.10.2.2 Configure Flex Compilation . . . 104
1.10.2.3 Specify a CFC . . . 104
1.10.2.4 Use the CFC . . . 104
1.10.2.5 Compile and Run the application . . . 104
1.10.2.6 Lazy loading across client and server . . . 104
1.10.3 Offline AIR Application Support . . . 105
1.10.3.1 ColdFusion server side . . . 105
1.10.3.2 Offline AIR application code constructs . . . 105
1.10.3.3 Client Side . . . 106
1.10.3.4 Using the AIR SyncManager class to manage data . . . 106
1.10.3.5 Conflict management . . . 106
1.10.3.6 Offline AIR application example . . . 107
1.10.3.7 Offline AIR application support in ColdFusion 9.0.1 . . . 108
1.10.4 Proxy ActionScript Classes for ColdFusion Services . . . 108
1.10.4.1 About the cfservices.swc file . . . 109
1.10.4.2 Using the Chart class . . . 109
1.10.4.3 Using the Config class . . . 109
1.10.4.4 Using the Document class . . . 109
1.10.4.5 Using the Image class . . . 109
1.10.4.6 Using the Mail class . . . 110
1.10.4.7 Using the PDF class . . . 110
1.10.4.8 Using the Pop class . . . 111
1.10.4.9 Using the Util class . . . 111
1.10.5 Using the LiveCycle Data Services ES Assembler . . . 111
1.10.5.1 About ColdFusion and Flex-Developing guide . . . 111
1.10.5.2 Application development and deployment process-Developing guide . . . 111
1.10.5.3 Configuring a destination for the Data Service Adapter . . . 112
1.10.5.4 Writing the ColdFusion CFCs . . . 113

1.10.5.5 Notifying the Flex application when data changes . . . 113
1.10.5.6 Authentication . . . 114
1.10.5.7 Enabling SSL . . . 114
1.10.5.8 Data translation-Developing guide . . . 114
1.10.6 Using Server-Side ActionScript . . . 114
1.10.6.1 About server-side ActionScript . . . 114
1.10.6.2 Connecting to the Flash Remoting service . . . 114
1.10.6.3 Using server-side ActionScript functions . . . 114
1.10.6.4 Global and request scope objects . . . 115
1.10.6.5 About the CF.query function and data sources . . . 115
1.10.6.6 Using the CF.query function . . . 115
1.10.6.7 Building a simple application . . . 115
1.10.6.8 About the CF.http function . . . 115
1.10.6.9 Using the CF.http function . . . 116
1.11 Requesting and Presenting Information . . . 116
1.11.1 Introduction to Retrieving and Formatting Data . . . 116
1.11.1.1 Using forms in ColdFusion . . . 116
1.11.1.2 Working with action pages . . . 117
1.11.1.3 Working with queries and data . . . 117
1.11.1.4 Returning results to the user . . . 118
1.11.1.5 Dynamically populating list boxes . . . 118
1.11.1.6 Creating dynamic check boxes and multiple-selection list boxes . . . 118
1.11.2 Building Dynamic Forms with cfform Tags . . . 119
1.11.2.1 Creating custom forms with the cfform tag . . . 119
1.11.2.2 Building drop-down list boxes . . . 119
1.11.2.3 Building slider bar controls . . . 119
1.11.2.4 Building tree controls with the cftree tag . . . 119
1.11.2.5 Creating data grids with the cfgrid tag . . . 120
1.11.2.6 Embedding Java applets . . . 121
1.11.3 Validating Data-Developing guide . . . 121
1.11.3.1 About ColdFusion validation . . . 122
1.11.3.2 Handling invalid data . . . 122
1.11.3.3 Masking form input values . . . 123
1.11.3.4 Validating data with the IsValid function and the cfparam tag . . . 123
1.11.3.5 Validating form data using hidden fields . . . 123
1.11.3.6 Validating form data with regular expressions . . . 124
1.11.3.7 Validating form fields . . . 124
1.11.3.8 Validating form input and handling errors with JavaScript . . . 124
1.11.4 Creating Forms in Flash . . . 125
1.11.4.1 About Flash forms . . . 125
1.11.4.2 Best practices for Flash forms . . . 125
1.11.4.3 Binding data in Flash forms . . . 125
1.11.4.4 Building Flash forms . . . 125
1.11.4.5 Setting styles and skins in Flash forms . . . 127
1.11.4.6 Using ActionScript in Flash forms . . . 127
1.11.5 Creating Skinnable XML Forms . . . 127
1.11.5.1 About XML skinnable forms . . . 127
1.11.5.2 Building XML skinnable forms . . . 127
1.11.5.3 ColdFusion XML format . . . 128
1.11.5.4 Creating XSLT skins . . . 129
1.11.6 Using Ajax User Interface Components and Features . . . 130
1.11.6.1 About Ajax and ColdFusion user interface features . . . 130
1.11.6.2 Controlling Ajax user interface layout . . . 130
1.11.6.3 Using menus and toolbars . . . 131
1.11.6.4 Uploading files . . . 132
1.11.6.5 Using Ajax form controls and features . . . 132
1.11.6.6 Using geographical maps . . . 135
1.11.6.7 Using the cfprogressbar tag . . . 136
1.11.6.8 Using the cfmessagebox tag . . . 136
1.11.7 Using Ajax Data and Development Features . . . 137
1.11.7.1 About ColdFusion Ajax data and development features . . . 137
1.11.7.2 Binding data to form fields . . . 137
1.11.7.3 Managing the client-server interaction . . . 138
1.11.7.4 Using Spry with ColdFusion . . . 139
1.11.7.5 Specifying client-side support files . . . 139
1.11.7.6 Using data interchange formats . . . 140
1.11.7.7 Debugging Ajax applications . . . 140
1.11.7.8 Ajax programming rules and techniques . . . 140
1.12 Office file interoperability . . . 140
1.12.1 Using cfdocument . . . 141
1.12.1.1 Working with documents using OpenOffice . . . 141
1.12.1.2 Working with PowerPoint presentation files . . . 141

1.12.2 Using cfpresentation . . . 141
1.12.2.1 Example for converting from PowerPoint to Connect . . . 141
1.12.3 Using cfspreadsheet . . . 141
1.12.3.1 Example- storing the spreadsheet data in a CSV string . . . 141
1.12.4 Supported Office conversion formats . . . 142
1.12.5 SharePoint integration . . . 142
1.12.5.1 Load SharePoint actions from ColdFusion . . . 142
1.12.5.2 Using cfsharepoint . . . 142
1.12.5.3 Access ColdFusion from SharePoint using custom Web Parts . . . 142
1.12.5.4 Use Single Sign-On to access ColdFusion applications via SharePoint . . . 142
1.13 ColdFusion Portlets . . . 143
1.13.1 Run a ColdFusion portlet on JBoss Portal Server . . . 143
1.13.1.1 Prerequisites . . . 143
1.13.1.2 Build a portlet for a local server . . . 143
1.13.1.3 Access remote ColdFusion portlets (WSRP) . . . 143
1.13.2 Run a ColdFusion portlet on WebSphere Portal Server . . . 143
1.13.3 Common methods used in portlet.cfc . . . 144
1.13.4 ColdFusion portlet components . . . 144
1.13.4.1 Portlet modes . . . 144
1.13.4.2 Portlet window states . . . 144
1.13.4.3 Portlet title . . . 144
1.13.4.4 Portlet scope . . . 144
1.13.4.5 Create portlet parameters . . . 144
1.13.4.6 Processing actions using form posts . . . 144
1.13.4.7 Examples . . . 144
1.13.5 JSR-286 Support . . . 145
1.13.5.1 Publishing and Processing Events . . . 145
1.13.5.2 Publishing an event (Event producer) . . . 145
1.13.5.3 Processing an event (event consumer) . . . 145
1.13.5.4 Initiate events in portlet (CFC) . . . 145
1.13.5.5 Using filters . . . 145
1.14 Working with Documents, Charts, and Reports . . . 145
1.14.1 Manipulating PDF Forms in ColdFusion . . . 145
1.14.1.1 About PDF forms . . . 145
1.14.1.2 Populating a PDF form with XML data . . . 146
1.14.1.3 Prefilling PDF form fields . . . 146
1.14.1.4 Embedding a PDF form in a PDF document . . . 146
1.14.1.5 Extracting data from a PDF form submission . . . 146
1.14.1.6 Application examples that use PDF forms . . . 147
1.14.2 Assembling PDF Documents . . . 148
1.14.2.1 About assembling PDF documents . . . 148
1.14.2.2 Using shortcuts for common tasks . . . 148
1.14.2.3 Using DDX to perform advanced tasks . . . 150
1.14.2.4 FDF format support for AcroForms . . . 151
1.14.2.5 Application examples . . . 151
1.14.3 Creating and Manipulating ColdFusion Images . . . 152
1.14.3.1 About ColdFusion images . . . 152
1.14.3.2 Creating ColdFusion images . . . 152
1.14.3.3 Converting images . . . 153
1.14.3.4 Verifying images . . . 153
1.14.3.5 Enforcing size restrictions . . . 153
1.14.3.6 Compressing JPEG images . . . 153
1.14.3.7 Manipulating ColdFusion images . . . 153
1.14.3.8 Writing images to the browser . . . 154
1.14.3.9 Application examples that use ColdFusion images . . . 154
1.14.4 Creating Charts and Graphs . . . 155
1.14.4.1 About charts . . . 155
1.14.4.2 Creating a basic chart . . . 156
1.14.4.3 Charting data . . . 156
1.14.4.4 Controlling chart appearance . . . 157
1.14.4.5 Creating charts: examples . . . 158
1.14.4.6 Administering charts . . . 158
1.14.4.7 Writing a chart to a variable . . . 159
1.14.4.8 Linking charts to URLs . . . 159
1.14.4.9 Charting enhancements . . . 159
1.14.5 Creating Reports and Documents for Printing . . . 159
1.14.5.1 About printable output . . . 159
1.14.5.2 Creating PDF and FlashPaper output with the cfdocument tag . . . 160
1.14.5.3 Creating reports with Crystal Reports (Windows only) . . . 160
1.14.6 Creating Reports with Report Builder . . . 160
1.14.6.1 About Report Builder . . . 161
1.14.6.2 Getting started . . . 161

1.14.6.3 Common reporting tasks and techniques . . . 161
1.14.6.4 Creating a simple report . . . 163
1.14.7 Creating Slide Presentations . . . 164
1.14.7.1 About ColdFusion presentations . . . 164
1.14.7.2 Creating a slide presentation . . . 164
1.14.7.3 Adding presenters . . . 164
1.14.7.4 Adding slides . . . 165
1.14.7.5 Sample presentations . . . 165
1.14.8 PDF Generation in ColdFusion 11 . . . 165
1.15 Using Web Elements and External Objects . . . 166
1.15.1 Using XML and WDDX . . . 166
1.15.1.1 About XML and ColdFusion . . . 166
1.15.1.2 The XML document object . . . 166
1.15.1.3 ColdFusion XML tag and functions . . . 167
1.15.1.4 Using an XML object . . . 167
1.15.1.5 Creating and saving an XML document object . . . 167
1.15.1.6 Modifying a ColdFusion XML object . . . 168
1.15.1.7 Validating XML documents . . . 169
1.15.1.8 Transforming documents with XSLT . . . 169
1.15.1.9 Extracting data with XPath . . . 169
1.15.1.10 Example: using XML in a ColdFusion application . . . 169
1.15.1.11 Moving complex data across the web with WDDX . . . 170
1.15.1.12 Using WDDX . . . 171
1.15.2 Using Web Services . . . 171
1.15.2.1 Web services . . . 171
1.15.2.2 Working with WSDL files . . . 171
1.15.2.3 Consuming web services . . . 172
1.15.2.4 Publishing web services . . . 173
1.15.2.5 Using request and response headers . . . 174
1.15.2.6 Handling complex data types . . . 174
1.15.2.7 Troubleshooting SOAP requests and responses . . . 174
1.15.3 Using ColdFusion Web Services . . . 175
1.15.3.1 WorkFlow for a ColdFusion web service . . . 175
1.15.3.2 Enable ColdFusion services . . . 175
1.15.3.3 AccessColdFusionservicesusingSOAP . . . 175
1.15.3.4 Access ColdFusion services from a PHP server . . . 175
1.15.3.5 Access a ColdFusion service from .NET . . . 175
1.15.3.6 Web service enhancements in ColdFusion 10 . . . 175
1.15.3.7 RESTful Web Services in ColdFusion Developing Guide . . . 176
1.15.4 Integrating JEE and Java Elements in CFML Applications . . . 176
1.15.4.1 About ColdFusion, Java, and JEE . . . 176
1.15.4.2 Interoperating with JSP pages and servlets . . . 176
1.15.4.3 Using JSP tags and tag libraries . . . 177
1.15.4.4 Enhanced Java integration in ColdFusion 10 . . . 177
1.15.4.5 Using Java objects . . . 178
1.15.5 Using Microsoft .NET Assemblies . . . 180
1.15.5.1 About ColdFusion and .NET . . . 180
1.15.5.2 Accessing .NET assemblies . . . 180
1.15.5.3 Using .NET classes . . . 181
1.15.5.4 .NET Interoperability Limitations . . . 182
1.15.5.5 Example applications . . . 182
1.15.5.6 Advanced tools . . . 182
1.15.6 Integrating COM and CORBA Objects in CFML Applications . . . 183
1.15.6.1 About COM and CORBA . . . 183
1.15.6.2 Creating and using objects . . . 183
1.15.6.3 Getting started with COM and DCOM . . . 183
1.15.6.4 Creating and using COM objects . . . 183
1.15.6.5 Getting started with CORBA . . . 185
1.15.6.6 Creating CORBA objects . . . 185
1.15.6.7 Using CORBA objects in ColdFusion . . . 185
1.15.6.8 Handling CORBA object method exceptions . . . 185
1.15.6.9 CORBA example . . . 186
1.16 Using External Resources . . . 186
1.16.1 Sending and Receiving E-Mail . . . 186
1.16.1.1 Using ColdFusion with mail servers . . . 186
1.16.1.2 Sending e-mail messages . . . 186
1.16.1.3 Sample uses of the cfmail tag . . . 186
1.16.1.4 Using the cfmailparam tag . . . 187
1.16.1.5 Receiving e-mail messages . . . 187
1.16.1.6 Handling POP mail . . . 187
1.16.2 Interacting with Microsoft Exchange Servers . . . 188
1.16.2.1 Using ColdFusion with Microsoft Exchange servers . . . 188

1.16.2.2 Managing connections to the Exchange server . . . 188
1.16.2.3 Creating Exchange items . . . 189
1.16.2.4 Getting Exchange items and attachments . . . 189
1.16.2.5 Modifying Exchange items . . . 190
1.16.2.6 Deleting Exchange items and attachments . . . 190
1.16.2.7 Working with meetings and appointments . . . 190
1.16.2.8 Connect to Microsoft Exchange Server 2010 . . . 191
1.16.3 Interacting with Remote Servers . . . 192
1.16.3.1 About interacting with remote servers . . . 192
1.16.3.2 Using cfhttp to interact with the web . . . 192
1.16.3.3 Creating a query object from a text file . . . 192
1.16.3.4 Using the cfhttp Post method . . . 192
1.16.3.5 Performing file operations with cfftp . . . 193
1.16.4 Managing Files on the Server . . . 193
1.16.4.1 About file management . . . 193
1.16.4.2 Using cffile . . . 194
1.16.4.3 Using cfdirectory . . . 195
1.16.4.4 Using cfcontent . . . 195
1.16.5 Using Event Gateways . . . 195
1.16.5.1 About event gateways . . . 195
1.16.5.2 Event gateway facilities and tools . . . 196
1.16.5.3 Structure of an event gateway application . . . 196
1.16.5.4 Configuring an event gateway instance . . . 196
1.16.5.5 Developing an event gateway application . . . 196
1.16.5.6 Deploying event gateways and applications . . . 197
1.16.5.7 Using the CFML event gateway for asynchronous CFCs . . . 197
1.16.5.8 Using the example event gateways and gateway applications . . . 197
1.16.6 Using the Instant Messaging Event Gateways . . . 198
1.16.6.1 About ColdFusion and instant messages . . . 198
1.16.6.2 Configuring an IM event gateway . . . 198
1.16.6.3 Handling incoming messages . . . 198
1.16.6.4 Sending outgoing messages . . . 199
1.16.6.5 Sample IM message handling application . . . 199
1.16.6.6 Using the GatewayHelper object . . . 199
1.16.7 Using the SMS Event Gateway . . . 200
1.16.7.1 About SMS and ColdFusion . . . 200
1.16.7.2 Configuring an SMS event gateway . . . 201
1.16.7.3 Handling incoming messages-SMS event gateway . . . 201
1.16.7.4 Sending outgoing messages-event gateway . . . 201
1.16.7.5 ColdFusion SMS development tools . . . 202
1.16.7.6 Sample SMS application . . . 202
1.16.8 Using the FMS event gateway . . . 202
1.16.8.1 About Flash Media Server . . . 202
1.16.8.2 How ColdFusion and Flash Media Server interact through the FMS gateway . . . 203
1.16.8.3 Application development and deployment process . . . 203
1.16.9 Using the Data Services Messaging Event Gateway . . . 203
1.16.9.1 About Flex and ColdFusion . . . 203
1.16.9.2 Configuring a Data Services Messaging event gateway . . . 203
1.16.9.3 Sending outgoing messages to a Flex application . . . 203
1.16.9.4 Handling incoming messages from a Flex application . . . 203
1.16.9.5 Data translation . . . 204
1.16.10 Using the Data Management Event Gateway . . . 204
1.16.10.1 About ColdFusion and Flex . . . 204
1.16.10.2 Configuring a Data Management event gateway . . . 204
1.16.10.3 Sending messages . . . 204
1.16.10.4 Data translation - Developing guide . . . 204
1.16.11 Creating Custom Event Gateways . . . 204
1.16.11.1 Event gateway architecture . . . 204
1.16.11.2 Event gateway elements . . . 205
1.16.11.3 Building an event gateway . . . 205
1.16.11.4 Deploying an event gateway . . . 206
1.16.12 Using the ColdFusion Extensions for Eclipse . . . 206
1.16.12.1 About the ColdFusion Extensions for Eclipse . . . 206
1.16.12.2 Eclipse RDS Support . . . 206
1.16.12.3 ColdFusion/Flex Application wizard . . . 207
1.16.12.4 ColdFusion/Ajax Application wizard . . . 207
1.16.12.5 ActionScript to CFC wizard . . . 207
1.16.12.6 CFC to ActionScript wizard . . . 207
1.16.12.7 RDS CRUD wizard . . . 207
1.16.12.8 Services Browser . . . 207

Developing Adobe ColdFusion Applications

The Developing ColdFusion Application guide provides tools for developing Internet applications using Adobe ColdFusion.

- Changes in ColdFusion
- Introduction to application development using ColdFusion
- Introducing ColdFusion
- The CFML Programming Language
- Building Blocks of ColdFusion Applications
- Developing CFML Applications
- Accessing and Using Data
- ColdFusion ORM
- ColdFusion and HTML 5
- Flex and AIR Integration in ColdFusion
- Requesting and Presenting Information
- Office file interoperability
- ColdFusion Portlets
- Working with Documents, Charts, and Reports
- Using Web Elements and External Objects
- Using External Resources

© 2014 Adobe Systems Incorporated. All rights reserved.

Changes in ColdFusion

New in ColdFusion 11

ColdFusion 11 has gone through a lot of changes and enhancements and this section highlights those changes:

End-to-end mobile development

The ColdFusion Mobile Platform aims at providing a server and development infrastructure that facilitates rapid and robust mobile application development, debugging, packaging, and deployment. The ColdFusion 11 release introduces rapid application development through ColdFusion Builder 3. This release of ColdFusion introduces full-fledged on-device debugging to quickly debug your ColdFusion-based mobile applications on devices.

You can quickly build a mobile application by reading the information provided in the following sections:

- Building Mobile Applications
- Debugging Mobile Applications
- Inspecting Mobile Applications
- Packaging Mobile Applications
- Troubleshooting Mobile Applications
- Device Detection
- Client-side CFML
- Mobile Templates
- Getting Started Examples

A new lightweight edition

The **ColdFusion Getting Started Server** enables you to quickly setup a development or production instance of a Server without following the installation procedure. The ColdFusion Getting Started Server is a new ‘lighter’ version of the ColdFusion Server and is ideally suited for developers to quickly setup and run a ColdFusion Server for testing and debugging purposes.

See Installing the ColdFusion Express

Language enhancements

ColdFusion 11 has gone through various language enhancements that will provide a better development experience for ColdFusion developers. The core CFML language enhancements in ColdFusion 11 includes new language constructs, extended tag support, enhanced script functions, and support for new operations.

See ColdFusion Language Enhancements

Websocket enhancements

ColdFusion 11 has introduced the proxy support for WebSocket. There is a new proxy module (that runs inside IIS and Apache Web Server) that can intercept the ColdFusion WebSocket requests and redirect the requests to the ColdFusion Server.

See WebSocket enhancements

PDF generation enhancements

© 2014 Adobe Systems Incorporated. All rights reserved.

To generate high quality PDFs from HTML documents, 2 new tags, **<cfhtmltopdf>** and **<cfhtmltopdfitem>** have been introduced in ColdFusion 11.

See PDF generation enhancements

Security enhancements

ColdFusion 11 has a lot of security enhancements and bug fixes. This update fixes a few security issues and has strengthened the Server to a large extent. Some notable security enhancements are described in the following document:

See Security Enhancements (ColdFusion 11)

Social enhancements

ColdFusion 11 has introduced the support for dynamically generating Like button, Tweet button, and Comment box for social media sites.

See Social Enhancements

REST enhancements

ColdFusion 11 now supports site-level REST applications and enables pluggable serializer and deserializer.

See REST Enhancements in ColdFusion 11

Charting enhancements

The server-side charting introduced in ColdFusion 10 that allowed you to create highly interactive charts has been further enhanced to produce visually more appealing charts.

See Charting enhancements

Compression enhancements

The following sections describe the enhancements made to the <cfzip> and <cfzipparam> tags:

- cfzip
- cfzipparam

New functions

The following new functions are added:

- GetSafeHTML
- isSafeHTML
- ImageGetMetadata
- GeneratePBKDFKey

Changes in functions

The following functions are enhanced:

- Canonicalize

© 2014 Adobe Systems Incorporated. All rights reserved.

- cflocation
- EncodeForCSS
- EncodeForHTML
- EncodeForHTMLAttribute
- EncodeForJavaScript
- EncodeForURL
- EncodeForXML

Restrictions

When you are using ColdFusion Scheduler, output can be saved only as .log or .txt files. The same restriction is applicable for validation queries on databases.

Also, for the <cfinclude> tag, this restriction is applicable. By default, you can include only CFM files. However, you can modify allowedextinclude key in neoruntime.xml file to add your own file type.

Deprecated

- For the <cfsetting> tag, URL.RequestTimeout attribute has been removed in ColdFusion 11.
- The HTMLEditFormat() function has ben deprecated.

#back to top

© 2014 Adobe Systems Incorporated. All rights reserved.

Replacement of JRun with Tomcat

Instead of JRun, Tomcat is embedded with a stand-alone ColdFusion 10 installation.
Previous versions of ColdFusion installer let you create multi-server installations whereas with ColdFusion 10 installer, only stand-alone installation is possible. After installing ColdFusion in stand-alone mode, you can create multiple instances and clusters, provided you have an Enterprise or Developer license.
For details, *Installing Adobe ColdFusion*.

#back to top

© 2014 Adobe Systems Incorporated. All rights reserved.

Security enhancements

Security enhancements helps to reduce security vulnerabilities, particularly the ones resulting from threats posed by XSS and CSRF attacks.
The release also includes enhancements that help you manage ColdFusion sessions effectively.
For details, see Security enhancements in ColdFusion 10.

#back to top

© 2014 Adobe Systems Incorporated. All rights reserved.

ColdFusion WebSocket

Develop realtime applications for stock, charting, online gaming, social networking, dashboard for various purposes, or monitoring using COldFusion WebSocket.
ColdFusion implements WebSocket by providing a messaging layer for the WebSocket protocol, which you can easily control using CFML and JavaScript.
For details, see Using ColdFusion WebSocket.

#back to top

© 2014 Adobe Systems Incorporated. All rights reserved.

ColdFusion closures

For details, see Using closures.

#back to top

© 2014 Adobe Systems Incorporated. All rights reserved.

Enhanced Java integration

Load Java libraries from a custom path.
For details, see Enhanced Java integration in ColdFusion 10

#back to top

© 2014 Adobe Systems Incorporated. All rights reserved.

ColdFusion ORM search for indexing and search

Use the indexing and search capabilities of ColdFusion ORM.
When you develop an application that uses ColdFusion ORM, the search feature facilitates full text search. You can load all the persistent entities that match your search criteria based on a query text.
For details, see ColdFusion ORM search

#back to top

© 2014 Adobe Systems Incorporated. All rights reserved.

Solr enhancements

- Use Data Import Handler for database indexing
- Index and search based on dynamic custom fields
- Reload individual collections
- Add languages for search
- Secure your search system
- Autocommit indexed documents
- Boost specific fields or entire document for improved search results
  For details, see Solr enhancements in ColdFusion 10.

#back to top

© 2014 Adobe Systems Incorporated. All rights reserved.

Scheduler enhancements

Schedule your tasks in a granular, scalable, and organized way. The release supports Quartz scheduling service. For details, see Using Scheduler.

#back to top

© 2014 Adobe Systems Incorporated. All rights reserved.

Integration with Microsoft Exchange Server 2010

Adobe ColdFusion can interact with Microsoft Exchange Server 2010 SP1. The enhancements offer support for Microsoft Exchange Web Services (EWS) which brings in efficacy with the following operations:

- Folder operations such as create, modify, or delete.
- Get rooms and roomlist in the exchange organization.
- Information on user availability, that helps effective scheduling.
- Conversation operations such as find conversation details, copy, move, and the status if the conversation is read.
  For details, see Connect to Microsoft Exchange Server 2010.

#back to top

© 2014 Adobe Systems Incorporated. All rights reserved.

Lazy loading across client and server in ColdFusion

Need-based loading of related entities for applications that use ColdFusion ORM in the back end and Flex as the front end is possible in this release.
Your application can now fetch the main entity and not return the related entities. Only when the client application tries to access the related entities, they are loaded.
For details, see Lazy loading across client and server.

#back to top

© 2014 Adobe Systems Incorporated. All rights reserved.

Web service enhancements

ColdFusion 10 has Axis 2 Web service framework integrated. This enables your web services to use WSDL 2 specifications, SOAP 1.2 protocol, and document literal wrapped style.
Also the enhancements resolve many interoperability issues that you might encounter while working with Web services in ColdFusion 9.
For details, see Web service enhancements in ColdFusion 10.

#back to top

© 2014 Adobe Systems Incorporated. All rights reserved.

RESTful Web Services in ColdFusion

- What is REST
- REST and ColdFusion
- Creating the REST web service
  - Example
- Registering an application with REST service
  - Using ColdFusion Administrator (Data & Services > REST Services)
- Accessing the web service
  - Interpreting the URL
  - Providing accept header
- HTTP Content-type negotiation
  - Examples
- Specifying subresources
  - Resource function
  - Subresource function
    - Example
  - Subresource locator
    - Example
- HTTP Responses
  - Success responses
  - Error responses
  - Custom responses
    - Send custom success responses using restSetResponse
    - Send custom error response using cfthrow
- Modifications to Application.cfc
- Extending REST Web services
- REST services and data interchange formats
  - XML serialization for REST services
    - Serialization specifications
    - Deserialization specifications
    - Format definition
    - Query
    - Example
    - Struct
    - Example
    - CFC Component
    - Example
    - Format for array and array of cfcomponent
    - Example
    - Example: Array of CFC Serialization
    - Example: Array of CFC: Deserialization
    - string, boolean, numeric, binary, and date
    - Handling cyclic dependency
  - JSON serialization and REST services
    - Serialization specifications
    - Deserialization specifications
    - Format definitions

© 2014 Adobe Systems Incorporated. All rights reserved.

- Format for query
- Format for struct
- Format for component
- Format for array
- string, boolean, numeric, and date

- Support for GZip encoding

Site-level REST application support

- **Option 1: autoregister Application Setting**
- **Option 2: Registering a REST application using ColdFusion Administrator console**
- Option 3: Registering a REST application using the ColdFusion Admin API
- Option 4: Registering a REST application using the restInitApplication method

Support for pluggable serializer and deserializer

ColdFusion 10 lets you create and publish REST (Representational State Transfer) services that can be consumed by clients over HTTP/HTTPS request.

What is REST

The following URL takes you to the Java Tutorial that provides conceptual information on REST: http://download.oracle.com/javaee/6/tutorial/doc/gijqy.html

REST and ColdFusion

You can create REST services by defining certain attributes in the tags `cfcomponent`, `cffuntion`, and `cfargument` and publish as REST resources.

- *__Follows HTTP request-response model:__* Beyond having HTTP as a medium, the service lets you follow all HTTP norms. The components published as REST services can be consumed over HTTP/HTTPS request. The REST services are identified with URI (Uniform Resource Identifier) and can be accessed from a web page as well as by specifying the URI in the browser's address bar.
- **Supports all HTTP methods :** The REST enabled CFCs support the following HTTP methods: `GET`, `POST`, `PUT`, `DELETE`, `HEAD`, and `OPTIONS`.
- **Implicit handling of serialization/deserialization:** ColdFusion natively supports JSON and XML serialization/deserialization. So client applications can consume REST services by issuing HTTP/HTTPS request. The response can either be serialized to XML or JSON format.
- **Publish web service as both REST service and WSDL service:** You can create and publish the same ColdFusion component as a REST service and WSDL service.

Creating the REST web service

- You can create and publish a ColdFusion component or any functions in a component as REST resource.
- To create a CFC as REST web service, specify either of the following in the tag `cfcomponent`: `restPath` or `rest`.
- In `cffunction`, set the attribute `access` to `remote` for the functions that you have to expose as REST resource.

Example

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cfcomponent rest="true" restpath="restService">
 <cffunction name="sayHello" access="remote" returntype="String" httpmethod="GET">
  <cfset rest = "Hello World">
  <cfreturn rest>
 </cffunction>
</cfcomponent>
```

Registering an application with REST service

After you create the CFC you want to REST-enable, specify the folder for registering as web service either using the function `restInitAplication` or in the ColdFusion Administrator.

> **Note**
>
> Nested REST applications cannot be registered.

Using ColdFusion Administrator (Data & Services > REST Services)

When you specify a folder, all CFCs in that folder or subfolders for which you have specified `rest` or `restPath` are registered.

1. Browse and select the application path or root folder where ColdFusion would search for CFCs.
2. (Optional) In the Service Mapping section, specify virtual mapping in place of application name.If the folder has an Application.cfc and an application name, then the service is identified with the application name. You can override this by specifying the service mapping. In this case, the service is identified with the service mapping that is provided. If there is no Applicaiton.cfc in the folder, then it is mandatory to specify the Service mapping.If you choose to specify REST folder outside of ColdFusion root, then add one of the following mappings to register the folder:Consider that you have added a mapping for c:\restfolder as \map

   ```
   <cfset restinit("c:\restfolder","myapp")>
   <cfset restinit("\map","myapp")>
   ```

3. (Optional) Specify an application as default REST service. Only one application can be set as default for a server instance. You can change the default application at any time. Check Set the default application and then click Add Service. To remove the service as default, uncheck it.
4. After you specify the details, click Add Service to register.The Active ColdFusion REST Services section specifies the details of all registered web services.
   After you register, all CFCs are published as RESTful services. On subsequent startups, the registered services automatically get published.

   > **Note**
   >
   > Refresh the application whenever there is a change in REST-related component in the application.

Accessing the web service

- Use the tag `cfhttp`to access the web service as shown in the following example:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cfhttp
 url="http://localhost:8500/rest/RestTest/restService"
 method="get"
 port="8500"
 result="res">
</cfhttp>
```

- Access using the browser by providing the URL: http://localhost:8500/rest/RestTest/restService

Interpreting the URL

The URL provided in the example can be interpreted as follows:

URL component	Description
http://localhost:8500	Base URL which includes the IP address and port of the ColdFusion server.If you deploy ColdFusion as a JEE application, the URL will contain a context root, for example, http://localhost:8500*/cfusion* .
rest	Implies that the request sent is a REST request.This default value can be renamed by revising the context path in `web.xml` available at `cfusion/wwroot/WEB-INF` and update the same mapping in `uriworkermap.properties` file found at `config\wsconfig\1`.
restTest	Application name or service mapping that you have used while registering the service in ColdFusion Administrator. If you do not specify a service mapping in the ColdFusion Administrator, then the application name is taken from Application.cfc.
restService	Rest path you defined in the service. That is, the value of the attribute `restPath` in the tag `cfcomponent`.

Providing accept header

The accept header can be provided in the REST URL. The following examples illustrate this:

- For the REST URL http://localhost:8500/rest/RestTest/restService.json, the accept parameter would be set to `application\json`.
- For the REST URL http://localhost:8500/rest/RestTest/restService.xml, the accept parameter would be set to `application\xml`.

HTTP Content-type negotiation

© 2014 Adobe Systems Incorporated. All rights reserved.

The content-type that is returned by a RESTful web service depends on the Accept HTTP header.
Client can set an Accept request header in the following order:

- Comma-separated list of preferred content types.
- (Followed by a semi-colon (;) A floating point number in the range 0 through 1 in the format q=*0-1*. The default value is 1.0 is the least value which indicates that the content type is not acceptable whereas 1 is the maximum value with the highest priority.
  If two types are provided the same priority, then the sequential priority is considered.

Examples

In the following example, client can use both XML and JSON format as no value is specified to indicate a priority in the Accept header:

```
GET http://adobe.com/stuff
Accept: application/xml, application/json
```

The request in the following example specifies a priority for returning the content-type:

```
GET http://adobe.com/stuff
Accept: text/*;q=0.9, */;q=0.1, audio/mpeg, application/xml;q=0.5
```

The order of precedence for content type is as follows:

1. audio/mpeg (as no priority is specified)
2. text/* (0.9 is the highest value)
3. application/xml
4. */
   In the following example, though you have specified same priority for both text/* and audio/mpeg, text/* gets precedence because of the sequence.

   ```
   GET http://adobe.com/stuff
   Accept: text/*;q=0.9, */;q=0.1, audio/mpeg=0.9, application/xml;q=0.5
   ```

Specifying subresources

Functions in a REST service can either be a resource function, subresource function, or subresource locator.

Resource function

The functions for which you have not specified the `RestPath` at function level but specified {{httpMethod.}}In this case, the resource function is invoked when URL of the CFC is accessed.

Subresource function

The functions for which you have specified both `resptPath` and `httpMethod`.
Subresource functions are used to create subresources for the resource created by CFC. If the subresource has `http tpMethod` attribute in `cffunction`, the function can directly handle HTTP requests as shown in the following

© 2014 Adobe Systems Incorporated. All rights reserved.

example.

Example

Employee.cfc

```
<cfcomponent rest="true" restPath="/hello">
       <cffunction name="sayHello" access="remote" returnType="String"
httpMethod="GET" restPath="name">
---------
 </cffunction>
</cfcomponent>
```

In this example, `httpMethod` and `restPath` are defined. The *`baseurl`*`/hello/name` is a subresource to the URL *`baseurl`*`/hello.`

Subresource locator

If you have not specified the `httpMethod` attribute, but have specified `restPath`, you can dynamically resolve the component that handles the request. Any returned object is treated as a resource class instance and used to either handle the request or to further resolve the object that handles the request.

Example

In this example, StudentService.cfc and Student.cfc are the two REST resources. In StudentService.cfc the function `getStudent` returns a component. In the function, the object of Student is created and the values name and age are set. When the object is specified in the return type, the function defined in the object with the requested `httpmethod` is invoked.StudentService.cfc can be invoked directly as it has a `restPath` specified. Student.cfc can only be invoked through StudentService.cfc because the function `getStudent` in StudentService.cfc is acting as the subresource locator and returns Student.cfc.StudentService.cfc:

```
<cfcomponent rest="true" restpath="/studentService">

 <cffunction name="addStudent" access="remote" returntype="void" httpmethod="POST">
  <cfargument name="name" type="string" required="yes" restargsource="Form"/>
  <cfargument name="age" type="numeric" required="yes" restargsource="Form"/>
   <!--- Adding the student to data base. --->
 </cffunction>
  <cffunction name="getStudent" access="remote" returntype="student"
restpath="{name}-{age}">
  <cfargument name="name" type="string" required="yes" restargsource="Path"/>
  <cfargument name="age" type="string" required="yes" restargsource="Path"/>
   <!--- Create a student object and return the object. This object will handle the
request now. --->
  <cfset var myobj = createObject("component", "student")>
  <cfset myobj.name = name>
  <cfset myobj.age = age>
  <cfreturn myobj>
 </cffunction>
</cfcomponent>
```

Student.cfc

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cfcomponent>
 <cfproperty name="name" type="string"/>
 <cfproperty name="age" type="numeric"/>
 <!--- Getting the student detail. --->
 <cffunction name="getStudent" access="remote" returntype="String" httpmethod="GET"
produces="text/xml">
  <!--- Retrieve the Student from the DB. --->
  <!--- get student from db where name and age matches --->
  <cfset st.name = "Thomas">
  <cfset st.age = "25">
  <cfset st.surname = "Paul">
  <cfset str = "<student><name>" & st.name & "</name><age>" & st.age &
"</age><surname>" & st.surname & "</surname></student>">
  <cfreturn str>
 </cffunction>
 <!--- Updating the student detail. --->
 <cffunction name="updateStudent" access="remote" returntype="void"
httpmethod="PUT">
  <!--- Retrieve the Student from the DB. --->
  <!--- Update the student in DB. --->
  <cfset st.name = name>
  <cfset st.age = age>
 </cffunction>
 <!--- Deleting the student. --->
 <cffunction name="deleteStudent" access="remote" returntype="String"
httpmethod="DELETE">
  <!--- Delete the Student from the DB. --->
  <!---<cfset st = deleteStudentFromDB(name)>--->
  <cfreturn "Student deleted">
 </cffunction>
</cfcomponent>
```

Student.cfm

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<!--- adding the student --->
<cfhttp url="http://localhost:8500/rest/RestTest/studentService" method="post"
result="res">
<cfhttpparam type="formfield" name="name" value="Thomas" >
<cfhttpparam type="formfield" name="age" value="25" >
</cfhttp>
<cfoutput>Student Added</cfoutput>
</br>
</br>
<!--- fetching the details --->
<cfhttp url="http://localhost:8500/rest/RestTest/studentService/Thomas-25"
method="get" result="res">
</cfhttp>
<cfoutput>#xmlformat(res.filecontent)#</cfoutput>
</br>
</br>
<!--- updating the student details --->
<cfhttp url="http://localhost:8500/rest/RestTest/studentService/Thomas-25"
method="put" result="res">
</cfhttp>
<cfoutput>Updated the student</cfoutput>
</br>
</br>
<!--- deleting the student --->
<cfhttp url="http://localhost:8500/rest/RestTest/studentService/Thomas-25"
method="delete" result="res">
</cfhttp>
<cfoutput>Student Deleted</cfoutput>
```

HTTP Responses

By default, ColdFusion provides default HTTP success and failure responses to the client as follows:

Success responses

Default Response	Description
`200 OK`	Sent if the response has body.
`204 No Content`	Sent if the response does not have body.

Error responses

Default Response	Description
`404 Not found`	Request URL is not valid.
`406 Not Acceptable`	No function in the REST service can produce the MIME type requested by the client.

© 2014 Adobe Systems Incorporated. All rights reserved.

415 Unsupported Media Type Error	A resource is unable to consume the MIME type of client request.
405 Method not allowed	If the client invokes an HTTP method on a valid URI to which the request HTTP method is not bound.This error does not appear if the client requests for HTTP `HEAD` a nd `OPTIONS` methods.If a resource method that can service `HEAD` requests for that particular URI is not available, but there exists a method that can handle `GE T`, it is invoked and returns the response from that method without the request body. If there is no existing method that can handle `OPTIONS`, a meaningful automatically generated response is sent along with the Allow header set.

Custom responses

In addition to the default responses available with ColdFusion, you can set custom responses.
For example, assume that you have to provide a success response `201 Created`. Such a default response does not exist. You can only have `200 OK` or `204 No Content` to send.
In such scenarios, you can create custom responses in either of the following ways:

Send custom success responses using restSetResponse

1. When you define the CFC as REST service, ensure that the `returnType` is set to `void` in the `cffunction`(the function for which you want to send the custom response).For example,

```
<cffunction name="create" httpMethod="POST" produces="application/xml"
returnType="void">
 <cffunction>
```

2. Create a struct for the custom response that you want to send in the `cffunction`as shown in the following example:

```
<cfset response=structNew()>
<cfset response.status=201>
<cfset response.content="<customer
id="&id&"><name>"&name&"</name></customer>">        <cfset
response.headers=structNew()>
<cfset
response.headers.location="http://localhost:8500/rest/CustomerService/customer
s/123">
```

   In this example, you have set `201` as the status of the response, content that you want to send. For example, customer details and the location where you have created the resource for the `POST`request.

© 2014 Adobe Systems Incorporated. All rights reserved.

> **Note**
>
> If you do not specify the status in the custom response, `500 Internal server error` is sent as the response status.

3. Use the function `restSetResponseas` follows:

```
restSetResponse( response );
```

Send custom error response using cfthrow

Assume that you want to send a custom error response. For example, consider the following:method.service.cfc

```
<cffunction name="getCustomer" httpMethod="GET" produces="application/xml"
restPath="{id}" return="string">
 <cfargument name="id" type="numeric" argtype="PathParam">
<!--- Getting the customer. ---><cffunction>
```

In this case, you have a customer database and you are doing a `GET` HTTP request on `/customers/123`.But you find that the customer with the specified ID `123` is not available. So you want to send `404 Resource Not Found` response back to the client, which is not possible by default.In this case, you can use `cfthrow` to send custom error response as follows:

```
<cfthrow type="RestError" errorcode="404">
```

Modifications to Application.cfc

The following enhancements have been made to Application.cfc for REST support:

Variable	Description
`this.restsettings.cfclocation`	To publish the CFCs only in a particular location, provide comma-separated list of directories where the REST CFCs are located. The directory paths can be absolute or relative.If not set, all the CFCs from the application root are published.

© 2014 Adobe Systems Incorporated. All rights reserved.

`this.restsettings.skipCFCWithError`	When an error occurs, continue publishing, ignoring the CFC that has caused the exception.If `true`, the CFC with error is ignored and the rest of the CFCs are published. By default it is false.If set to `false`, in case of an error, the application itself is not published. But other registered application are published.If an error occurs during application startup, the error is printed in console.Each application has separate log files for logging the issues.

Extending REST Web services

The following conditions apply when you extend the RESTful CFCs:

- You can define the REST attributes for functions in the base CFC. So all the CFCs that extend from the base CFC inherits the REST attributes.In the following example, CustomerResource.cfc extends BaseCustomerResource.cfc:BaseCustomerResource.cfc

```
<cfcomponent>
 <cffunction name="SayPlainHello" access="remote" produces="text/plain"
returntype="string" httpmethod="POST">
  <cfreturn "BaseCustomerResource Plain">
 </cffunction>
 <cffunction name="SayXMLHello" access="remote"
produces="text/xml,application/xml" returntype="string" httpmethod="POST">
  <cfreturn "BaseCustomerResource XML">
 </cffunction>
</cfcomponent>
```

  BaseCustomerResource.cfc has all REST attributes applied to functions within the CFC. After you define BaseCustomerResource.cfc, you can define CustomerService.cfc that extends BaseCustomerResource.cfc as follows:Customerservice.cfc

```
<cfcomponent rest="true" restpath="/customerservice"
extends="BaseCustomerResource">

 <cffunction name="SayPlainHello" access="remote" returntype="string">

  <!--- Implement the method. --->
  <cfreturn "CustomerResource Plain">
 </cffunction>

 <cffunction name="SayXMLHello" access="remote" returntype="string">

  <!--- Implement the method. --->
  <cfreturn "CustomerResource XML">
 </cffunction>

</cfcomponent>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

BaseCustomerResource.cfm

```
<cfhttp url="http://localhost:8500/rest/RestTest/customerservice"
method="post" result="res">
<cfhttpparam type="header" name="accept" value="text/plain" >
</cfhttp>
<cfoutput>#res.filecontent#</cfoutput>
</br>
</br>
<cfhttp url="http://localhost:8500/rest/RestTest/customerservice"
method="post" result="res">
<cfhttpparam type="header" name="accept" value="application/xml" >
</cfhttp>
<cfoutput>#res.filecontent#</cfoutput>
</br>
</br>
```

Except the `rest` and `restPath`, no other REST attributes are required within CustomerService.cfc.

- When you inherit a RESTful CFC, you can choose to override the attributes used in the CFC that you extend. For example, the following CFC extends the BaseCustomerResource.CFC, but the function `SayPlainHello` overrides the function in Base CFC.

```
<cfcomponent rest="true" restPath="/customerservice"
extends="BaseCustomerResource">
 <cffunction name="SayPlainHello" access="remote" produces="text/plain"
returntype="string" httpmethod="PUT">
  <cfargument name="username" type="string" argtype="pathparam">
  <!--- Implement the method. --->
 </cffunction>


 <cffunction name="SayXMLHello" access="remote" returntype="string">
  <cfargument name="username" type="string">
  <!--- Implement the method. --->
 </cffuntion>
</cfcomponent>
```

Note

Even if you override only one attribute in a function, you have to respecify all REST attributes

- You have to specify the REST attributes (`rest`/`restPath`) in the derived CFC to use it as a REST service. Just that it is defined in the base CFC does not work.

REST services and data interchange formats

REST services in ColdFusion can be represented in XML or JSON formats for transmitting over the web. ColdFusion takes care of the serialization/deserialization of the ColdFusion data types used in the services to XML or JSON implicitly.
A function that is REST-enabled can take the following ColdFusion data types as arguments: query, struct, CFC

© 2014 Adobe Systems Incorporated. All rights reserved.

type, array, array of CFCs, XML, string, numeric, boolean, date, and binary (for XML).
ColdFusion serializes data to pre-defined formats of XML and JSON. Similarly, ColdFusion deserializes the body only if the body is in the format defined by ColdFusion.

XML serialization for REST services

Serialization specifications

- The Accept header of the request has to be either `text/xml` or `application/xml`.
- There has to be a function in the service that can produce the required MIME types.
- The function has to return any of the ColdFusion supported data types.
- Cyclic arrays are not supported. You might see the serialized string published, but not with the expected output as explained in the following example:

```
<cfset this.arr1 = arrayNew(1)>
<cfset this.arr1[1] = "1">
<cfset this.arr1[2] = this.arr1>
<cfset this.arr1[3] = "3">
```

When an array is assigned to a variable, in this case, `<cfset this.arr12 = this.arr1>`, you assign a `rr1` as the item in the second index of `arr1`. ColdFusion implicitly creates a new array and copies the data from `arr1` to the newly created array. The newly created array is assigned to the second index of `arr1`. Therefore, both the instances are different, and therefore cyclic dependency is impacted.When you serialize, you get the following output:

```
<array id="1" size="3">
<item index="1" type="string">1</item>
<item index="2" type="array">
<array id="2" size="1">
<item index="1" type="string">1</item>
</array>
</item>
<item index="3" type="string">3</item>
</array>
```

As you can observe, the inner array gets truncated after the first element.

Deserialization specifications

- The content of the request has to be in a pre-defined format specified by ColdFusion (see details in the section Format definition).
- The content type of the request has to be either `text/xml` or `application/xml`.
- There has to be a function in the service that can consume the MIME type of the request.
- `cfargument` cannot have the attributes `restArgSource` and `restArgName` specified. That is, you can only send data in the body of the request.
- `cfargument` type should be ColdFusion supported data type
- There can only be one argument that does not specify the `restArgSource` attribute. The whole body of the request is deserialized to the argument type.
- Cyclic arrays are not supported.

Format definition

© 2014 Adobe Systems Incorporated. All rights reserved.

Query

- The root element of the XML has to be `query`.
- Use the attribute `ID` to handle circular dependency.
- The following are the valid attribute values: string, date, boolean, numeric, document, query, struct, array, and CFC (if the value is a CFC instance).

Example

In the following example, The root `query` element has two child elements `columnnames` and `rows`. `columnnames` is the name of the columns in the query. `rows` can have data in multiple `row` elements. Here, the order of the column in the row should match the column defined in the `columnames`. For each column in the row, a `type` attribu te is mandatory. The `type` attribute can have any of the ColdFusion data types as values.

```
<query id="1">
    <columnnames>
        <column name="columnnameone"/>
        <column name="columnnametwo"/>
    </columnnames>
    <rows>
        <row>
            <column type="string">value one</column>
            <column type="string">value two</column>
        </row>
        <row>
            <column type="number">444.0</column>
            <column type="string">value four</column>
        </row>
    </rows>
</query>
```

Struct

- The `root` element of the XML should be struct.
- Use the attribute `ID` to handle circular dependency.
- The struct can have multiple `entry` child elements. That is, a key-value pair in the Struct instance to be serialized. `entry` element requires two mandatory attributes `name` (name of the entry) and `type` (the type of the value of the entry). The `type` attribute can have one of the ColdFusion data types as values.

Example

```
<struct id="1">
    <entry name="name" type="string">joe</entry>
    <entry name="age" type="string">30</entry>
    <entry name="address" type="string">101 some drive, pleasant town, 90010</entry>

    <entry name="eyecolor" type="string">blue</entry>
    <entry name="income" type="string">50000</entry>
</struct>
```

CFC Component

- The root element of the XML has to be `component`.

© 2014 Adobe Systems Incorporated. All rights reserved.

- `ID` attribute is used to handle circular dependency.
- The `name` attribute has to provide the fully qualified name of the CFC from webroot.
- A `component` element can contain multiple `property` child elements.
- The `property` elements are the `cfproperty` values defined in `cfcomponent`.
- The `name` attribute of the `property` corresponds to the name of the `cfproperty` defined.
- `type` attribute specifies the type of the `cfproperty`.
- If, in the component, any property has a null value, then it does not appear in the serialized format. This applies also for deserialization.

Example

Student.cfc

```
<component id="1" name="testrest.student">
    <property name="name" type="string">paul</property>
    <property name="age" type="number">444.0</property>
    <property name="dob" type="date">478377000000</property>
</component>
```

In the following example, `testrest.student` means the CFC Student.cfc is placed in the testrest directory under webroot.

Format for array and array of cfcomponent

- The root element has to be `array`.
- `size` attribute specifies the length of the array.
- The `type` attribute is not mandatory for the `array` element whereas for `item` element, it is mandatory.
- If the array is a `cfcomponent` array, then the `type` attribute of `array` element should have the fully qualified name of the CFC.
- The `array` element contains multiple `item` child elements. Only the non-null items in the array are serialized.
- The `item` element has two attributes: `index` that specifies that index of the item in the array and `type`.

Example

The following example shows an array containing two struct objects:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<array id="1" size="2">
    <item index="0" type="struct">
        <struct id="2">
            <entry name="name" type="string">joe</entry>
            <entry name="age" type="string">30</entry>
            <entry name="address" type="string">101 some drive, pleasant town,
90010</entry>
            <entry name="eyecolor" type="string">blue</entry>
            <entry name="income" type="string">50000</entry>
        </struct>
    </item>
    <item index="1" type="struct">
        <struct id="3">
            <entry name="name" type="string">paul</entry>
            <entry name="age" type="string">25</entry>
            <entry name="address" type="string">some other address</entry>
            <entry name="eyecolor" type="string">black</entry>
            <entry name="income" type="string">40000</entry>
        </struct>
    </item>
</array>
```

Example: Array of CFC Serialization

The following example illustrates serializing array of CFC to XML and JSON formats.

1. Create an array of arrayCFCdefinition.cfc:

```
<cfcomponent>
 <cfproperty name="str" type="string"/>
</cfcomponent>
```

2. arrayCFC.cfc produces the required array as follows:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cfcomponent restpath="arrayOfCFC">
 <cffunction name="func1" access="remote" output="false"
returntype="arrayCFCdefinition[]"
             httpmethod="get" produces="text/xml">
  <cfset arrCFC = arraynew(1)>
  <cfloop from=1 to=2 index="i">
   <cfset obj = createObject("component", "arrayCFCdefinition")>
   <cfset obj.str = i>
   <cfset arrayAppend(arrCFC, obj)>
  </cfloop>
  <cfreturn arrCFC>
 </cffunction>
 <cffunction name="func2" access="remote" output="false"
returntype="arrayCFCdefinition[]"
             httpmethod="get" produces="text/json">
  <cfset arrCFC = arraynew(1)>
  <cfloop from=1 to=2 index="i">
   <cfset obj = createObject("component", "arrayCFCdefinition")>
   <cfset obj.str = i>
   <cfset arrayAppend(arrCFC, obj)>
  </cfloop>
  <cfreturn arrCFC>
 </cffunction>
</cfcomponent>
```

3. Do the following to access the resource:
   - For XML:

```
<cfhttp url="http://127.0.0.1:8500/rest/RestTest/arrayOfCFC" method="get"
result="res1">
 <cfhttpparam type="header" name="accept" value="text/xml">
</cfhttp>
```

   - For JSON:

```
<cfhttp url="http://127.0.0.1:8500/rest/RestTest/arrayOfCFC" method="get"
result="res2">
 <cfhttpparam type="header" name="accept" value="text/json">
</cfhttp>
```

4. You receive the following serialized output as response:
   - For XML:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<array id="1" size="2"
type="cfsuite.restservices.restservices.new.ArrayCFCdefinition">
 <item index="1" type="COMPONENT">
  <component id="2"
name="cfsuite.restservices.RESTServices.New.arrayCFCdefinition">
   <property name="STR" type="NUMBER">
    1.0
   </property>
  </component>
 </item>
 <item index="2" type="COMPONENT">
  <component id="3"
name="cfsuite.restservices.RESTServices.New.arrayCFCdefinition">
   <property name="STR" type="NUMBER">
    2.0
   </property>
  </component>
 </item>
</array>
```

- For JSON:

```
[{"Str":1},{"Str":2}]
```

Example: Array of CFC: Deserialization

The following example illustrates deserializing array of CFC from XML format.

> **Note**
>
> Deserializing array of CFC is unsupported for JSON.

1. Create an array of arrayCFCdefinition.cfc:

```
<cfcomponent>
 <cfproperty name="str" type="string"/>
 <cffunction name="check" returntype="any">
  <cfreturn this.str>
 </cffunction>
</cfcomponent>
```

2. arrayCFC.cfc produces the required array as follows:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cfcomponent>
 <cffunction name="func3" access="remote" output="false" returntype="string"
             httpmethod="put" produces="text/xml" consumes="text/xml">
  <cfargument name="arg" type="arrayCFCdefinition[]"/>

  <cfif arg[2].check() eq "2">
   <cfreturn "true">
  <cfelse>
   <cfreturn "false">
  </cfif>
 </cffunction>
</cfcomponent>
```

3. Do the following to access the resource for XML:

```
<cfhttp url="http://127.0.0.1:8500/rest/RestTest/arrayOfCFC" method="put"
result="res3">
 <cfhttpparam type="header" name="content-type" value="text/xml">
 <cfhttpparam type="header" name="accept" value="text/xml">
 <cfhttpparam type="body" value="<ARRAY ID=""1"" SIZE=""2""
TYPE=""cfsuite.restservices.restservices.new.ArrayCFCdefinition""><ITEM
INDEX=""1"" TYPE=""COMPONENT""><COMPONENT ID=""2""
NAME=""cfsuite.restservices.restservices.new.ArrayCFCdefinition""><PROPERTY
NAME=""STR"" TYPE=""NUMBER"">1.0</PROPERTY></COMPONENT></ITEM><ITEM
INDEX=""2"" TYPE=""COMPONENT""><COMPONENT ID=""3""
NAME=""cfsuite.restservices.restservices.new.ArrayCFCdefinition""><PROPERTY
NAME=""STR"" TYPE=""NUMBER"">2.0</PROPERTY></COMPONENT></ITEM></ARRAY>">
</cfhttp>
```

4. Refer to the function. You are verifying the value of the property of arrayCFC definition.cfc for the second index of the array.

string, boolean, numeric, binary, and date

Specify the values directly in the body of the request.

Handling cyclic dependency

In ColdFusion, cyclic dependency is handled using the ID reference. All ColdFusion complex data types have unique IDs when serialized. If the same object has to be serialized elsewhere, instead of serializing the object again, a reference is made to the already serialized data using its `ID`. In the following example, the main object is a struct. The struct contains an array of objects. The array has two elements and both the elements are the same instance of a struct. During serialization, the first element in the array is serialized as it is. The `ID` of the serialized struct is `2`. Instead of serializing the second element, as that object is already serialized, `IDREF` attribute is used to refer to the already serialized struct instance.

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<struct id="1">
    <entry name="arrayinastruct" type="array">
        <array id="2" size="2">
            <item index="0" type="struct">
                <struct id="3">
                   <entry name="name" type="string">joe</entry>
                    <entry name="age" type="string">30</entry>
                    <entry name="address" type="string">101 some drive, pleasant
town, 90010</entry>
                    <entry name="eyecolor" type="string">blue</entry>
                    <entry name="income" type="string">50000</entry>
                </struct>
            </item>
            <item index="1" type="struct">
                <struct idref="3"/>
            </item>
        </array>
    </entry>
</struct>
```

Note

Object reference is taken care of by ColdFusion at the time of deserialization also.

JSON serialization and REST services

Serialization specifications

- The content type of the Accept header of the request has to be `text/JSON`, `application/JSON`, or `text/plain`.
- REST service should have a function that can handle the required MIME types.
- Function has to return any of the ColdFusion supported data types other than binary.
- Cyclic behavior is unsupported. But in the case of arrays, you might see the serialized string published, but not with expected output as explained in the following example:

```
<cfset this.arr1 = arrayNew(1)>
<cfset this.arr1[1] = "1">
<cfset this.arr1[2] = this.arr1>
<cfset this.arr1[3] = "3">
```

When an array is assigned to a variable (in this case) `<cfset this.arr12 = this.arr1>`, you assign a `rr1` as the item in the second index of `arr1`. ColdFusion implicitly creates a new array and copies the data from `arr1` to the newly created array. The newly created array is assigned to the second index of `arr1`. Therefore, both the instances are different, and therefore cyclic dependency is impacted.When you serialize, you get the following output:

```
[1,[1],3]
```

© 2014 Adobe Systems Incorporated. All rights reserved.

As you can observe, the inner array gets truncated after the first element.

Deserialization specifications

- The content of the request is in a predefined format specified by ColdFusion.
- The content type of the request is `text/JSON`, `application/JSON`, or `text/plain`.
- A function in the service consumes the MIME type of the request.
- `cfargument` does not have the attributes `restargsource` and `restargname` specified.
- `cfargument` type is ColdFusion supported data type other than binary or CFC definition.
- Only one argument has no `restAargSource` attribute specified. The whole body of the request is deserialized to the argument type.
- Cyclic behavior is unsupported. But in the case of cyclic arrays, you might see the deserialized array published, but not giving expected output.

Format definitions

Format for query

```
{'COLUMNS':['columnNameOne','columnNameTwo'],'Data':[['value one','value
two'],['444.0','value four']]}
```

Format for struct

```
{'NAME':'joe','AGE':30,'ADDRESS':'101 Some Drive, Pleasant Town,
90010','EYECOLOR':'blue','INCOME':50000}
```

Format for component

```
{'NAME':'Paul','AGE':444.0,'DOB':'July, 27 2011 00:00:00'}
```

Note

Deserialization is unsupported for components.

Format for array

```
[{'NAME':'joe','AGE':30,'ADDRESS':'101 Some Drive, Pleasant Town,
90010','EYECOLOR':'blue','INCOME':50000},{'NAME':'paul','AGE':25,'ADDRESS':'Some
other address','EYECOLOR':'black','INCOME':40000}]
```

string, boolean, numeric, and date

Specify the values directly in the body of the request. ----

Support for GZip encoding

© 2014 Adobe Systems Incorporated. All rights reserved.

If the request contains a Content-Encoding header of "gzip" then the request entity (if any) is uncompressed using the gzip algorithm. If the request contains an Accept-Encoding header containing "gzip" and an "If-None-Match" Header, entitytag value is checked: if it contains the -gzip suffix, remove this suffix, otherwise, completely remove the "if-none-match" header.

If the request contains a Accept-Encoding header that contains "gzip", then the response entity (if any) is compressed using gzip and a Content-Encoding header of "gzip" is added to the response. As this filter is active, the resource representation can be compressed. the value "Accept-Encoding" is so added to the Vary header. If any entityTag is used and content may be gzipped, the "-gzip" suffix is added to entitytag value.

Site-level REST application support

In ColdFusion 10, multiple applications cannot have the same name, even if the applications are residing in different hosts. With the enhanced ColdFusion 11 REST feature, you can have multiple applications with the same name but available in different hosts. Also, you can set one default application (containing the REST service) for each virtual host.

You can register the directory containing the REST service by using any one of the following options:

- autoregister Application Setting
- ColdFusion Administrator console
- ColdFusion Admin API
- restInitApplication method

Option 1: autoregister Application Setting

A new Application setting *autoregister* is introduced in ColdFusion 11.

- You can set the auto register to true if you want to enable the auto registration for the application:

```
<cfset this.restsettings.autoregister="true"/>
```

- Specifying the servicemapping is optional. If servicemapping is not specified, the "this.name"/application name will be taken as default.

```
<cfset this.restsettings.servicemapping="testmapping"/>
```

- Specify the usehost or hostname, If usehost attribute is set to true, then the host name is parsed from the URL and it is used:

```
<cfset this.restsettings.usehost=true/>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

- Explicitly naming the host name will make the host name. If the host name is not mentioned, then the usehost name will be defaulted.

```
<cfset this.restsettings.host="www.adobe.com"/>
```

- Set isDefault to true and the application will be made as default app.

```
<cfset this.restsettings.isDefault=true/>
```

When the first request comes to the application and if that request is non-REST request, the application will be started and registered. If both *usehost* and *host* are not specified, the apps will be registered without host name.

	If the first request itself is a REST request, then the application will not get started.

Option 2: Registering a REST application using ColdFusion Administrator console

- Use the Administrator console to register a directory containing REST-enabled CF components. Select Adobe **ColdFusion 11 Administrator console** > **Data & Services** > **REST services**.
- Browse and select the root path (or the application path) where ColdFusion will search for the directory containing a set of REST enabled CF components.
  (Optional) In the Service Mapping section, specify the virtual mapping in place of application name. If the folder has an *Application.cfc file* and an application name, use the application name to identify the REST services in the directory.
  (Optional) Enter the Host Name. This will enable the mapping of the selected application with the host. Each host can have a default application and multiple applications with the same application name can be mapped to different hosts.
- Set the REST application as a default application by selecting the option Set as default application. By selecting this option, you omit the need to specify the service mapping or the application name in the URI.
- Click the **Add Service** button to complete registration of the directory.
- View the Active ColdFusion REST Services display table to view the list of active service mappings. You can use the same to edit or delete the service mappings in future.

Option 3: Registering a REST application using the ColdFusion Admin API

You can use functions defined in CFIDE.adminapi.extensions CFC to manage a REST application. The functions are:

- ***registerRESTService(path[,serviceMapping[,host[,isdef]]]***: This function registers the REST application. The root path specifies the directory containing REST-enabled CF component. Optionally, the service

© 2014 Adobe Systems Incorporated. All rights reserved.

mapping for the REST application, host name, and isdefault can be specified.

- ***getRESTServices():*** This function returns an array of REST services registered with the ColdFusion Administrator.
- ***deleteRESTService(rootPath):*** This function deletes the specified REST application registered with the ColdFusion Administrator.
- ***refreshRESTService(rootPath):*** If you make any changes to the REST-enabled CF component, you can refresh the registered application by calling this function.
- ***getDefaultRestService():*** Returns the server wide default REST application.
- ***getAllDefaultRESTServices:*** Returns all the default REST services. It is an array of path – host pair.

Option 4: Registering a REST application using the restInitApplication method

You can also register a REST application by calling the method restInitApplication

The syntax is:

```
restInitApplication(rootPath[,serviceMapping[,options]])
```

The options are an optional argument. In the options struct you can pass the:

- Host
- useHost
- isDefault

For registering by specifying the host explicitly:

```
<cfset str=structNew()>
<cfset str.host = "www.site1.com:82">
<cfset str.isDefault = "true">
<cfset
RestInitApplication("C:\dev\ColdFusion\cf_main\cfusion\wwwroot\withhostAndDefault",
"withhostAndDefault", str)>
App registered
```

For registering by specifying the UseHost attribute: The host name will be extracted from the request URL and will be used for registration.

```
<cfset str=structNew()>
<cfset str.useHost = "true">
<cfset str.isDefault = "true">
<cfset
RestInitApplication("C:\dev\ColdFusion\cf_main\cfusion\wwwroot\withhostAndDefault",
"withhostAndDefault", str)>
App registered
```

© 2014 Adobe Systems Incorporated. All rights reserved.

	You do not need Administrator privileges to perform the task. The syntax is: `restInitApplication(rootPath[,serviceMapping[,options]])`

- In the options you can specify *host, useHost* and *isDefault.* The option usage is same as *autoRegister* feature .
- If you have already registered the application using the Administrator module, call restInitApplication to refresh the REST service.
- Use the restDeleteApplication function to delete the REST service. The syntax is restDeleteApplication(rootPath)

Support for pluggable serializer and deserializer

In the application.cfc, you can register your own handler for serializing and deserializing the complex types. If the serializer is not specified, ColdFusion uses the default mechanism for serialization.

Example: If you have a phone directory to be serialized on the following parameters:

- The main data structure is an Array.
- Each element of the array is a struct.
- The Struct contains 2 elements.
- The first is the name of the contact and the second is the struct which contains the code and the phone number.

Array

```
Struct
        [
            name(String],
            phoneNo(Struct)
                [
                    code(String),
                    no(String)
                ]
        ]
    Struct
        [
            name(String],
            phoneNo(Struct)
                [
                    code(String),
                    no(String)
                ]
        ]
```

And in this example, you want to serialize only struct in a simple format and want the result as follows:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<Array size="1"> <Item index="1"> <root> <Name>Paul</Name> <PhoneNo> <root>
<Code>080</Code> <No>411150326</No> </root> </PhoneNo> </root> </Item> </Array>
```

With the enhanced ColdFusion 11 REST feature, the user can use the plugged-in custom serializer instead of using the default serialization function. The custom serializer has four functions:

- ***CanSerialize*** - Returns a boolean value and takes the "Accept Type" of the request as the argument. You can return true if you want the customserialzer to serialize the data to the passed argument type.
- ***Serialize*** - The main serialization logic must be implemented in this function. If canSerialize returns "True" for a request, then ColdFusion will use this function to serialize. If canSerialize returns false, then ColdFusion will use the default serialization to serialize.
- ***CanDeserialize*** - Returns a boolean value and takes the "Content Type" of the request as the argument. You can return true if you want the customserialzer to deserialize the data.
- ***DeSerialize*** - The main deserialization logic must be implemented in this function. If canDeSerialize returns "True" for a request, then ColdFusion will use this function to deserialize. If canDeSerialize returns false, then ColdFusion will use the default de-serialization to deserialize.

The below CustomSerializer code sample will help you to achieve the result for the scenario explained above (Customizing serialization/deserialization of phone directory):

```
<cfcomponent>
    <cffunction name="serialize" access="remote" returntype="String">
        <cfargument name="arg" type="any" hint="The object to be serialized"/>
        <cfargument name="type" type="string"
                    hint="The accept header of the request in array format. The
order of types in the array is according to the priority of the MIME types in the
header."/>

      <cfset var result = "">
      <cfset var key = "">

        <cfif arguments.type eq "XML">
            <cfif isStruct(arguments.arg)>
                <cfset result = "<root>">
                <cfloop collection="#arguments.arg#" item="key">
                    <cfset result = result & "<" & key & ">">
                    <cfset result = result & serializeXML(arguments.arg[key], true)>
                    <cfset result = result & "</" & key & ">">
                </cfloop>
                <cfset result = result & "</root>">
                <cfreturn result>
            <cfelse>
                <!--- SerializeXML is a new function added in ColdFusion 11. This
function will serialize the object to XML
    using ColdFusion's default serialization mechanism." --->
                <cfreturn serializeXML(arguments.arg)>
            </cfif>
        <cfelseif arguments.type eq "JSON">
            <cfdump var="#arguments.arg.getClass().getName()#" output="console">
            <cfif arguments.arg.getClass().getName() eq "java.lang.String">
                <cfreturn "test" & arguments.arg>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

```
            <cfelse>
                <cfreturn serializeJSON(arguments.arg)>
            </cfif>
        <cfelse>
            <!--- Serialize is a new function added in ColdFusion 11. This function
will serialize the object to a
   a specified type using ColdFusion's default serialization mechanism. --->
            <cfreturn serialize(arguments.arg, arguments.type)>
        </cfif>

    </cffunction>

    <cffunction name="canSerialize" access="remote" returntype="boolean">
        <cfargument name="type" type="string"/>

        <cfif arguments.type eq "XML">
            <cfreturn true>
        <cfelseif arguments.type eq "JSON">
            <cfreturn true>
        <cfelse>
            <cfreturn false>
        </cfif>
    </cffunction>

    <cffunction name="canDeserialize" access="remote" returntype="boolean">
        <cfargument name="type" type="string"/>

        <cfif arguments.type eq "XML">
            <cfreturn true>
        <cfelseif arguments.type eq "JSON">
            <cfreturn true>
        <cfelse>
            <cfreturn false>
        </cfif>
    </cffunction>

    <cffunction name="deserialize" access="remote" returntype="any">
        <cfargument name="arg" type="String" hint="The string to be deserialized"/>
        <cfargument name="type" type="String" hint="The content-type header of the
request."/>

      <cfset var xmlDoc = "">
      <cfset var result = "">
      <cfset var numEntries = "">
      <cfset var key = "">
      <cfset var value = "">

        <cfif arguments.type equals "XML" and isXml(arguments.arg)>
            <cfset xmlDoc = xmlParse(arguments.arg)>

            <cfif xmlDoc.XmlRoot.XmlName equals "root">
                <cfset result = StructNew()>
                <cfset numEntries = ArrayLen(xmlDoc.root.XMLChildren)>
                <cfloop index="i" from="1" to="#numEntries#">
                    <cfset key = xmlDoc.root.XMLChildren[i].XmlName>
                    <cfif ! len(Trim(xmlDoc.root.XMLChildren[i].XMLText))>
                        <cfset value =
deserializeXML(ToString(xmlDoc.root.XMLChildren[i].XMLChildren[1]), true)>
                    <cfelse>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

```
                        <cfset value =
deserializeXML(xmlDoc.root.XMLChildren[i].XMLText, true)>
                    </cfif>
                    <cfset result[key] = value>
                </cfloop>
                <cfreturn result>
            <cfelse>
                <cfreturn deserializeXML(arguments.arg, true)>
            </cfif>
        <cfelse>
            <cfreturn deserializeXML(arguments.arg, true)>
        </cfif>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

```
    </cffunction>

</cfcomponent>
```

The CustomSerializer that you have implemented can be specified through application.cfc.

```
<cfset this.customSerializer="CustomSerializer">
```

© 2014 Adobe Systems Incorporated. All rights reserved.

Media Player enhancements

The enhancements in this release support

- Play back capability for HTML 5 videos
- Fallback to HTML 5 video playback if Flash player is not installed
- Browser independent video controls
- Dynamic streaming of Flash videos
- Advanced skinning for media player
- Play list for Flash videos
- Embedding subtitles in SRT format using HTML `track` element
- Extending media player using plug-ins built using Open Source Media Framework (OSMF), for example to:
- Play videos in the YouTube server
- Use stage video support by showing advertisements within the videos in linear and non-linear mode
- Adding title to the video

> **Note**
>
> The mediaplayer's behavior is subject to the video capability of your device.

© 2014 Adobe Systems Incorporated. All rights reserved.

Displaying geolocation

Displays user location on the map if the attribute `showUser` is specified in `cfmap`. This feature works only on HTML 5 compliant browsers.

The following sections describe the changes to the tags `cfmap` and `cfmapitem` to display the user location.

© 2014 Adobe Systems Incorporated. All rights reserved.

Client-side charting

ColdFusion 10 supports client-side charting. This is in addition to the existing server-side charting feature (which continues to serve the way it used to).

Client-side charting supports the following:

- **Dynamic and interactive charting:** Modify the charts, add styles, and add new series or plots.
- **Popular chart formats with appropriate fallback functionality:** Use HTML 5, Flash, SVG, or VML charts. If your browser does not support HTML 5 features relevant to charting, charts are rendered in Flash. Similarly, if Flash is not supported, charts are rendered in HTML.
- **Features identical to server-side charting:** Most of the server-side charting features are available with client-side charting.
- * Old and new charts:* In addition to the contemporary chart types, offers a new set of charts.
- **Needs minimal trips to server:** As compared to generating charts at server-level, for every user interaction.

© 2014 Adobe Systems Incorporated. All rights reserved.

Caching enhancements

- Application-specific caching
- Enhanced query caching using Ehcache
  For details, see *Caching enhancements in ColdFusion 10* in Optimizing ColdFusion applications.

#back to top

© 2014 Adobe Systems Incorporated. All rights reserved.

Server update using ColdFusion Administrator

Verify if there are any product updates using the ColdFusion Administrator (Server Update > Update). The updates can include hot fixes and security hot fixes for ColdFusion 10.
For details, see *Configuring and Administering Adobe ColdFusion*.

#back to top

© 2014 Adobe Systems Incorporated. All rights reserved.

Secure Profile for ColdFusion Administrator

ColdFusion allows you to secure ColdFusion server furthermore by enabling or disabling selected settings on the ColdFusion Administrator. When installing ColdFusion, you can enable Secure Profile by selecting the option when prompted on the Secure Profile screen. Further, you could provide a comma separate list of IP addresses that may be allowed to access the ColdFusion Administrator. For more information, see Enabling Secure Profile for ColdFusion Administrator.

© 2014 Adobe Systems Incorporated. All rights reserved.

Introduction to application development using ColdFusion

The guide is intended for web application programmers who are learning ColdFusion or want to extend their ColdFusion programming knowledge. It provides a solid grounding in the tools that ColdFusion provides to develop many different types of web applications of varying complexity.

- Using the Developing ColdFusion Applications guide
- About Adobe ColdFusion documentation for Developers

#back to top

© 2014 Adobe Systems Incorporated. All rights reserved.

Using the Developing ColdFusion Applications guide

The *Developing ColdFusion Applications* guide includes basic and advanced information on CFML. However, it is most useful if you have basic ColdFusion experience or have viewed the Getting Started experience, which is available from the Adobe ColdFusion Administrator. Use the guide in conjunction with the *CFML Reference*, which contains detailed information on the CFML language elements.

#back to top

© 2014 Adobe Systems Incorporated. All rights reserved.

About Adobe ColdFusion documentation for Developers

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

© 2014 Adobe Systems Incorporated. All rights reserved.

Introducing ColdFusion

You use Adobe ColdFusion to create dynamic Internet applications.

- About Internet applications and web application servers
- About ColdFusion
- About JEE and the ColdFusion architecture

#back to top

© 2014 Adobe Systems Incorporated. All rights reserved.

About Internet applications and web application servers

With ColdFusion, you develop Internet applications that run on web application servers.

© 2014 Adobe Systems Incorporated. All rights reserved.

About web application servers

Typically, web browsers make requests, and web servers, such as Microsoft Internet Information Server (IIS) and the Apache web server, fulfill those requests by returning the requested information to the browser. This information includes, but is not limited to, HTML and FLA files.

Web server capabilities are limited because all it does is wait for requests to arrive and attempt to fulfill those requests as soon as possible. A web server does not let you do the following tasks:

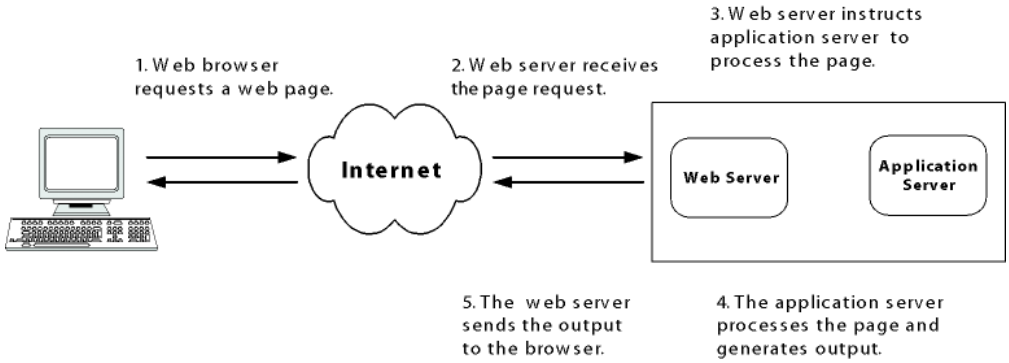
- Interact with a database, other resource, or other application.
- Serve customized information based on user preferences or requests.
- Validate user input.

A web server, basically, locates information and returns it to a web browser.
To extend the capabilities of a web server, you use a *web application server,* a program that extends web server capabilities to do tasks such as those in the preceding list.

How a web server and web application server work together

The following steps explain how a web server and web application server work together to process a page request:

1. The user requests a page by typing a URL in a browser, and the web server receives the request.
2. The web server looks at the filename extension to determine whether a web application server must process the page. Then, one of the following actions occur:
   - If the user requests a file that is a simple web page (often one with an HTM or HTML extension), the web server fulfills the request and sends the file to the browser.
   - If the user requests a file that is a page that a web application server must process (one with a CFM, CFML, or CFC extension for ColdFusion requests), the web server passes the request to the web application server. The web application server processes the page and sends the results to the web server, which returns those results to the browser. The following image shows this process:



Because web application servers interpret programming instructions and generate output that a web browser can interpret, they let web developers build highly interactive and data-rich websites, which can do tasks such as the following:

- Query other database applications for data.
- Dynamically populate form elements.
- Dynamically generate Flash data.
- Provide application security.
- Integrate with other systems using standard protocols such as HTTP, FTP, LDAP, POP, and SMTP.
- Create shopping carts and e-commerce websites.
- Respond with an e-mail message immediately after a user submits a form.
- Return the results of keyword searches.

#back to top

© 2014 Adobe Systems Incorporated. All rights reserved.

© 2014 Adobe Systems Incorporated. All rights reserved.

About web pages and Internet applications

The Internet has evolved from a collection of static HTML pages to an application deployment platform. First, the Internet changed from consisting of static web pages to providing dynamic, interactive content. Rather than providing unchanging content where organizations merely advertise goods and services, dynamic pages enable companies to conduct business ranging from e-commerce to managing internal business processes. For example, a static HTML page lets a bookstore publish its location, list services such as the ability to place special orders, and advertise upcoming events like book signings. A dynamic website for the same bookstore lets customers order books online, write reviews of books they read, and even get suggestions for purchasing books based on their reading preferences.

More recently, the Internet has become the underlying infrastructure for a wide variety of applications. With the arrival of technologies such as XML, web services, J2EE (Java 2 Platform, Enterprise Edition), and Microsoft .NET, the Internet has become a multifaceted tool for integrating business activities. Now, enterprises can use the Internet to integrate distributed activities, such as customer service, order entry, order fulfillment, and billing.

Adobe ColdFusion is a rapid application development environment that lets you build dynamic websites and Internet applications quickly and easily. It lets you develop sophisticated websites and Internet applications without knowing the details of many complex technologies, yet it lets advanced developers take advantage of the full capabilities of many of the latest Internet technologies.

© 2014 Adobe Systems Incorporated. All rights reserved.

About ColdFusion

Adobe ColdFusion is a rapid scripting environment server for creating dynamic Internet Applications. ColdFusion Markup Language (CFML) is a tag-based scripting language that is easy to learn. CFML provides connectivity to enterprise data and powerful built-in search and charting capabilities. ColdFusion enables developers to easily build and deploy dynamic websites, content publishing systems, self-service applications, commerce sites, and more.

ColdFusion pages are plain text files that you use to create web applications. You can create your ColdFusion applications by writing all the code manually or by using wizards (provided with some editors) to generate the majority of the code for you.

© 2014 Adobe Systems Incorporated. All rights reserved.

ColdFusion Administrator

ColdFusion Administrator configures and manages the ColdFusion application server. It is a secure web-based application that you can access using any web browser, from any computer with an Internet connection. It includes a Server Monitor, which lets you see the status of your ColdFusion server.

For more information about ColdFusion Administrator, see *Configuring and Administering ColdFusion*.

#back to top

© 2014 Adobe Systems Incorporated. All rights reserved.

Elements of ColdFusion

ColdFusion consists of the following core elements:

- ColdFusion scripting environment
- CFML
- ColdFusion Administrator

The ColdFusion scripting environment

The ColdFusion scripting environment provides an efficient development model for Internet applications. At the heart of the ColdFusion scripting environment is the ColdFusion Markup Language (CFML), a tag-based programming language that encapsulates many of the low-level details of web programming in high-level tags and functions.

ColdFusion Markup Language

ColdFusion Markup Language (CFML) is a tag-based language, like HTML, that uses special tags and functions. With CFML, you can enhance standard HTML files with database commands, conditional operators, high-level formatting functions, and other elements to rapidly produce web applications that are easy to maintain. However, CFML is not limited to enhancing HTML. For example, you can create Flash output that consist entirely of Flash elements and CFML. Similarly, you can use CFML to create web services for use by other applications.For more information, see Elements of CFML.

CFML tags

CFML looks like HTML-it includes starting and, in most cases, ending tags, and each tag is enclosed in angle brackets. All ending tags are preceded with a forward slash and all tag names are preceded with `cf`; for example:

```
<cftagname>
 tag body text and CFML
</cftagname>
```

CFML increases productivity by providing a layer of abstraction that hides many low-level details involved with Internet application programming. At the same time, CFML is powerful and flexible. ColdFusion lets you easily build applications that integrate files, databases, legacy systems, mail servers, FTP servers, objects, and components. CFML tags serve many functions. They provide programming constructs, such as conditional processing and loop structures. They also provide services, such as charting and graphing, full-text search, access to protocols such as FTP, SMTP/POP, and HTTP, and much more. The following table lists a few examples of commonly used ColdFusion tags:

Tag	Purpose
`cfquery`	Establishes a connection to a database (if one does not exist), executes a query, and returns results to the ColdFusion environment.
`cfoutput`	Displays output that can contain the results of processing ColdFusion functions, variables, and expressions.
`cfset`	Sets the value of a ColdFusion variable.

© 2014 Adobe Systems Incorporated. All rights reserved.

`cfmail`	Lets an application send SMTP mail messages using application variables, query results, or server files. (Another tag, `cfpop`, gets mail.)
`cfchart`	Converts application data or query results into graphs, such as bar charts or pie charts, in Flash, JPG, or PNG format.
`cfobject`	Invokes objects written in other programming languages, including COM (Component Object Model) components, Java objects such as Enterprise JavaBeans, or Common CORBA (Object Request Broker Architecture) objects.

CFML Reference describes the CFML tags in detail.

CFML functions and CFScript

CFML includes built-in functions that perform a variety of roles, including string manipulation, data management, and system functions. CFML also includes a built-in scripting language, CFScript, that lets you write code in a manner that is familiar to programmers and JavaScript writers.

CFML extensions

You can extend CFML further by creating custom tags or user-defined functions (UDFs), or by integrating COM, C++, and Java components (such as JSP tag libraries). You can also create ColdFusion components (CFCs), which encapsulate related functions and properties and provide a consistent interface for accessing them.All these features let you easily create reusable functionality that is customized to the types of applications or websites that you are building.

CFML development tools

Adobe Dreamweaver CS3 helps you develop ColdFusion applications efficiently. It includes many features that simplify and enhance ColdFusion development, including tools for debugging CFML. Because CFML is written in an HTML-like text format, and you often use HTML in ColdFusion pages, you can also use an HTML editor or a text editor, such as Notepad, to write ColdFusion applications.ColdFusion 9 includes a line debugger that you can use to debug your ColdFusion applications in Eclipse or Adobe Flex Builder.

© 2014 Adobe Systems Incorporated. All rights reserved.

Saving ColdFusion pages

In order for the ColdFusion server to process a page, save the ColdFusion page on a computer where ColdFusion is installed. If you are creating your pages on a local server (on which ColdFusion is running), you can save the pages locally; if you are running ColdFusion on a remote server, you must save your pages on that server.

If you are using ColdFusion's built-in web server, you would save your pages to *cfroot*/cfusion/wwwroot (or the wwwroot folder of another instance, if using ColdFusion Enterprise's ability to run multiple instances). .

If you are using an external web server, such as IIS or Apache, you would save your files to the location defined as the document root for your web site. For example, in IIS, you might save your files to c:\inetpub\wwwroot.

© 2014 Adobe Systems Incorporated. All rights reserved.

Testing ColdFusion pages

To ensure that the code you wrote is working as expected, you view the ColdFusion page in a browser by going to the appropriate URL, for example `http://localhost/test/mypage.cfm`. If you are using the built-in web server, specify the port to use in the URL, for example, `http://localhost:8500/test/cfpage.cfm`. The address localhost is only valid when you view pages locally.

> **Note**
>
> On Vista, the address `::1` is equivalent to localhost. You can use the ColdFusion `GetLocalHostIP` function to get the IP address of localhost.

The URL for a remote site includes the server name or IP address of the server where ColdFusion is installed; for example, `http://<serveripaddress>/test/mypage.cfm`. Some ColdFusion J2EE configurations require a context root in the URL; for example, `http://<server>/<context-root>/mypage.cfm`. For example, if you deploy an EAR file and use the default context root of cfconroot, you specify `http://localhost/cfconroot/test/mypage.cfm`.

© 2014 Adobe Systems Incorporated. All rights reserved.

About JEE and the ColdFusion architecture

As the Internet software market has matured, the infrastructure services required by distributed Internet applications, including ColdFusion applications, have become increasingly standardized. The most widely adopted standard today is the Java Enterprise Edition (JEE, formerly "J2EE") specification. JEE provides a common set of infrastructure services for accessing databases, protocols, and operating system functionality, across multiple operating systems.

© 2014 Adobe Systems Incorporated. All rights reserved.

About ColdFusion and the JEE platform

ColdFusion is implemented on the Java technology platform and uses a Java Enterprise Edition (JEE, formerly termed J2EE) application server for many of its base services, including database connectivity, naming and directory services, and other runtime services. ColdFusion can be configured to use an embedded JEE server (in the *server configuration* of ColdFusion Standard or Enterprise) or it can be deployed as a JEE web application on an independent JEE application server such as IBM WebSphere, Oracle Application Server, Oracle WebLogic, and JBoss, using the *JEE configuration* option in ColdFusion Enterprise..

For more information on ColdFusion configurations, see *Installing ColdFusion*.

By implementing the ColdFusion scripting environment on top of the JEE platform, ColdFusion takes advantage of the power of the JEE platform while also providing an easy-to-use scripting environment and built-in services. Moreover, because ColdFusion is built on a JEE platform, you can easily integrate JEE and Java functionality into your ColdFusion application. As a result, ColdFusion pages can do any of the following:

- Share session data with JSPs (Java Server Pages) and Java servlets.
- Import custom JSP tag libraries and use them like ColdFusion custom tags.
- Integrate with Java objects, including the JEE Java API, JavaBeans, and Enterprise JavaBeans.

For more information on using JEE features in ColdFusion, see Integrating JEE and Java Elements in CFML Applications.

© 2014 Adobe Systems Incorporated. All rights reserved.

The CFML Programming Language

In this section, you can understand using ColdFusion variables, expressions, number signs, arrays, structures and on extending the ColdFusion pages with CFML scripting.

- Elements of CFML
- Using ColdFusion Variables
- Using Expressions and Number Signs
- Using Arrays and Structures
- Extending ColdFusion Pages with CFML Scripting
- Using Regular Expressions in Functions
- ColdFusion Language Enhancements
- Built-in functions as first class citizen

#back to top

© 2014 Adobe Systems Incorporated. All rights reserved.

Elements of CFML

The basic elements of CFML, including tags, functions, constants, variables, expressions, and CFScript, make it a powerful tool for developing interactive web applications.

- CFML Basics
- Comments
- Tags
- Functions
- ColdFusion components
- Constants
- Variables
- Expressions
- Data types
- Flow control
- Character case
- Special characters
- Reserved words in ColdFusion
- cfscript tag
- Elvis operator

© 2014 Adobe Systems Incorporated. All rights reserved.

CFML Basics

CFML is a dynamic application development tool with many of the features of a programming language. These features include functions, expressions, variables and constants, and flow-control constructs, such as if-then and loops. CFML also has a "language within a language," CFScript, which enables you to use a syntax like JavaScript for many operations.

These elements and other basic CFML entities such as comments, data types, escape characters, and reserved words, let you create complex applications.

© 2014 Adobe Systems Incorporated. All rights reserved.

Comments

Comments

ColdFusion comments have a similar format to HTML comments. However, they use three dash characters instead of two; for example:

```
<!--- This is a ColdFusion Comment. Browsers do not receive it. --->
```

The ColdFusion server removes all ColdFusion comments from the page before returning it to the web server. As a result, the page that a browser receives does not include the comment. Users cannot see the comment even if they view the page source.
You can embed CFML comments in begin tags (not just tag bodies), functions calls, and variable text in number signs. ColdFusion ignores the text in comments such as the following:

```
<cfset MyVar = var1 <!--- & var2 --->>
<cfoutput>#Dateformat(now() <!---, "dddd, mmmm yyyy" --->)#</cfoutput>
```

This technique can be useful if you want to temporarily comment out parts of expressions or optional attributes or arguments.
You can also nest comments, as the following example shows:

```
<!--- disable this code
<!--- display error message --->
<cfset errormessage1="Oops!">
<cfoutput>
#errormessage1#
</cfoutput>
--->
```

This nesting is useful if you want to temporarily disable a section of code while you test your application.
You can embed comments within comments, however, use this technique carefully.

Note

You cannot embed comments inside a tag name or function name, such as `<cf_My<!--- New --->CustomTag>`. You also cannot embed comments inside strings, as in the following example: `IsDefined("My<!--- New --->Variable")`.

© 2014 Adobe Systems Incorporated. All rights reserved.

Tags

ColdFusion *tags* tell the ColdFusion server that it must process information. The ColdFusion server only processes tag contents; it returns text outside ColdFusion to the web server unchanged. ColdFusion provides a wide variety of built-in tags and lets you create custom tags.

Tag syntax

ColdFusion tags have the same format as HTML tags. They are enclosed in angle brackets (< and >) and can have zero or more named attributes. Many ColdFusion tags have bodies; that is, they have beginning and end tags with text for processing between them. For example:

```
<cfoutput>
 Hello #YourName#! <br>
</cfoutput>
```

Other tags, such as `cfset` and `cfhttp`, never have bodies. All the required information goes between the beginning (<) character and the ending (>) character, as in the following example:

```
<cfset YourName="Bob">
```

Note

The `cfset` tag differs from other tags in that it does not have a body or arguments. Instead, the tag encloses an assignment statement that assigns a value to a variable. The `cfset` tag can also call a function without assigning a value to a result variable.

Sometimes, although the tag can have a body, it is unnecessary because the attributes specify all the required information. You can omit the end tag and place a forward slash character before the closing (>) character, as in the following example:

```
<cfprocessingdirective pageencoding="euc-jp" />
```

In most cases, you specify tag attributes directly in the tag using the format *attributeName*=" }}*attributeValue* {{" }}, as the preceding example shows. However, as an alternative, you can place all the attributes in a structure and specify the structure in a single {{attributeCollection attribute, using the following format:

```
<tagname attributeCollection="#structureName#">
```

When you use this format for all built-in ColdFusion tags except `cfmodule`, the tag must have only the `attributeCollection` attribute. This format is useful when you use dynamic arguments, where the number and values of the arguments to a tag can vary based on processing results. The following example shows this usage:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<!--- Configure dynamic attribute variables. --->
 <cfparam name="theURL" default="http://www.adobe.com">
 <cfparam name="resolveURL" default="yes">

<!--- Code that dynamically changes values for attributes can go here. --->

<!--- Create an arguments structure using variables. --->
 <cfset myArgs=StructNew()>
 <cfset myArgs.url="#theURL#">
<!--- Include a user name and password only if they are available. --->
 <cfif IsDefined("username")>
 <cfset myArgs.username="#username#">
</cfif>
<cfif IsDefined("password")>
 <cfset myArgs.password="#password#">
</cfif>
<cfset myArgs.resolveURL="#resolveURL#">
<cfset myArgs.timeout="2">

<!--- Use the myArgs structure to specify the cfhttp tag attributes. --->
 <cfhttp attributeCollection="#myArgs#">
 <cfoutput>
  #cfhttp.fileContent#
 </cfoutput>
```

Note

The `attributeCollection` attribute used in the `cfmodule` tag and when calling custom tags directly is different from the `attributeCollection` attribute for all other tags. In the `cfmodule` tag and in custom tags, you can mix the `attributeCollection` attribute and explicit custom tag attributes. Also, in the `cfmodule` tag, the `attributeCollection` attribute cannot contain the `name` and `template` attributes. Specify these attributes directly in the `cfmodule` tag.

You can use the `attributeCollection` attribute in all tags *except* the following:

cfargument	cfelseif	cflogout	cfset
cfbreak	cffunction	cfloop	cfsilent
cfcase	cfif	cfparam	cfswitch
cfcatch	cfimport	cfprocessingdirective	cftry
cfcomponent	cfinterface	cfproperty	
cfdefaultcase	cflogin	cfrethrow	
cfelse	cfloginuser	cfreturn	

Built-in tags

© 2014 Adobe Systems Incorporated. All rights reserved.

Built-in tags make up the heart of ColdFusion. These tags have many uses, including the following:

- Manipulating variables
- Creating interactive forms
- Accessing and manipulating databases
- Displaying data
- Controlling the flow of execution on the ColdFusion page
- Handling errors
- Processing ColdFusion pages
- Managing the CFML application framework
- Manipulating files and directories
- Using external tools and objects, including COM, Java, and CORBA objects, and executable programs
- Using protocols, such as mail, http, ftp, and pop
  The *CFML Reference* documents each tag in detail.

Custom tags

ColdFusion lets you create custom tags. You can create two types of custom tags:

- CFML custom tags that are ColdFusion pages
- CFX tags that you write in a programing language such as Java or C++
  Custom tags can encapsulate frequently used business logic or display code. These tags enable you to place frequently used code in one place and call it from many places. Custom tags also let you abstract complex logic into a single, simple interface. They provide an easy way to distribute your code to others. You can even distribute encrypted versions of the tags to prevent access to the tag logic.
  You can access a variety of free and commercial custom tags on the Adobe ColdFusion Exchange (www.adobe.com/go/learn_cfu_cfdevcenter_en). They perform tasks ranging from checking if Cookies and JavaScript are enabled on the client browser to moving items from one list box to another. Many of these tags are free and include source code.

CFML custom tags

When you write a custom tag in CFML, you can take advantage of all the features of the ColdFusion language, including all built-in tags and even other custom tags. CFML custom tags can include body sections and end tags. Because they are written in CFML, you do not need to know a programming language such as Java. CFML custom tags provide more capabilities than user-defined functions, but are less efficient.
For more information on CFML custom tags, see Creating and Using Custom CFML Tags. For information about, and comparisons among, ways to reuse ColdFusion code, including CFML custom tags, user-defined functions, and CFX tags, see Creating ColdFusion Elements.

CFX Tags

CFX tags are ColdFusion custom tags that you write in a programming language such as Java or C++. These tags can take full advantage of all the tools and resources provided by these languages, including their access to runtime environments. CFX tags also generally execute faster than CFML custom tags because they are compiled. CFX tags can be cross-platform, but are often platform-specific, for example if they take advantage of COM objects or the Windows API.
For more information on CFX tags, see Building Custom CFXAPI Tags.

Tags as functions and operators

ColdFusion provides many functions or operator language elements that correspond to CFML tags. Together with the existing CFScript language, these elements let you define many CFCs and functions entirely in CFScript.
The new functions and operators belong to the following tag categories:

© 2014 Adobe Systems Incorporated. All rights reserved.

- Tags without bodies, such as cfexit and cfinclude
- Language tags with bodies, such as cflock and cftransaction
- Service tags with bodies, such as cfmail and cfquery
- Tags for defining and using components and functions: cfcomponent, cfinterface, cfimport, cffunction, cfproperty, cfargument. For more information, see Defining components and functions in CFScript.

Tag without bodies

Several basic ColdFusion tags now have corresponding CFScript operators. These operators take a subset of standard tag attributes, and do not allow custom attributes. They do not return values.
The following list specifies the CFML tags and their corresponding CFScript syntax:

- cfabort: `abort ["message"];`
- cfexit: `exit ["methodName"];`
- cfinclude: `include "template";`
- cfparam: `param [type] name [=defaultValue];`

The param attribute can now take any number of name=value pairs. Param can also take all the attributes of `<cfparam>` as name-value pairs.For example:

```
<cfscript>
param name="paramname" default="value" min="minvalue" max="maxvalue"
pattern="pattern"
</cfscript>
```

- cfrethrow: `rethrow;`
- cfthrow: `throw "message";`

For detailed information on the statement parameters, see the corresponding tag attribute description in the *CFML Reference*.

Language-level tags with bodies

ColdFusion includes CFScript elements that provide the functions of the following language (compiler)-level tags, which have bodies. These tags manage the execution of code within their bodies:

- cflock: `lock`
- cfthread: `thread`
- cftransaction: `transaction`

Thread and transaction support also include functions, such as `threadJoin` and `transactionCommit`, that let you manage any thread or transaction, whether you define it with a tag or a function.

The lock, thread, and transaction operations have the following syntax:

```
operationName attributeName1=value1 attributName2=value2...
{body contents }
```

cflock

The lock operation has no special characteristics or limitations. All cflock tag attributes are valid operation

© 2014 Adobe Systems Incorporated. All rights reserved.

parameters. The following code uses the lock operation:

```
lock scope = "request" timeout = "30" type = "Exclusive" {
request.number = 1;
writeoutput("E-Turtleneck has now sold "& request.number &"
turtlenecks!");
}
```

cftransaction

To use the transaction operation you specify a begin action parameter. A transaction has the following general form:

```
TRANSACTION action="begin" [isolation="isolationValue"] {
transaction code
}
```

Within the transaction block you call the following methods to manage the transaction:

- `transactionCommit()`
- `transactionRollback([savepoint])`
- `transactionSetSavepoint([savepoint])`
  The `savepoint` parameter is a string identifier for the savepoint.

> **Note**
>
> You can also use theses methods in a `cftransaction` tag body.

You can nest transaction operations. For more information on nested transactions, see cftransaction in *CFML Reference*.
The following example uses nested transaction operations:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cfscript>
 qry = new Query();
 qry.setDatasource("test");
 qry.setSQL("delete from art where artid=62");
 qry.execute();
 TRANSACTION action="begin"
 {writeoutput("Transaction in cfscript test");
 TRANSACTION action="begin" {
 qry.setSQL("insert into art(artid, artistid, artname, description, issold, price)
 values ( 62, 1, 'art12', 'something', 1, 100)");
 qry.execute();}
 transactionSetSavepoint("sp01");
 qry.setSQL("update art set artname='art45' where artid=62");
 qry.execute();
 transactionSetSavepoint("sp02");
 qry.setSQL("update art set artname='art56' where artid=62");
 qry.execute();
 transactionrollback("sp02");
 transactioncommit();
 }
</cfscript>
```

cfthread

To use the thread operation you specify a run action parameter. The thread runs the code in the operation body. A thread block has the following general form:

```
THREAD name="text" [action="run"] [priority="priorityValue"
application-specific attributes] {
thread code
}
```

The code in the thread operation body executes in a single ColdFusion thread. Code outside the body is not part of the thread. You can use the following methods to manage the thread:

- `threadTerminate(threadName)`
  This function terminates the thread specified by the threadName parameter. It behaves in the same way as cfthread action="terminate".
- `threadJoin([[threadName], timeout])`
  This function joins the current thread with the specified thread or threads. The current thread waits until either the specified threads complete, or the timeout period passes, whichever happens first. The current thread inside a thread function block belongs to that block thread and the current thread outside a thread function block is the page thread.
  The threadName parameter is a comma-delimited list specifying one or more threads to join with the page thread. If you omit this attribute, the current thread waits until all ColdFusion threads finish running.
  The timeout parameter specifies the maximum time, in milliseconds, the calling thread waits for the other threads to complete processing. If one or more threads do not complete before the time out period, the current thread processing begins immediately. If you omit this attribute, the current thread waits until all specified threads finish running.

© 2014 Adobe Systems Incorporated. All rights reserved.

> **Note**
>
> You can also use these functions with transactions that you create by using cftransaction tags.

Service tags with bodies

ColdFusion provides objects, implemented as CFCs, that correspond to the following service tags:

- `cfftp`
- `cfhttp`
- `cfmail`
- `cfpdf`
- `cfquery`
- `cfstoredproc`
  These tags have bodies and provide services such as executing queries or sending mail. Many of them have action attributes, whereas others have an implicit action, such as execute. For each service tag, except for `cfmail` and `cfpdf`, a component is returned with applicable properties set and you need to invoke getters on the properties to access the data.

> **Note**
>
> Previously, invoking getName() and getResult() methods returned data like query resultset, pdf object, or ftp prefix, but now this has been changed and instead a component is returned with appropriate properties set.

The object names are the tag names without the cf prefix, for example, ftp. These objects also support child tag functionality, such as cfmailpart and cfmailparam.

> **Note**
>
> There may be thread-safety issues if implicit setters are used and child tags such as `cfmailpart` or `cfmailparam` are added because they get added into the CFC variable scope. It is therefore recommended that you create a new component for each service. If you need to preserve the attribute state, use duplicate() on the component to retain any initialized attribute values.

To use these tags in functions you:

1. Instantiate a service object.
2. Set object attributes and child tags
3. Execute one or more actions on the object.

> **Note**
>
> Unlike the corresponding tags, you cannot use application-specific parameters in these functions. You can only use the parameters that ColdFusion supports directly.

Step 1: Instantiate a service object

To create a function object, such as a mail object, use the `new` operator or `createobject()` function, as in the

© 2014 Adobe Systems Incorporated. All rights reserved.

following example:

```
myMail = new mail(server="sendmail.myCo.com");
```

Step 2a: Managing attributes

You can set attributes in several ways:

- As name=value parameters to the object initializer when you instantiate the object, as in the following example.

```
myMail = new mail(server="sendmail.myCo.com");
```

- As name=value parameters to an object action method, as in the following example:

```
Q = myQuery.execute(sql="select * from art");
```

- By using attribute setters, as in the following example:

```
myMail.setSubject("Hi");
```

Note

You cannot use a `getAttributeName` function to get the value of the attribute specified by AttributeName. Instead, use `GetAttributes(AttributeName)`.

- By using the following functions:

```
SetAttributes(attrib1=value,attrib2=value,...);
GetAttributes([attribName1[,attribName2]]....);
ClearAttributes([attribName1[,attribName2]]...);
```

Note

If you specify a result attribute for a stored procedure, then calling `getPrefix()` returns, `executionTime`, `statusCode`, `cached` . If you do not specify a result attribute, `getPrefix()` returns only `executionTime` and `statusCode`.

Step 2b: Managing child tag operations

All service objects correspond to tags that have child tags. For example, `cfmail` has `cfmailpart` and `cfmailparam` child tags.To specify the child tag functionality, use the following methods:

© 2014 Adobe Systems Incorporated. All rights reserved.

- `httpObj.addParam`
- `mailObj.addParam`
- `mailObj.addPart`
- `pdfObj.addParam`
- `queryObj.addParam`
- `storedProcObj.addParam`
- `storedProcObj.addProcResult`
  For example:

```
mailObj.addparam(file="#ExpandPath('test.txt')#");
mailObj.addPart(name="foo",type="html",charset="utf-8",
body="This is a test message.");
```

You can also clear child tag settings by calling the following functions.

- `httpObj.clearParams`
- `mailObj.clearParams`
- `mailObj.clearParts`
- `pdfObj.clearParams`
- `queryObj.clearParams`
- `storedProcObj.clearParams`
- `storedProcObj.clearProcResults`
  If you used multiple add methods on an object, the clear method clears all values set in all the methods.

Step 3: Executing service actions

Service tags, excluding cfmail and cfpdf, have one or more actions that return results. Some, including the cfpdf and cfftp tags have action attributes. For these tags, each action corresponds to a method on the service object. For example, the ftp object action methods include `open`, `close`, `listDir`, `getFile`, `rename`, and many others. However, the way service tags return data has changed. Now, a component is returned with applicable properties set and you need to invoke getters on the properties to access the data.

> **Note**
>
> The PDF object has two action methods whose names differ from the corresponding cfftp action attribute values: `getPDFInfo` and `setPDFInfo` instead of `getInfo` and `setInfo`. This difference is required to prevent name collisions with the set and get methods for the PDF info attribute.

The `cfhttp`, `cfmail`, `cfquery`, and `cfstoredproc` tags do not have action attributes. Instead, the tags perform a single action. To perform these actions in `cfscript`, call the following functions:

- `httpObj.send()`
- `mailObj.send()`
- `queryObj.execute()`
- `storedProcObj.execute()`
  To get an action result, you typically assign the results of the action method to a variable, as in the following example:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
Q = qry.execute(sql="select * from art");
```

Note

The attributes that specify you for an action are valid only for that action and are cleared once the action is completed.

Service code example: mail, ftp, and http

The following example shows the use of the mail, http, and ftp services in `cfscript`.

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<!---mail and ftp service --->
<cfscript>
 m = new mail();
<!---mail service --->
 m.setTo("x@adobe.com");
<!---set attribute using implicit setter --->
 m.setSubject("Hi");
 m.setBody("test mail");
<!---users need to use 'body' to specify  cfmail and cfmailpart content --->
 m.addparam(file="#E xpandPath(test.txt)#");
<!---add cfmail param tags --->
 m.addPart(type="html",charset="utf-8",body="some
 mailpart content");
<!---add cfmailpart tags --->
 m.send(to="y@abc.com" ....);
<!---attributes can be overriden when sending mail --->
 m.clear();
<!---clearAttributes(),clearParams() and clearParts() can also be used to clear --->

 individual items, if needed
<!---ftp service --->
 f = new ftp(server="s",username="u",password="p");
<!---check if a specified directory already exists (note the usage of getPrefix
())--->
 f.existsDir(directory ="some_dir").getPrefix().returnValue ? WriteOutput("Directory

 exists"):WriteOutput("Directory does not exist");
 <!---list directory contents (note the usage of getResult() and getPrefix() --->
 r = f.listDir(directory="some_dir",name="dirContents");
 dirContents = r.getResult();
 r.getPrefix().succeeded ? WriteOutput("List Directory operation successful") :
</cfscript>
<!---http service --->
<cfscript>
 httpObj = new http();
 <!---example 1 --->
  <!---add params--->
  httpObj.addParam(type="cgi", Name="Content-type", value =
  "application/x-www-form-urlencoded",encoded="no");
  httpObj.addParam(type="body",value="test1=value1&test2=
  value2&arraytest=value1&arraytest=value2");
 <!---assign the component returned to a variable--->
  r = httpObj.send(url="http://localhost:8500/
  project1/cfscript_test_files/thread-
  safe/http/_cfhttpparam_body.cfm",method="POST");
 <!---use getPrefix() to dump the   cfhttp prefix --->
  writedump(r.getPrefix());
 <!---example 2 --->

 <!---using attributes that return a query --->
  r = httpObj.send(url="
  http://localhost:8500/language_enhancements_2/cfscript_test_files/thread-
  safe/http/vamsee.txt")",name="myqry", firstrowasheaders="no",method="GET");
 <!---dump result and name attributes data --->
  writedump(r.getPrefix());
  writedump(r.getResult());
</cfscript>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

For `cfftp`, following are available getters on the returned component:

- `getPrefix()}}Returns the tag prefix {{cfftp`, which is a struct, available after any `cfftp` operation
- `getResult()` Applicable only to `action="listDir"`
  For `cfhttp`, following are the available getters on the returned component:
- `getPrefix()}}Returns the {{cfhttp` prefix (struct) available after the tag has executed
- `getResult()` Applicable only if attributes like `columns`, `delimiter`, `firstrowasheaders`, `name`, or `textQualifier` are specified, which direct ColdFusion to return a query object.

Query service example

```
<cfscript>
 qryObj = new createObject("component","com.adobe.coldfuison.query").init();
 <!---r here is no longer the query recordset but a component --->
 r = qryObj.execute(sql="select * from art",
 datasource="cfdocexamples",result="myresult",name="myquery");
 <!---new way to access the data --->
 resultset =r.getResult();
 prefixData = r.getPrefix();
 writedump(resultset);
 writedump(prefixData);
 <!---Using QoQ--->
 qryObj.setAttributes(myquery=resultset);
 r = qryObj.execute(sql="select * from myquery", dbtype="query");
 writedump(r.getResult());
 writedump(r.getPrefix());
</cfscript>
```

The following are the available getters on the returned component:

- {{getPrefix()}}Returns the result struct available after the query has executed.
- `getResult()}}Returns the resultset returned by query ({{SELECT` query) and throws an error for other types of SQL statements or queries (like `INSERT`, `UPDATE`, `DELETE`).

PDF example

Whenever any action is performed for which a name attribute is specified, the new pdf is returned back to the user. The following code shows typical actions on a PDF.

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cfscript>
 pdfObj = new pdf();
 x = pdfObj.read(source=#sourcefile#, name="PDFInfo");
 x = pdfObj.processddx(ddxfile="#tocddx#",inputfiles="#inputStruct#",outputfiles=
  "#outputStruct#",name="ddxVar");
 x = pdfObj.addWatermark(source="#pdf1#",image="#image1#", pages="1",
  overwrite="yes", name="test2");
 x = pdfObj.removewatermark(source="#pdf1#", name="temp");
 x = pdfObj.deletePages(source="#destfolder#dest.pdf",pages="2-4", name="deltest");
 pdfObj.addparam(source="#pdf1#", pages="1-2,4");
 pdfObj.merge(destination="#destfolder#merge-oneBigFile-5.pdf", overwrite="yes");
 pdfObj.thumbnail(source="#pdf1#", overwrite="yes");
 pdfObj.setInfo(source="#pdf1#", info="#{Author="Donald Duck"}#",
     destination="#destfolder#pdfinfo.pdf", overwrite="yes");
 pdfObj.write(source="myBook", destination="#destfolder#write1.pdf", version="1.4",
    overwrite="yes");
 pdfObj.protect(source="MyPdfVar", password="adobe", permissions="none",
     newuserpassword="newuserpw",  newownerpassword="newownerpw");
</cfscript>
```

Storedproc example

The following code shows sample usage of the storedproc service object.

```
<cfscript>
 sp = new storedproc();
 <!---add cfprocparam tags --->
  sp.addParam(TYPE = "IN", CFSQLTYPE="CF_SQL_VARCHAR", VALUE="David",
    DBVARNAME="@firstname");
  sp.addParam(TYPE="IN", CFSQLTYPE="CF_SQL_VARCHAR", VALUE="Peterson",
  DBVARNAME="@lastname", null ="yes");
  sp.add Param(TYPE="OUT", CFSQLTYPE="CF_SQL_INTEGER", variable="MyCount",
    DBVARN AME="@MyCount");
 <!---add cfprocresult  tags --->
  sp.addProcResult(NAME = "home r", RESULTSET = 1);
   sp.addProcResult( NAME = "home r2", RESULTSET = 2);
  sp.addProcResult(NAME = "home r3", RESULTSET = 3) ;
 <!---execute stored proc--->
  r = sp.execute(procedure="sp_weird",datasource="some_dsn",result="r");
  writedump(r.getProcResultSets());
 <!---changed from sp.getProcResults()--->
  writedump(r.getProcResultSets ("home r3"));
  writedump(r.getPrefix());
 <!---changed from sp.getResult()--->
  writedump(r.getProcOutVariables());
 <!---changed from sp.getProcVars()--->
</cfscript>
```

The following are the available getters on the returned component:

- `getPrefix()}}Returns the {{cfstoredproc` prefix (struct) available after the procedure has executed.

© 2014 Adobe Systems Incorporated. All rights reserved.

- {{getProcResultsets()}}Returns any resultsets returned by the strored procedure.
- `getProcOutVariables()`Returns any `OUT` or `INOUT` variables set by the procedure.

© 2014 Adobe Systems Incorporated. All rights reserved.

Functions

Functions

Functions typically manipulate data and return a result. You can also create user-defined functions (UDFs), sometimes referred to as custom functions.
Functions have the following general form:

```
functionName([argument1[, argument2]]...)
```

Some functions, such as the `Now` function take no arguments. Other functions require one or more comma-separated arguments and can have additional optional arguments. All ColdFusion functions return a value. For example, `Round(3.14159)` returns the value 3.

Built-in functions

ColdFusion built-in functions perform a variety of tasks, including, but not limited to, the following:

- Creating and manipulating complex data variables, such as arrays, lists, and structures
- Creating and manipulating queries
- Creating, analyzing, manipulating, and formatting strings and date and time values
- Evaluating the values of dynamic data
- Determining the type of a variable value
- Converting data between formats
- Performing mathematical operations
- Getting system information and resources
  For alphabetical and categorized lists of ColdFusion functions, see ColdFusion Functions in the *CFML Reference*.
  You use built-in functions throughout ColdFusion pages. Built-in functions are frequently used in a `cfset` or `cfoutput` tag to prepare data for display or further use. For example, the following line displays today's date in the format October 24, 2007:

```
<cfoutput>#DateFormat(Now(), "mmmm d, yyyy")#</cfoutput>
```

This code uses two *nested* functions. The `Now` function returns a ColdFusion date-time value representing the current date and time. The `DateFormat` function takes the value returned by the `Now` function and converts it to the desired string representation.
Functions are also valuable in CFScript scripts. ColdFusion does not support ColdFusion tags in CFScript, so you must use functions to access ColdFusion functionality in scripts.

Implicit Get and Set Functions

ColdFusion components support private properties with public setter and getter methods. This behavior supports object-oriented programming by letting you hide component properties from direct access.
Use the following code, for example, to set and get the MyProp property of `myCFC` component:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
myCFC.setMyProp(27);
theProp = myCFC.getMyProp();
```

Features of properties with setter and getter methods include the following:

- Component properties you assign with the set method are in the Variables scope that is private to the CFC. You can get or reset the properties only by calling get or set methods.
- If a property has a type attribute value, ColdFusion validates the data you pass to the setter function. The default attribute has no effect on the property and does not set an initial property value.
- A direct assignment statement, such as `myCFC.MyProp=27` creates a standard `This` scope variable in the CFC, even if you specify the property in a `cfproperty` tag. The `This` scope variable is independent of the properties that you access using the set and get methods. In fact, you can have a `This` scope variable with the same name as a property that you access using the set and get methods.
- Use the `cfproperty` tag getter and setter attributes to control access to a property from outside the CFC: A setter attribute value of true allows application code to set the property (the default behavior). A false value specifies that the property can only be set from within the CFC. The getter attribute works similarly.
- Explicit set or get methods override the implicit set and get methods. Therefore, if a CFC has a `MyProp` prop erty with an explicit `setMyProp` method, and you call the `setMyProp()` function in your application code, ColdFusion uses your function and not an implicit function.

Validate and validateparams attributes

The `validate` attribute available with `<cfproperty>` takes the validator to be used for validating data when implicit setter for this property is called. It takes the following validators:

- string
- boolean
- integer
- numeric
- date
- time
- creditcard: A 13-16 digit number conforming to the mod10 algorithm.
- email: A valid e-mail address.
- eurodate: A date-time value. Any date part must be in the format dd/mm/yy. The format can use /, -, or . characters as delimiters.
- regex: Matches input against pattern specified in `validateparams`.
- ssn: A U.S. social security number.
- telephone: A standard U.S. telephone number.
- UUID: A Home Universally Unique Identifier, formatted 'XXXXXXXX-XXXX-XXXX-XXXXXXXXXXXXXXX', where 'X' is a hexadecimal number.
- guid: A Universally Unique Identifier of the form "XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX" where 'X' is a hexadecimal number
- zipcode: U.S., 5- or 9-digit format ZIP codes
  The `validateparams` attribute available with `<cfproperty>` takes the parameters required by the validator specified in the validate attribute. This should be specified in the implicit struct notation.

- min: Minimum value if validate is integer/numeric/
- max: Maximum value if the validate is integer/numeric/
- minLength: Minimum length of the string if the validate is string
- maxLength: Maximum length of the string if the validate is string
- pattern: regex expression if the validator specified in validate attribute is regex

© 2014 Adobe Systems Incorporated. All rights reserved.

For example, the following code sets the validators for e-mail, zipcode, and age of an employee.

> **Note**
>
> For age, `validate` checks if the value is an integer and `validateparams` checks the range of the value supplied.

```
<!---Setting validators for an employee's e-mail, age, and zipcode--->
<cfcomponent>
<cfproperty name="mail" validate="email">
<cfproperty name="zip" validate="zipcode">
<cfproperty name="age" validate="integer" validateparams="{min=18,max=60}">
</cfcomponent>
```

User-defined functions

You can write your own functions, *user-defined functions* (UDFs). You can use these functions in ColdFusion expressions or in CFScript. You can call a user-defined function anywhere you can use a built-in CFML function. You create UDFs using the `cffunction` tag or the `cfscript function` statement. UDFs that you create using the `cffunction` tag can include ColdFusion tags and functions. UDFs that you create in CFScript can only include functions. You can create stand-alone UDFs or encapsulate them in a ColdFusion component.
User-defined functions let you encapsulate logic and operations that you use frequently in a single unit. This way, you can write the code once and use it multiple times. UDFs ensure consistency of coding and enable you to structure your CFML more efficiently.
Typical user-defined functions include mathematical routines, such as a function to calculate the logarithm of a number; string manipulation routines, such as a function to convert a numeric monetary value to a string such as "two dollars and three cents"; and can even include encryption and decryption routines.

> **Note**
>
> The Common Function Library Project at www.cflib.org includes a number of free libraries of user-defined functions.

For more information on user-defined functions, see Writing and Calling User-Defined Functions.

© 2014 Adobe Systems Incorporated. All rights reserved.

ColdFusion components

ColdFusion components encapsulate multiple, related, functions. A ColdFusion component is essentially a set of related user-defined functions and variables, with additional functionality to provide and control access to the component contents. ColdFusion components can make their data private, so that it is available to all functions (also called methods) in the component, but not to any application that uses the component.
ColdFusion components have the following features:

- They are designed to provide related services in a single unit.
- They can provide web services and make them available over the Internet.
- They can provide ColdFusion services that Flash clients can call directly.
- They have several features that are familiar to object-oriented programmers, including data hiding, inheritance, packages, and introspection.
  For more information on ColdFusion components, see Building and Using ColdFusion Components.

© 2014 Adobe Systems Incorporated. All rights reserved.

Constants

Constants

The value of a *constant* does not change during program execution. Constants are simple scalar values that you can use within expressions and functions, such as "Robert Trent Jones" and 123.45. Constants can be integers, real numbers, time and date values, Boolean values, or text strings. ColdFusion does not allow you to give names to constants.

© 2014 Adobe Systems Incorporated. All rights reserved.

Variables

Variables

Variables are the most frequently used operands in ColdFusion expressions. Variable values can be set and reset, and can be passed as attributes to CFML tags. Variables can be passed as parameters to functions, and can replace most constants.
ColdFusion has several built-in variables that provide information about the server and ColdFusion tags return. For a list of the ColdFusion built-in variables, see Reserved Words and Variables in the *CFML Reference*.
The following two characteristics classify a variable:

- The *scope* of the variable, which indicates where the information is available and how long the variable persists
- The *data type* of the variable value, which indicates the type of information a variable represents, such as number, string, or date
  See Data types for a list of data types (which also apply to constant values). For detailed information on ColdFusion variables, including data types, scopes, and their use, see Using ColdFusion Variables.

© 2014 Adobe Systems Incorporated. All rights reserved.

Expressions

Expressions

ColdFusion *expressions* consist of *operands* and *operators*. Operands are constants and variables, such as "Hello" or MyVariable. Operators, such as the string concatenation operator (&) or the division operator (\/) are the verbs that act on the operands. ColdFusion functions also act as operators.
The simplest expression consists of a single operand with no operators. Complex expressions consist of multiple operands and operators. For example, the following statements are all ColdFusion expressions:

```
12
MyVariable
(1 + 1)/2
"father" & "Mother"
Form.divisor/Form.dividend
Round(3.14159)
```

For detailed information on using variables, see Using ColdFusion Variables. For detailed information on expressions and operators, see Using Expressions and Number Signs.

© 2014 Adobe Systems Incorporated. All rights reserved.

Data types

ColdFusion is considered *typeless* because you do not explicitly specify variable *data types*.
However, ColdFusion data, the constants and the data that variables represent, *do* have data types, which correspond to the ways the data is stored on the computer.
ColdFusion data belongs to the following type categories:

Category	Description and types
Simple	Represents one value. You can use simple data types directly in ColdFusion expressions. ColdFusion simple data types are: • strings A sequence of alphanumeric characters enclosed in single or double quotation marks, such as "This is a test." • integers A sequence of numbers written without quotation marks, such as 356. • real numbers, such as -3.14159 • Boolean values Use True, Yes, or 1 for true and False, No, or 0 for false. Boolean values are not case sensitive. • date-time values ColdFusion supports a variety of data formats. For more information, see DateTimeFormat.
Complex	A container for data. Complex variables generally represent more than one value. ColdFusion built-in complex data types are: • arrays • structures • queries
Binary	Raw data, such as the contents of a GIF file or an executable program file
Object	COM, CORBA, Java, web services, and ColdFusion Component objects: Complex objects that you create and access using the `cfobject` tag and other specialized tags.

Note

ColdFusion does not have a data type for unlimited precision decimal numbers, but it can represent such numbers as strings and provides a function that supports unlimited precision decimal arithmetic. For more information, see PrecisionEvaluate in the CFML Reference.

For more information on ColdFusion data types, see Using ColdFusion Variables.

© 2014 Adobe Systems Incorporated. All rights reserved.

Flow control

ColdFusion provides several tags that let you control how a page gets executed. These tags generally correspond to programming language flow control statements, such as if, then, and else. The following tags provide ColdFusion flow control:

Tags	Purpose
`cfif`, `cfelseif`, `cfelse`	Select sections of code based on whether expressions are True or False.
`cfswitch`, `cfcase`, `cfdefaultcase`	Select among sections of code based on the value of an expression. Case processing is not limited to True and False conditions.
`cfloop`, `cfbreak`	Loop through code based on any of the following values: entries in a list, keys in a structure or external object, entries in a query column, an index, or the value of a conditional expression.
`cfabort`, `cfexit`	End processing of a ColdFusion page or custom tag.

CFScript also provides a set of flow-control statements. For information on using flow-control statements in CFScript, see Extending ColdFusion Pages with CFML Scripting. For more details on using flow-control tags, see the reference pages for these tags in the *CFML Reference.*

cfif, cfelseif, and cfelse

The `cfif`, `cfelseif`, and `cfelse` tags provide if-then-else conditional processing, as follows:

1. The `cfif` tag tests a condition and executes its body if the condition is True.
2. If the preceding `cfif` (or `cfelseif`) test condition is False, the `cfelseif` tag tests another condition and executes its body if that condition is True.
3. The `cfelse` tag can optionally follow a `cfif` tag and zero or more `cfelseif` tags. Its body executes if all the preceding tags' test conditions are False.The following example shows the use of the `cfif`, `cfelseif`, and `cfelse`tags. If the value of the type variable is "Date," the date displays; if the value is "Time," the time displays; otherwise, both the time and date display.

```
<cfif type IS "Date">
 <cfoutput>#DateFormat(Now())#</cfoutput>
<cfelseif type IS "Time">
 <cfoutput>#TimeFormat(Now())#</cfoutput>
<cfelse>
 <cfoutput>#TimeFormat(Now())#, #DateFormat(Now())#</cfoutput>
</cfif>
```

cfswitch, cfcase, and cfdefaultcase

The `cfswitch`, `cfcase`, and `cfdefaultcase` tags let you select among different code blocks based on the value of an expression. ColdFusion processes these tags as follows:

© 2014 Adobe Systems Incorporated. All rights reserved.

1. The `cfswitch` tag evaluates an expression. The `cfswitch` tag body contains one or more `cfcase` tags and optionally includes `cfdefaultcase` tag.
2. Each `cfcase` tag in the `cfswitch` tag body specifies a value or set of values. If a value matches the value determined by the expression in the `cfswitch` tag, ColdFusion runs the code in the body of the `cfcase` tag and then exits the `cfswitch` tag. If two `cfcase` tags have the same condition, ColdFusion generates an error.
3. If none of the `cfcase` tags match the value determined by the `cfswitch` tag, and the `cfswitch` tag body includes a `cfdefaultcase` tag, ColdFusion runs the code in the `cfdefaultcase` tag body.

Note

Although the `cfdefaultcase` tag does not have to follow all `cfcase` tags, it is good programming practice to place it at the end of the `cfswitch` statement.

The `cfswitch` tag provides better performance than a cfif tag with multiple `cfelseif` tags, and is easier to read. Switch processing is commonly used when different actions are required based on a string variable such as a month or request identifier.
The following example shows switch processing:

```
<cfoutput query = "GetEmployees">
<cfswitch expression = #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>
 </cfdefaultcase>
</cfswitch>
</cfoutput>
```

cfloop and cfbreak

The `cfloop` tag loops through the tag body zero or more times based on a condition specified by the tag attributes. The `cfbreak` tag exits a `cfloop` tag.

cfloop

The `cfloop` tag provides the following types of loops:

Loop type	Description
Index	Loops through the body of the tag and increments a counter variable by a specified amount after each loop until the counter reaches a specified value.

© 2014 Adobe Systems Incorporated. All rights reserved.

Conditional	Checks a condition and runs the body of the tag if the condition is True.
Query	Loops through the body of the tag once for each row in a query.
List, file, or array	Loops through the body of the tag once for each entry in a list, each line in a file, or each item in an array.
Collection	Loops through the body of the tag once for each key in a ColdFusion structure or item in a COM/DCOM object.

The following example shows a simple index loop:

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

The following example shows a simple conditional loop. The code does the following:

1. Sets up a ten-element array with the word "kumquats" in the fourth entry.
2. Loops through the array until it encounters an array element containing "kumquats" or it reaches the end of the array.
3. Prints the value of the Boolean variable that indicates whether it found the word *kumquats*and the array index at which it exited the loop.

```
<cfset myArray = ArrayNew(1)>
<!--- Use ArraySet to initialize the first ten elements to 123 --->
<cfset ArraySet(myArray, 1, 10, 123)>
<cfset myArray[4] = "kumquats">

<cfset foundit = False>
<cfset i = 0>
<cfloop condition = "(NOT foundit) AND (i LT ArrayLen(myArray))">
 <cfset i = i + 1>
 <cfif myArray[i] IS "kumquats">
  <cfset foundit = True>
 </cfif>
</cfloop>
<cfoutput>
i is #i#<br>
foundit is #foundit#<br>
</cfoutput>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

Note

You can get an infinite conditional loop if you do not force an end condition. In this example, the loop is infinite if you omit the `<cfset i = i + 1>` statement. To end an infinite loop, stop the ColdFusion application server.

cfbreak

The `cfbreak` tag exits the `cfloop` tag. You typically use it in a `cfif` tag to exit the loop if a particular condition occurs. The following example shows the use of a `cfbreak` tag in a query loop:

```
<cfloop query="fruitOrder">
 <cfif fruit IS "kumquat">
  <cfoutput>You cannot order kumquats!<br></cfoutput>
  <cfbreak>
 </cfif>
 <cfoutput>You have ordered #quantity# #fruit#.<br></cfoutput>
</cfloop>
```

cfabort and cfexit

The `cfabort` tag stops processing of the current page at the location of the `cfabort` tag. ColdFusion returns to the user or calling tag everything that was processed before the `cfabort` tag. You can optionally specify an error message to display. You can use the `cfabort` tag as the body of a `cfif` tag to stop processing a page when a condition, typically an error, occurs.
The `cfexit` tag controls the processing of a custom tag, and can only be used in ColdFusion custom tags. For more information see, *Terminating tag execution* in Executing custom tags and the *CFML Reference*.

© 2014 Adobe Systems Incorporated. All rights reserved.

Character case

ColdFusion is not case sensitive. For example, the following all represent the `cfset` tag: `cfset`, `CFSET`, `CFSet`, and even `cfsEt.` However, get in the habit of consistently using the same case rules in your programs; for example:

- Develop consistent rules for case use, and stick to them. If you use lowercase characters for some tag names, use them for all tag names.
- Always use the same case for a variable. For example, do not use both myvariable and MyVariable to represent the same variable on a page.Follow these rules to prevent errors on application pages where you use both CFML and case-sensitive languages, such as JavaScript.

© 2014 Adobe Systems Incorporated. All rights reserved.

Special characters

The double-quotation marks ("), single-quotation mark ('), and number sign (#) characters have special meaning to ColdFusion. To include any of them in a string, double the character; for example, use ## to represent a single # character.

The need to escape the single- and double-quotation marks is context sensitive. Inside a double-quoted string, you do not need to escape single-quotation mark (apostrophe) characters. Inside a single-quoted string, you do not escape double-quotation mark characters.

The following example illustrates escaping special characters, including the use of mixed single- and double-quotation marks:

```
<cfset mystring = "We all said ""Happy birthday to you.""">
<cfset mystring2 = 'Then we said "How old are you now?"'>
<cfoutput>
 #mystring#<br>
 #mystring2#<br>
 Here is a number sign: ##
</cfoutput>
```

The output looks as follows:

```
We all said "Happy birthday to you."
Then we said "How old are you now?"
Here is a number sign: #
```

© 2014 Adobe Systems Incorporated. All rights reserved.

Reserved words in ColdFusion

As with any programming tool, you cannot use just any word or name for ColdFusion variables, UDFs and custom tags. Avoid using any name that can be confused with a ColdFusion element. In some cases, if you use a word that ColdFusion uses — for example, a built-in structure name — you can overwrite the ColdFusion data.
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. For a complete list of reserved words, see the *CFML Reference*.

- Built-in function names, such as Now or Hash
- Scope names, such as Form or Session
- Any name starting with cf. However, when you call a CFML custom tag directly, you prefix the custom tag page name with cf_.
- 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
  Also, do 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. (For more information on form field validation, see Introduction to Retrieving and Formatting Data.)

- _integer
- _float
- _range
- _date
- _time
- _eurodate
  Because ColdFusion is not case-sensitive, all of the following are reserved words: IS, Is, iS, and is.

© 2014 Adobe Systems Incorporated. All rights reserved.

cfscript tag

CFScript is a language within a language. CFScript is a scripting language that is similar to JavaScript but is simpler to use. Also, unlike JavaScript, CFScript only runs on the ColdFusion server; it does not run on the client system. A CFScript script can use all ColdFusion functions and all ColdFusion variables that are available in the script's scope. CFScript provides a compact and efficient way to write ColdFusion logic. Typical uses of CFScript include:

- Simplifying and speeding variable setting
- Building compact flow control structures
- Encapsulating business logic in user-defined functions
  The following sample script populates an array and locates the first array entry that starts with the word "key". It shows several of the elements of CFScript, including setting variables, loop structures, script code blocks, and function calls. Also, the code uses a `cfoutput` tag to display its results. Although you can use CFScript for output, the `cfoutput` tag is often easier to use.

```
<cfscript>
strings = ArrayNew(1);
strings[1]="the";
strings[2]="key to our";
strings[4]="idea";
for( i=1 ; i LE 4 ; i = i+1 )
{
 if(Find("key",strings[i],1))
  break; }
</cfscript>
<cfoutput>Entry #i# starts with "key"</cfoutput><br>
```

You use CFScript to create user-defined functions.
For more information on CFScript, see Extending ColdFusion Pages with CFML Scripting. For more information on user-defined functions, see Writing and Calling User-Defined Functions.

#back to top

© 2014 Adobe Systems Incorporated. All rights reserved.

Elvis operator

The support has been provided in ColdFusion for the Elvis operator (?:). The Elvis operator is primarily used to assign the ‘right default’ for a variable or an expression. In an expression, if the resultant value is not defined, then the object will be assigned to the left most part of the expression otherwise a default value (define at the right most part) will be assigned.

For instance,

```
myDisplayName = userName ?: “Anonymous”;
```

In the above example, if userName is defined, it will be assigned to the myDisplayName variable. If the userName is not defined, the value “Anonymous” will be assigned to the myDisplayName variable.

See the following example:

```
employeeName = getEmployeeName(ID) ?: “Joe”;
```

In the above example, if getEmployeeName(ID) does not return any value, the value “Joe” will be assigned to the employeeName variable.

Similiarly, you can use this operator for Struct:

```
securityNumber = securityStruct[‘Joe’] ?: “”;
```

© 2014 Adobe Systems Incorporated. All rights reserved.

Using ColdFusion Variables

Adobe ColdFusion variables are the most frequently used operands in ColdFusion expressions. Variable values can be set and reset, and can be passed as attributes to CFML tags. Variables can be passed as parameters to functions, and can replace most constants.
To create and use ColdFusion variables, you should know the following:

- How variables can represent different types of data
- How the data types get converted
- How variables exist in different scopes
- How scopes are used
- How to use variables correctly

© 2014 Adobe Systems Incorporated. All rights reserved.

Creating variables

You create most ColdFusion variables by assigning them values. (You must use the `ArrayNew` function to create arrays.) Most commonly, you create variables by using the `cfset` tag. You can also use the `cfparam` tag, and assignment statements in CFScript. Tags that create data objects also create variables. For example, the `cfquery` tag creates a query object variable.
ColdFusion automatically creates some variables that provide information about the results of certain tags or operations. ColdFusion also automatically generates variables in certain scopes, such as Client and Server. For information on these special variables, see Reserved Words and Variables in the *CFML Reference* and the documentation of the CFML tags that create these variables.
ColdFusion generates an error when it tries to use a variable before it is created. This can happen, for example, when processing data from an incompletely filled form. To prevent such errors, test for the variable's existence before you use it. For more information on testing for variable existence, see Ensuring variable existence.
For more information on how to create variables, see *Creating and using variables in scopes* in the About scopes.

Variable naming rules

ColdFusion variable names, including form field names and custom function and ColdFusion component argument names, must conform to Java naming rules and the following guidelines:

- A variable name must begin with a letter, underscore, or Unicode currency symbol.
- The initial character can by followed by any number of letters, numbers, underscore characters, and Unicode currency symbols.
- A variable name cannot contain spaces.
- A query result is a type of variable, so it overwrites a local variable with the same name.
- ColdFusion variables are not case sensitive. However, consistent capitalization makes the code easier to read.
- When creating a form with fields that are used in a query, match form field names with the corresponding database field names.
- Periods separate the components of structure or object names. They also separate a variable scope from the variable name. You cannot use periods in simple variable names, with the exception of variables in the Cookie and Client scopes. For more information on using periods, see Using periods in variable references.
  The following rule applies to variable names, but does not apply to form field and argument names:

- Prefix each variable's name with its scope. Although some ColdFusion programmers do not use the Variables prefix for local variable names, use prefixes for all other scopes. Using scope prefixes makes variable names clearer and increases code efficiency. In many cases, you must prefix the scope. For more information, see About scopes.

> **Note**
>
> In some cases, when you use an existing variable name, you must enclose it with number signs (#) to allow ColdFusion to distinguish it from string or HTML text, and to insert its value, as opposed to its name. For more information, see Using number signs.

Variable characteristics

You can classify a variable using the following characteristics:

- The data type of the variable value, which indicates the kind of information a variable represents, such as number, string, or date
- The scope of the variable, which indicates where the information is available and how long the variable persists.

© 2014 Adobe Systems Incorporated. All rights reserved.

Data types- Developing guide

Data types

ColdFusion is often referred to as *typeless* because you do not assign types to variables and ColdFusion does not associate a type with the variable name. However, the data that a variable represents does have a type, and the data type affects how ColdFusion evaluates an expression or function argument. ColdFusion can automatically convert many data types into others when it evaluates expressions. For simple data, such as numbers and strings, the data type is unimportant until the variable is used in an expression or as a function argument.
ColdFusion variable data belongs to one of the following type categories:

- **Simple** One value. Can use directly in ColdFusion expressions. Include numbers, strings, Boolean values, and date-time values.
- **Binary** Raw data, such as the contents of a GIF file or an executable program file.
- **Complex** ** A container for data. Generally represent more than one value. ColdFusion built-in complex data types include arrays, structures, queries, and XML document objects. You cannot use a complex variable, such as an array, directly in a ColdFusion expression, but you can use simple data type elements of a complex variable in an expression. For example, with a one-dimensional array of numbers called myArray, you cannot use the expression myArray * 5. However, you could use an expression myArray3 * 5 to multiply the third element in the array by five.
- **Objects** Complex constructs. Often encapsulate both data and functional operations. The following table lists the types of objects that ColdFusion can use, and identifies the chapters that describe how to use them:

Object type	See
Component Object Model (COM)	Integrating COM and CORBA Objects in CFML Applications
Common Object Request Broker Architecture (CORBA)	Integrating COM and CORBA Objects in CFML Applications
Java	Integrating JEE and Java Elements in CFML Applications
ColdFusion component	Building and Using ColdFusion Components
Web service	Using Web Services

Data type notes

Although ColdFusion variables do not have types, it is often convenient to use "variable type" as a shorthand for the type of data that the variable represents.
ColdFusion can validate the type of data contained in form fields and query parameters. For more information, see *T esting for a variables existence* in the Ensuring variable existence and *Using cfqueryparam* in the Enhancing security with cfqueryparam. The `cfdump` tag displays the entire contents of a variable, including ColdFusion complex data structures. It is an excellent tool for debugging complex data and the code that handles it.ColdFusion provides the following functions for identifying the data type of a variable:

- IsArray
- IsBinary
- IsBoolean
- IsImage

© 2014 Adobe Systems Incorporated. All rights reserved.

- IsNumericDate
- IsObject
- IsPDFObject
- IsQuery
- IsSimpleValue
- IsStruct
- IsXmlDoc
  ColdFusion also includes the following functions for determining whether a string can be represented as or converted to another data type:
- IsDate
- IsNumeric
- IsXML
  ColdFusion does not use a null data type. However, if ColdFusion receives a null value from an external source such as a database, a Java object, or some other mechanism, it maintains the null value until you use it as a simple value. At that time, ColdFusion converts the null to an empty string ("""). Also, you can use the JavaCast function in a call to a Java object to convert a ColdFusion empty string to a Java null.

Numbers

ColdFusion supports integers and real numbers. You can intermix integers and real numbers in expressions; for example, 1.2 + 3 evaluates to 4.2.

Integers

ColdFusion supports integers between -2,147,483,648 and 2,147,483,647 (32-bit signed integers). You can assign a value outside this range to a variable, but ColdFusion initially stores the number as a string. If you use it in an arithmetic expression, ColdFusion converts it into a floating-point value, preserving its value, but losing precision as the following example shows:

```
<cfset mybignum=12345678901234567890>
<cfset mybignumtimes10=(mybignum * 10)>
<cfoutput>mybignum is: #mybignum#</cfoutput><br>
<cfoutput>mybignumtimes10 is: #mybignumtimes10# </cfoutput><br>
```

This example generates the following output:
mybignum is: 12345678901234567890
mybignumtimes10 is: 1.23456789012E+020

Real numbers

Real numbers, numbers with a decimal part, are also known as floating point numbers. ColdFusion real numbers can range from approximately -10300 to approximately 10300. A real number can have up to 12 significant digits. As with integers, you can assign a variable a value with more digits, but the data is stored as a string. The string is converted to a real number, and can lose precision, when you use it in an arithmetic expression.
You can represent real numbers in scientific notation. This format is _x_E_y_, where *x* is a positive or negative real number in the range 1.0 (inclusive) to 10 (exclusive), and *y* is an integer. The value of a number in scientific notation is x times 10y. For example, 4.0E^2^ is 4.0 times 10^2^, which equals 400. Similarly, 2.5E^-2^ is 2.5 times 10^-2^, which equals 0.025. Scientific notation is useful for writing very large and very small numbers.

BigDecimal numbers

ColdFusion does not have a special BigDecimal data type for arbitrary length decimal numbers such as 1234567890987564.234678503059281. Instead, it represents such numbers as strings. ColdFusion does, however,

© 2014 Adobe Systems Incorporated. All rights reserved.

have a PrecisionEvaluate function that can take an arithmetic expression that uses BigDecimal values, calculate the expression, and return a string with the resulting BigDecimal value. For more information, see PrecisionEvaluate in the *CFML Reference*.

Strings

In ColdFusion, text values are stored in *strings*. You specify strings by enclosing them in either single- or double-quotation marks. For example, the following two strings are equivalent:
"This is a string"
'This is a string'
You can write an empty string in the following ways:

- "" (a pair of double-quotation marks with nothing in between)
- '' (a pair of single-quotation marks with nothing in between)
  Strings can be any length, limited by the amount of available memory on the ColdFusion server. However, the default size limit for long text retrieval (CLOB) is 64K. The ColdFusion Administrator lets you increase the limit for database string transfers, but doing so can reduce server performance. To change the limit, select the Enable retrieval of long text option on the Advanced Settings page for the data source.

Escaping quotation marks and number signs

To include a single-quotation character in a string that is single-quoted, use two single-quotation marks (known as escaping the single-quotation mark). The following example uses escaped single-quotation marks:

```
<cfset myString='This is a single-quotation mark: '' This is a double-quotation
mark: "'>
<cfoutput>#mystring#</cfoutput><br>
```

To include a double-quotation mark in a double-quoted string, use two double-quotation marks (known as escaping the double-quotation mark). The following example uses escaped double-quotation marks:

```
<cfset myString="This is a single-quotation mark: ' This is a double-quotation mark:
"">
<cfoutput>#mystring#</cfoutput><br>
```

Because strings can be in either double-quotation marks or single-quotation marks, both of the preceding examples display the same text:
This is a single-quotation mark: ' This is a double-quotation mark: "
To insert a number sign (#) in a string, you must escape the number sign, as follows:

```
"This is a number sign ##"
```

Lists

ColdFusion includes functions that operate on lists, but it does not have a list data type. In ColdFusion, a *list* is just a string that consists of multiple entries separated by delimiter characters.

© 2014 Adobe Systems Incorporated. All rights reserved.

The default delimiter for lists is the comma. If you use any other character to separate list elements, you must specify the delimiter in the list function. You can also specify multiple delimiter characters. For example, you can tell ColdFusion to interpret a comma or a semicolon as a delimiter, as the following example shows:

```
<cfset MyList="1,2;3,4;5">
<cfoutput>
List length using ; and , as delimiters: #listlen(Mylist, ";,")#<br>
List length using only , as a delimiter: #listlen(Mylist)#<br>
</cfoutput>
```

This example displays the following output:
List length using ; and , as delimiters: 5
List length using only , as a delimiter: 3
Each delimiter must be a single character. For example, you cannot tell ColdFusion to require two hyphens in a row as a delimiter.
If a list has two delimiters in a row, ColdFusion ignores the empty element. For example, if MyList is "1,2,,3,,4,,,5" and the delimiter is the comma, the list has five elements, and list functions treat it the same as "1,2,3,4,5".

Boolean values

A *Boolean* value represents whether something is true or false. ColdFusion has two special constants True and False to represent these values. For example, the Boolean expression 1 IS 1 evaluates to True. The expression `"Monkey" CONTAINS "Money"` evaluates to False.
You can use Boolean constants directly in expressions, as in the following example:

```
<cfset UserHasBeenHere = True>
```

In Boolean expressions, True, nonzero numbers, and the strings "Yes", "1", "True" are equivalent; and False, 0, and the strings "No", "0", and "False" are equivalent.
Boolean evaluation is not case sensitive. For example, True, TRUE, and true are equivalent.

Date-Time values

ColdFusion can perform operations on date and time values. Date-time values identify a date and time in the range 100 AD to 9999 AD. Although you can specify just a date or a time, ColdFusion uses one data type representation, called a date-time object, for date, time, and date and time values.
ColdFusion provides many functions to create and manipulate date-time values and to return all or part of the value in several different formats.
You can enter date and time values directly in a `cfset` tag with a constant, as follows:

```
<cfset myDate = "October 30, 2001">
```

When you do this, ColdFusion stores the information as a string. If you use a date-time function, ColdFusion stores the value as a date-time object, which is a separate simple data type. When possible, use date-time functions such as `CreateDate` and `CreateTime` to specify dates and times, because these functions can prevent you from specifying the date or time in an invalid format and they create a date-time object immediately.

Date and time formats

© 2014 Adobe Systems Incorporated. All rights reserved.

You can directly enter a date, time, or date and time, using standard U.S. date formats. ColdFusion processes the two-digit-year values 0 to 29 as twenty-first century dates; it processes the two-digit-year values 30 to 99 as twentieth century dates. Time values can include units down to seconds. The following table lists valid date and time formats:

To specify	Use these formats
Date	October 30, 2003Oct 30, 2003Oct. 30, 200310/30/032003-10-3010-30-2003
Time	02:34:122:34a2:34am02:34am2am
Date and Time	Any combination of valid date and time formats, such as these:October 30, 2003 02:34:12Oct 30, 2003 2:34aOct. 30, 2001 2:34am10/30/03 02:34am2003-10-30 2am10-30-2003 2am

Locale-specific dates and times

ColdFusion provides several functions that let you input and output dates and times (and numbers and currency values) in formats that are specific to the current locale. A *locale* identifies a language and locality, such as English (US) or French (Swiss). Use these functions to input or output dates and times in formats other than the U.S. standard formats. (Use the `SetLocale` function to specify the locale.) The following example shows how to do this:

```
<cfset oldlocale = SetLocale("French (Standard)")>
<cfoutput>#LSDateFormat(Now(), "ddd, dd mmmm, yyyy")#</cfoutput>
```

This example outputs a line like the following:
mar., 03 juin, 2003
For more information on international functions, see Developing Globalized Applications and the *CFML Reference*.

How ColdFusion stores dates and times

ColdFusion stores and manipulates dates and times as *date-time objects*. Date-time objects store data on a timeline as real numbers. This storage method increases processing efficiency and directly mimics the method used by many database systems. In date-time objects, one day is equal to the difference between two successive integers. The time portion of the date-and-time value is stored in the fractional part of the real number. The value 0 represents 12:00 AM 12/30/1899.
Although you can use arithmetic operations to manipulate date-and-time values directly, this method can result in code that is difficult to understand and maintain. Use the ColdFusion date-time manipulation functions instead. For information on these functions, see the *CFML Reference*.

Binary data type and binary encoding

Binary data (also referred to as a *binary object*) is raw data, such as the contents of a GIF file or an executable program file. You do not normally use binary data directly, but you can use the `cffile` tag to read a binary file into a variable, typically for conversion to a string binary encoding before transmitting the file using e-mail.
A string binary encoding represents a binary value in a string format that can be transmitted over the web. ColdFusion supports three binary encoding formats:

© 2014 Adobe Systems Incorporated. All rights reserved.

Encoding	Format
Base64	Encodes the binary data in the lowest six bits of each byte. It ensures that binary data and non-ANSI character data can be transmitted using e-mail without corruption. The Base64 algorithm is specified by IETF RFC 2045, at www.ietf.org/rfc/rfc2045.txt.
Hex	Uses two characters in the range 0-9 and A-F represent the hexadecimal value of each byte; for example, 3A.
UU	Uses the UNIX UUencode algorithm to convert the data.

ColdFusion provides the following functions that convert among string data, binary data, and string encoded binary data:

Function	Description
BinaryDecode	Converts a string that contains encoded binary data to a binary object.
BinaryEncode	Converts binary data to an encoded string.
CharsetDecode	Converts a string to binary data in a specified character encoding.
CharsetEncode	Converts a binary object to a string in a specified character encoding.
ToBase64	Converts string and binary data to Base64 encoded data.
ToBinary	Converts Base64 encoded data to binary data. The BinaryDecode function provides a superset of the ToBase64 functionality.
ToString	Converts most simple data types to string data. It can convert numbers, date-time objects, and Boolean values. (It converts date-time objects to ODBC timestamp strings.) Adobe recommends that you use the CharsetEncode function to convert binary data to a string in new applications.

Complex data types

Arrays, structures, and queries are ColdFusion built-in complex data types. Structures and queries are sometimes referred to as objects, because they are containers for data, not individual data values.

© 2014 Adobe Systems Incorporated. All rights reserved.

For details on using arrays and structures, see Using Arrays and Structures.

Arrays

Arrays are a way of storing multiple values in a table-like format that can have one or more dimensions. You create arrays using a function or an assignment statement:

- The ColdFusion `ArrayNew`function creates an array and specifies its initial dimensions. For example, the following line creates an empty two-dimensional array:

```
<cfset myarray=ArrayNew(2)>
```

- A direct assignment creates an array and populates an array element. For example, the following line creates a two-dimensional array and sets the value of row 1 column 2 to the current date.

```
<cfset myarray[1][2]=Now()>
```

You reference elements using numeric indexes, with one index for each dimension, as shown in the preceding example.
You can create arrays with up to three dimensions directly. However, there is no limit on array size or maximum dimension. To create arrays with more than three dimensions, create arrays of arrays.
After you create an array, you can use functions or direct references to manipulate its contents.
When you assign an existing array to a new variable, ColdFusion creates a new array and copies the old array's contents to the new array. The following example creates a copy of the original array:

```
<cfset newArray=myArray>
```

For more information on using arrays, see Using Arrays and Structures.

Structures

ColdFusion *structures* consist of key-value pairs, where the keys are text strings and the values can be any ColdFusion data type, including other structures. Structures let you build a collection of related variables that are grouped under a single name. To create a structure, use the ColdFusion `StructNew` function. For example, the following line creates a new, empty, structure called *depts*:

```
<cfset depts=StructNew()>
```

You can also create a structure by assigning a value in the structure. For example, the following line creates a new structure called *MyStruct* with a key named *MyValue*, equal to 2:

```
<cfset MyStruct.MyValue=2>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

> **Note**
>
> In ColdFusion versions through ColdFusion 7, this line created a Variables scope variable named "MyStruct.MyValue" with the value 2.

After you create a structure, you can use functions or direct references to manipulate its contents, including adding key-value pairs.
You can use either of the following methods to reference elements stored in a structure:

- *StructureName*`\.`*KeyName*
- *StructureName*`["`*KeyName*`"]`
  The following examples show these methods:

```
depts.John="Sales"
depts["John"]="Sales"
```

When you assign an existing structure to a new variable, ColdFusion does *not* create a new structure. Instead, the new variable accesses the same data (location) in memory as the original structure variable. In other words, both variables are references to the same object.
For example, the following line creates a new variable, myStructure2, that is a reference to the same structure as the myStructure variable:

```
<cfset myStructure2=myStructure>
```

When you change the contents of myStructure2, you also change the contents of myStructure. To copy the contents of a structure, use the ColdFusion `Duplicate` function, which copies the contents of structures and other complex data types.
Structure key names can be the names of complex data objects, including structures or arrays. This lets you create arbitrarily complex structures.
For more information on using structures, see Using Arrays and Structures.

Queries

A *query object*, sometimes referred to as a query, query result, or recordset, is a complex ColdFusion data type that represents data in a set of named columns, like the columns of a database table. Many tags can return data as a query object, including the following:

- `cfquery`
- `cfdirectory`
- `cfhttp`
- `cfldap`
- `cfpop`
- `cfprocresult`
  In these tags, the `name` attribute specifies the query object's variable name. The `QueryNew` function also creates query objects.
  When you assign a query to a new variable, ColdFusion does *not* copy the query object. Instead, both names point to the same recordset data. For example, the following line creates a new variable, myQuery2, that references the same recordset as the myQuery variable:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cfset myQuery2 = myQuery>
```

If you make changes to data in myQuery, myQuery2 also shows those changes.
You reference query columns by specifying the query name, a period, and the column name; for example:

```
myQuery.Dept_ID
```

When you reference query columns inside tags, such as `cfoutput` and `cfloop`, in which you specify the query name in a tag attribute, you do not have to specify the query name.
You can access query columns as if they are one-dimensional arrays. For example, the following line assigns the contents of the Employee column in the second row of the myQuery query to the variable myVar:

```
<cfset myVar = myQuery.Employee[2]>
```

Note

You cannot use array notation to reference a row (of all columns) of a query. For example, myQuery2 does not reference the second row of the myQuery query object.

Working with structures and queries

Because structure variables and query variables are references to objects, the rules in the following sections apply to both types of data.

Multiple references to an object

When multiple variables reference a structure or query object, the object continues to exist as long as at least one reference to the object exists. The following example shows how this works:

```
<cfscript> depts = structnew();</cfscript>
<cfset newStructure=depts>
<cfset depts.John="Sales">
<cfset depts=0>
<cfoutput>
 #newStructure.John#<br>
 #depts#
</cfoutput>
```

This example displays the following output:
Sales
0
After the `<cfset depts=0>` tag executes, the depts variable does not reference a structure; it is a simple variable with the value 0. However, the variable newStructure still refers to the original structure object.

© 2014 Adobe Systems Incorporated. All rights reserved.

Assigning objects to scopes

You can give a query or structure a different scope by assigning it to a new variable in the other scope. For example, the following line creates a server variable, Server.SScopeQuery, using the local myquery variable:

```
<cfset Server.SScopeQuery = myquery>
```

To clear the server scope query variable, reassign the query object, as follows:

```
<cfset Server.SScopeQuery = 0>
```

This line deletes the reference to the object from the server scope, but does not remove any other references that can exist.

Copying and duplicating objects

You can use the `Duplicate` function to make a true copy of a structure or query object. Changes to the copy do not affect the original.

Using a query column

When you are not inside a tag such as `cfloop`, `cfoutput`, or `cfmail` that has a `query` attribute, you can treat a query column as an array. However, query column references do not always behave as you might expect. This section explains the behavior of references to query columns using the results of the following `cfquery` tag in its examples:

```
<cfquery dataSource="cfdocexamples" name="myQuery">
 SELECT FirstName, LastName
 FROM Employee
</cfquery>
```

To reference elements in a query column, use the row number as an array index. For example, both of the following lines display the word "ben":

```
<cfoutput> #myQuery.Firstname[1]# </cfoutput><br>
<cfoutput> #myQuery["Firstname"][1]# </cfoutput><br>
```

ColdFusion behavior is less straightforward, however, when you use the query column references myQuery.Firstname and myQuery"Firstname" without using an array index. The two reference formats produce different results.
If you reference myQuery.Firstname, ColdFusion automatically converts it to the first row in the column. For example, the following lines print the word "ben":

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cfset myCol = myQuery.Firstname >
<cfoutput>#mycol#</cfoutput>
```

But the following lines display an error message:

```
<cfset myCol = myQuery.Firstname >
<cfoutput>#mycol[1]#</cfoutput><br>
```

If you reference `Query["Firstname"]`, ColdFusion does not automatically convert it to the first row of the column. For example, the following line results in an error message indicating that ColdFusion cannot convert a complex type to a simple value:

```
<cfoutput> #myQuery['Firstname']# </cfoutput><br>
```

Similarly, the following lines print the name "marjorie", the value of the second row in the column:

```
<cfset myCol = myQuery["Firstname"]>
<cfoutput>#mycol[2]#</cfoutput><br>
```

However, when you make an assignment that requires a simple value, ColdFusion automatically converts the query column to the value of the first row. For example, the following lines display the name "ben" twice:

```
<cfoutput> #myQuery.Firstname# </cfoutput><br>
<cfset myVar= myQuery['Firstname']>
<cfoutput> #myVar# </cfoutput><br>
```

FUNCTION is a ColdFusion data type in ColdFusion 10

For example,

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cfscript>
public FUNCTIOn FUNCTION a()
{
return variables.b;
}
public FUNCTION b()
{
return "Hal";
}
writeoutput(a());
writedump(a().getMetadata());
writedump(a.getMetadata());
writedump(x.getMetadata());
</cfscript>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

Using periods in variable references

ColdFusion uses the period (.) to separate elements of a complex variable such as a structure, query, XML document object, or external object, as in MyStruct.KeyName. A period also separates a variable scope identifier from the variable name, as in Variables.myVariable or CGI.HTTP_COOKIE.
With the exception of Cookie and Client scope variables, which must always be simple variable types, you cannot normally include periods in simple variable names. However, ColdFusion makes some exceptions that accommodate legacy and third-party code that does not conform to this requirement.
For more information, see About scopes, Using Arrays and Structures, and Using XML and WDDX.

Understanding variables and periods

The following descriptions use a sample variable named MyVar.a.b to explain how ColdFusion uses periods when getting and setting the variable value.

Getting a variable

ColdFusion can correctly get variable values even if the variable name includes a period. For example, the following set of steps shows how ColdFusion gets MyVar.a.b, as in `<cfset Var2 = myVar.a.b>` or `IsDefined(myVar.a.b)`:

1. Looks for myVar in an internal table of names (the symbol table).
2. If myVar is the name of a complex object, including a scope, looks for an element named a in the object.If myVar is not the name of a complex object, checks whether myVar.a is the name of a complex object and skips step 3.
3. If myVar is the name of a complex object, checks whether a is a complex object.
4. If a or myVar.a is the name of a complex object, checks whether b is the name of a simple variable, and returns the value of b.If myVar is a complex object but a is not a complex object, checks whether a.b is the name of a simple variable and returns its value.If myVar.a is not a complex object, checks whether myVar.a.b is the name of a simple variable and returns its value.
   This way, ColdFusion correctly resolves the variable name and can get its value.
   You can also use array notation to get a simple variable with a name that includes periods. In this form of array notation, you use the scope name (or the complex variable that contains the simple variable) as the "array" name. You place the simple variable name, in single- or double-quotation marks, inside the brackets. Using array notation is more efficient than using plain dot notation because ColdFusion does not have to analyze and look up all the possible key combinations. For example, both of the following lines write the value of myVar.a.b, but the second line is more efficient than the first:

```
<cfoutput>myVar.a.b is: #myVar.a.b#<br></cfoutput>
<cfoutput>myVar.a.b is: #Variables["myVar.a.b"]#<br></cfoutput>
```

Setting a variable

ColdFusion cannot be as flexible when it sets a variable value as when it gets a variable, because it must determine the type of variable to create or set. Therefore, the rules for variable names that you set are stricter. Also, the rules vary depending on whether the first part of the variable name is the Cookie or Client scope identifier.
For example, assume that you have the following code:

```
<cfset myVar.a.b = "This is a test">
```

© 2014 Adobe Systems Incorporated. All rights reserved.

If a variable myVar does not exist, it does the following:

1. Creates a structure named myVar.
2. Creates a structure named a in the structure myVar.
3. Creates a key named b in myVar.a.
4. Gives it the value "This is a test".
   If either myVar or myVar.a exist and neither one is a structure, ColdFusion generates an error.
   In other words, ColdFusion uses the same rules as for getting a variable to resolve the variable name until it finds a name that does not exist yet. It then creates any structures that are needed to create a key named b inside a structure, and assigns the value to the key.
   However, if the name before the first period is either Cookie or Client, ColdFusion uses a different rule. It treats all the text (including any periods) that follow the scope name as the name of a simple variable, because Cookie and Client scope variables must be simple. If you have the following code, you see that ColdFusion creates a single, simple Client scope variable named myVar.a.b:

```
<cfset Client.myVar.a.b = "This is a test">
<cfdump var=#Client.myVar.a.b#>
```

Creating variables with periods

Avoid creating the names of variables (except for dot notation in structures) that include periods. However, ColdFusion provides mechanisms for handling cases where you must do so, for example, to maintain compatibility with names of variables in external data sources or to integrate your application with existing code that uses periods in variable names. The following sections describe how to create simple variable names that include periods.

Using brackets to create variables with periods

You can create a variable name that includes periods by using associative array structure notation, as described in *Structure notation* in the About arrays. To do so, you must do the following:

- Reference the variable as part of a structure. You can always do this, because ColdFusion considers all scopes to be structures. For more information on scopes, see About scopes.
- Place the variable name that must include a period inside brackets and single- or double-quotation marks. The following example shows this technique:

```
<cfset Variables['My.Variable.With.Periods'] = 12>
<cfset Request["Another.Variable.With.Periods"] = "Test variable">
<cfoutput>
 My.Variable.With.Periods is: #My.Variable.With.Periods#<br>
 Request.Another.Variable.With.Periods is:
  #Request.Another.Variable.With.Periods#<br>
</cfoutput>
```

Creating Client and Cookie variables with periods

To create a Client or Cookie variable with a name that includes one or more periods, simply assign the variable a value. For example, the following line creates a Cookie named User.Preferences.CreditCard:

```
<cfset Cookie.User.Preferences.CreditCard="Discover">
```

© 2014 Adobe Systems Incorporated. All rights reserved.

© 2014 Adobe Systems Incorporated. All rights reserved.

Data type conversion

ColdFusion automatically converts between data types to satisfy the requirements of an expression's operations, including a function's argument requirements. As a result, you generally don't need to be concerned about compatibility between data types and the conversions from one data type to another. However, understanding how ColdFusion evaluates data values and converts data between types can help you prevent errors and create code more effectively.

Operation-driven evaluation

Conventional programming languages enforce strict rules about mixing objects of different types in expressions. For example, in a language such as C++ or Basic, the expression `("8" * 10)` produces an error because the multiplication operator requires two numeric operands and "8" is a string. When you program in such languages, you must convert between data types to ensure error-free program execution. For example, the previous expression might have to be written as `(ToNumber("8") * 10)`.
In ColdFusion, however, the expression `("8" * 10)` evaluates to the number 80 without generating an error. When ColdFusion processes the multiplication operator, it automatically attempts to convert its operands to numbers. Since "8" can be successfully converted to the number 8, the expression evaluates to 80.
ColdFusion processes expressions and functions in the following sequence:

1. For each operator in an expression, it determines the required operands. (For example, the multiplication operator requires numeric operands and the CONTAINS operator requires string operands.)For functions, it determines the type required for each function argument. (For example, the `Min` function requires two numbers as arguments and the `Len` function requires a string.)
2. It evaluates all operands or function arguments.
3. It converts all operands or arguments whose types differ from the required type. If a conversion fails, it reports an error.

Conversion between types

Although the expression evaluation mechanism in ColdFusion is powerful, it cannot automatically convert all data. For example, the expression `"eight" * 10` produces an error because ColdFusion cannot convert the string "eight" to the number 8. Therefore, you must understand the rules for conversion between data types.
The following table explains how conversions are performed. The first column shows values to convert. The remaining columns show the result of conversion to the listed data type.

Value	As Boolean	As number	As date-time	As string
"Yes"	True	1	Error	"Yes"
"No"	False	0	Error	"No"
True	True	1	Error	"Yes"
False	False	0	Error	"No"
Number	True if Number is not 0; False otherwise.	Number	See "Date-time values" earlier in this chapter.	String representation of the number (for example, "8").

© 2014 Adobe Systems Incorporated. All rights reserved.

String	If "Yes", TrueIf "No", FalseIf it can be converted to 0, FalseIf it can be converted to any other number, True	If it represents a number (for example, "1,000" or "12.36E-12"), it is converted to the corresponding number. If it represents a date-time (see next column), it is converted to the numeric value of the corresponding date-time object.	If it is an ODBC date, time, or timestamp (for example "{ts '2001-06-14 11:30:13'}", or if it is expressed in a standard U.S. date or time format, including the use of full or abbreviated month names, it is converted to the corresponding date-time value. Days of the week or unusual punctuation result in an error. Dashes, forward-slashes, and spaces are generally allowed.	String
Date	Error	The numeric value of the date-time object.	Date	An ODBC timestamp.

ColdFusion cannot convert complex types, such as arrays, queries, and COM objects, to other types. However, it can convert simple data elements of complex types to other simple data types.

Type conversion considerations

The following sections detail specific rules and considerations for converting between types.

The cfoutput tag

The `cfoutput` tag always displays data as a string. As a result, when you display a variable using the `cfoutput` tag, ColdFusion applies the type conversion rules to any non-string data before displaying it. For example, the `cfoutput` tag displays a date-time value as an ODBC timestamp.

Case-insensitivity and Boolean conversion

Because ColdFusion expression evaluation is not case sensitive, Yes, YES, and yes are equivalent; False, FALSE, and false are equivalent; No, NO, and no are equivalent; and True, TRUE, and true are equivalent.

Converting binary data

ColdFusion cannot automatically convert binary data to other data types. To convert binary data, use the `ToBase64` and `ToString` functions. For more information, see *Binary data type and binary encoding*.

Converting date and time data

© 2014 Adobe Systems Incorporated. All rights reserved.

To ensure that a date and time value is expressed as a real number, add zero to the variable. The following example shows this:

```
<cfset mynow = now()>
Use cfoutput to display the result of the now function:<br>
<cfoutput>#mynow#</cfoutput><br>
Now add 0 to the result and display it again:<br>
<cfset mynow = mynow + 0>
<cfoutput>#mynow#</cfoutput>
```

At 1:06 PM on June 6, 2003, its output looked like this:

```
Use cfoutput to display the result of the now function:
{ts '2003-06-03 13:06:44'}
Now add 0 to the result and display it again:
37775.5463426
```

Converting numeric values

When ColdFusion evaluates an expression that includes both integers and real numbers, the result is a real number. To convert a real number to an integer, use a ColdFusion function. The `Int`, `Round`, *`Fix`*, and `Ceiling` functions convert real numbers to integers, and differ in their treatment of the fractional part of the number.
If you use a hidden form field with a name that has the suffix `_integer` or `_range` to validate a form input field, ColdFusion truncates real numbers entered into the field and passes the resulting integer to the action page.
If you use a hidden form field with a name that has the suffix `_integer`, `_float`, or `_range` to validate a form input field, and the entered data contains a dollar amount (including a dollar sign) or a numeric value with commas, ColdFusion considers the input to be valid, removes the dollar sign or commas from the value, and passes the resulting integer or real number to the action page.
ColdFusion does not have an inherent data type for arbitrary precision decimal numbers (BigDecimal numbers). ColdFusion initially saves such numbers as strings, and if you use them in an expression, converts the value to a numeric type, often losing precision. You can retain precision by using the `PrecisionEvaluate` method, which evaluates string expressions using BigDecimal precision arithmetic and can return the result as a long string of numbers. For more information, see PrecisionEvaluate in the *CFML Reference*.

Evaluation and type conversion issues

The following sections explain several issues that you can encounter with type evaluation and conversion.

Comparing variables to True or False

You might expect the following two `cfif` tag examples to produce the same results:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cfif myVariable>
 <cfoutput>myVariable equals #myVariable# and is True
 </cfoutput>
</cfif>
<cfif myVariable IS True>
 <cfoutput>myVariable equals #myVariable# and is True
 </cfoutput>
</cfif>
```

However, if myVariable has a numeric value such as 12, only the first example produces a result. In the second case, the value of myVariable is not converted to a Boolean data type, because the IS operator does not require a specific data type and just tests the two values for identity. Therefore, ColdFusion compares the value 12 with the constant True. The two are not equal, so nothing is printed. If myVariable is 1, "Yes", or True, however, both examples print the same result, because ColdFusion considers these to be identical to Boolean True.
If you use the following code, the output statement does display, because the value of the variable, 12, is not equal to the Boolean value False:

```
<cfif myVariable IS NOT False>
 <cfoutput>myVariable equals #myVariable# and IS NOT False
 </cfoutput>
</cfif>
```

As a result, use the test `<cfif }}testvariable{{>`, and not use the `IS` comparison operator when testing whether a variable is True or False. This issue is a case of the more general problem of ambiguous type expression evaluation, described in the following section.

Ambiguous type expressions and strings

When ColdFusion evaluates an expression that does not require strings, including all comparison operations, such as `IS` or `GT`, it checks whether it can convert each string value to a number or date-time object. If so, ColdFusion converts it to the corresponding number or date-time value (which is stored as a number). It then uses the number in the expression.
Short strings, such as 1a and 2P, can produce unexpected results. ColdFusion can interpret a single "a" as AM and a single "P" as PM. This can cause ColdFusion to interpret strings as date-time values in cases where this was not intended.
Similarly, if the strings can be interpreted as numbers, you can get unexpected results.
For example, ColdFusion interprets the following expressions as shown:

Expression	Interpretation
`<cfif "1a" EQ "01:00">`	If 1:00am is 1:00am.
`<cfif "1P" GT "2A">`	If 1:00pm is later than 2:00am.
`<cfset age="4a"><cfset age=age + 7>`	Treat the variable age as 4:00 am, convert it to the date-time value 0.16666666667, and add 7 to make it 7.16666666667.

© 2014 Adobe Systems Incorporated. All rights reserved.

`<cfif "0.0" is "0">`	If 0 is 0.

To prevent such ambiguities when you compare strings, use the ColdFusion string comparison functions `Compare` and `CompareNoCase`, instead of the comparison operators.
You can also use the `IsDate` function to determine whether a string can be interpreted as a date-time value, or to add characters to a string before comparison to avoid incorrect interpretation.

Date-time functions and queries when ODBC is not supported

Many CFML functions, including the `Now`, `CreateDate`, `CreateTime`, and `CreateDateTime` functions, return date-time objects. ColdFusion creates Open Database Connectivity (ODBC) timestamp values when it converts date-time objects to strings. As a result, you can get unexpected results when using dates with a database driver that does not support ODBC escape sequences, or when you use SQL in a query of queries.
If you use SQL to insert data into a database or in a WHERE clause to select data from a database, and the database driver does not support ODBC-formatted dates, use the `DateFormat` function to convert the date-time value to a valid format for the driver. This rule also applies to queries of queries.
For example, the following SQL statement uses the `DateFormat` function in a query of queries to select rows that have MyDate values in the future:

```
<cfquery name="MyQofQQ" dbtype="query">
SELECT *
FROM DateQuery
WHERE MyDate >= '#DateFormat(Now())#'
</cfquery>
```

The following query of queries fails with the error message "Error:
{ts is not a valid date," because the ColdFusion `Now` function
returns an ODBC timestamp:

```
<cfquery name="MyQofQQ" dbtype="query">
SELECT *
FROM DateQuery
WHERE MyDate >= '#now()#'
</cfquery>
```

Using JavaCast with overloaded Java methods

You can overload Java methods so a class can have several identically named methods that differ only in parameter data types. At run time, the Java virtual machine attempts to resolve the specific method to use, based on the types of the parameters passed in the call. Because ColdFusion does not use explicit types, you cannot predict which version of the method the virtual machine will use.
The ColdFusion `JavaCast` function helps you ensure that the right method executes by specifying the Java type of a variable, as in the following example:

```
<cfset emp.SetJobGrade(JavaCast("int", JobGrade))>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

The `JavaCast` function takes two parameters: a string representing the Java data type and the variable whose type you are setting. You can specify the following Java data types: Boolean, int, long, float, double, and String.
For more information on the `JavaCast` function, see the *CFML Reference*.

Using quotation marks

To ensure that ColdFusion properly interprets string data, surround strings in single- or double-quotation marks. For example, ColdFusion evaluates "10/2/2001" as a string that can be converted into a date-time object. However, it evaluates 10/2/2001 as a mathematical expression, 5/2001, which evaluates to 0.00249875062469.

Examples of type conversion in expression evaluation

The following examples demonstrate ColdFusion expression evaluation.

Example 1

```
2 * True + "YES" - ('y' & "es")
```

Result value as string: "2"Explanation: (2*True) is equal to 2; ("YES"- "yes") is equal to 0; 2 + 0 equals 2.

Example 2

```
"Five is " & 5
```

Result value as string: "Five is 5"Explanation: 5 is converted to the string "5".

Example 3

```
DateFormat("October 30, 2001" + 1)
```

Result value as string: "31-Oct-01"Explanation: The addition operator forces the string "October 30, 2001" to be converted to a date-time object, and then to a number. The number is incremented by one. The DateFormat function requires its argument to be a date-time object; thus, the result of the addition is converted to a date-time object. One is added to the date-time object, moving it ahead by one day to October 31, 2001.

© 2014 Adobe Systems Incorporated. All rights reserved.

About scopes

Variables differ in how they are set (by your code or by ColdFusion), the places in your code where they are meaningful, and how long their values persist. These considerations are generally referred to as a variable's *scope*. Commonly used scopes include the Variables scope, the default scope for variables that you create, and the Request scope, which is available for the duration of an HTTP request.

> **Note**
>
> User-defined functions also belong to scopes. For more information, see *Specifying the scope of a function* in the Using UDFs effectively.

Scope types

The following table describes ColdFusion scopes:

Scope	Description
Application	Contains variables that are associated with one, named application on a server. The `cfapplication` tag `name` attribute or the Application.cfc This.name variable setting specifies the application name. For more information, see Using Persistent Data and Locking.
Arguments	Variables passed in a call to a user-defined function or ColdFusion component method. For more information, see *About the Arguments scope* in the Working with arguments and variables in functions.
Attributes	Used only in custom tag pages and threads. Contains the values passed by the calling page or `cfthread` tag in the tag's attributes. For more information, see Creating and Using Custom CFML Tags and Using ColdFusion Threads.
Caller	Used only in custom tag pages. The custom tag's Caller scope is a reference to the calling page's Variables scope. Any variables that you create or change in the custom tag page using the Caller scope are visible in the calling page's Variables scope. For more information, see Creating and Using Custom CFML Tags.
CGI	Contains environment variables identifying the context in which a page was requested. The variables available depend on the browser and server software. For a list of the commonly used CGI variables, see Reserved Words and Variables in the *CFML Reference*.

© 2014 Adobe Systems Incorporated. All rights reserved.

Client	Contains variables that are associated with one client. Client variables let you maintain state as a user moves from page to page in an application, and are available across browser sessions. By default, Client variables are stored in the system registry, but you can store them in a cookie or a database. Client variables cannot be complex data types and can include periods in their names. For more information, see Using Persistent Data and Locking.
Cookie	Contains variables maintained in a user's browser as cookies. Cookies are typically stored in a file on the browser, so they are available across browser sessions and applications. You can create memory-only Cookie variables, which are not available after the user closes the browser. Cookie scope variable names can include periods.
Flash	Variables sent by a SWF movie to ColdFusion and returned by ColdFusion to the movie. For more information, see Using the Flash Remoting Service.
Form	Contains variables passed from a Form page to its action page as the result of submitting the form. (If you use the HTML `form` tag, you must use `method="post"`.) For more information, see Introduction to Retrieving and Formatting Data.
Local (function local)	Contains variables that are declared inside a user-defined function or ColdFusion component method and exist only while a function executes. For more information, see Writing and Calling User-Defined Functions.
Request	Used to hold data that must be available for the duration of one HTTP request. The Request scope is available to all pages, including custom tags and nested custom tags, that are processed in response to the request. This scope is useful for nested (child/parent) tags. This scope can often be used in place of the Application scope, to avoid the need for locking variables. Several chapters discuss using the Request scope.
Server	Contains variables that are associated with the current ColdFusion server. This scope lets you define variables that are available to all your ColdFusion pages, across multiple applications. For more information, see Using Persistent Data and Locking.

© 2014 Adobe Systems Incorporated. All rights reserved.

Session	Contains variables that are associated with one client and persist only as long as the client maintains a session. They are stored in the server's memory and can be set to time out after a period of inactivity. For more information, see Using Persistent Data and Locking.
This	Exists only in ColdFusion components or `cffunction` tags that are part of a containing object such as a ColdFusion Struct. Exists for the duration of the component instance or containing object. Data in the This scope is accessible from outside the component or container by using the instance or object name as a prefix.
ThisTag	Used only in custom tag pages. The ThisTag scope is active for the current invocation of the tag. If a custom tag contains a nested tag, any ThisTag scope values you set before calling the nested tag are preserved when the nested tag returns to the calling tag. The ThisTag scope includes three built-in variables that identify the tag's execution mode, contain the tag's generated contents, and indicate whether the tag has an end tag.A nested custom tag can use the `cfassociate` tag to return values to the calling tag's ThisTag scope. For more information, see *Accessing tag instance data* in the Executing custom tags.
Thread	Variables that are created and changed inside a ColdFusion thread, but can be read by all code on the page that creates the thread. Each thread has a Thread scope that is a subscope of a cfthread scope. For more information, see Using ColdFusion Threads.
thread local	Variables that are available only within a ColdFusion thread. For more information, see Using ColdFusion Threads.
URL	Contains parameters passed to the current page in the URL that is used to call it. The parameters are appended to the URL in the format ?variablename = value&variablename=value...; for example www.MyCompany.com/inputpage.cfm?productCode=A12CD1510&quantity=3.

If a URL includes multiple parameters with the same name, the resulting variable in the ColdFusion URL scope consists of all parameter values separated by commas. For example, a URL of the form http://localhost/urlparamtest.cfm?param=1&param=2&param=3 results in a URL.param variable value of 1,2,3 on the ColdFusion page.|

© 2014 Adobe Systems Incorporated. All rights reserved.

Variables	The default scope for variables of any type that are created with the `cfset` and `cfparam` tags. A Variables scope variable is available only on the page on which it is created and any included pages (see also the Caller scope).Variables scope variables created in a CFC are available only to the component and its functions, and not to the page that instantiates the component or calls its functions.

To prevent data corruption, you lock code that uses Session, Application, or Server scope variables. For more information, see Using Persistent Data and Locking.

Creating and using variables in scopes

The following table shows how you create and reference variables in different scopes in your code. For more information on the mechanisms for creating variables in most scopes, see Creating variables.

Scope prefix(type)	Prefix required to reference	Where available	Created by
Application	Yes	For multiple clients in one application over multiple browser sessions. Surround code that uses application variables in `cf lock` blocks.	Specifying the prefix Application when you create the variable.
Arguments	No	Within the body of a user-defined function or ColdFusion component method.	The calling page passing an argument in the function call.
Attributes	Yes	On a custom tag page, or inside a thread	For custom tags, the calling page passing the values to a custom tag page in the custom tag's attributes. For threads, the `cfthread` tag specifying attribute values.
Caller	On the custom tag page, Yes.On the calling page, No (Variables prefix is optional).	On the custom tag page, by using the Caller scope prefix.On the page that calls the custom tag, as local variables (Variables scope).	On the custom tag page, by specifying the prefix Caller when you create the variable.On the calling page, by specifying the prefix Variables, or using no prefix, when you create the variable.

© 2014 Adobe Systems Incorporated. All rights reserved.

Cffile	Yes	Following an invocation of `cffile`.	A cffile tag.
CGI	No	On any page. Values are specific to the latest browser request.	The web server. Contains the server environment variables that result from the browser request.
Client	No	For one client in one application, over multiple browser sessions.	Specifying the prefix Client when you create the variable.
Cookie	No	For one client in one or more applications and pages, over multiple browser sessions.	A cfcookie tag. You can also set memory-only cookies by specifying the prefix Cookie when you create the variable.
Flash	Yes	A ColdFusion page or ColdFusion component called by a Flash client.	The ColdFusion Client access. You assign a value to Flash.You can assign values to the Flash.result and Flash.pagesize variables.
Form	No	On the action page of a form and in custom tags called by the action page; cannot be used on a form page that is not also the action page.	A `form` or cfform tag. Contains the values of form field tags (such as input) in the form body when the form is submitted. The variable name is the name of the form field.
Local	No	Within the body of a user-defined function or ColdFusion component method, only while the function executes.	Either of the following: • In the function or method definition, a `var` keyword in a cfset tag or a CFScript `var` statement. • Specifying the Local keyword when you create a variable in the function or method.

© 2014 Adobe Systems Incorporated. All rights reserved.

Request	Yes	On the creating page and in any pages run during the current HTTP request after the variable is created, including in custom tags and nested custom tags.	Specifying the prefix Request when you create the variable.
Server	Yes	To any page on the ColdFusion server. Surround all code that uses server variables in c `flock` blocks.	Specifying the prefix Server when you create the variable.
Session	Yes	For one client in one application and one browser session. Surround code that uses Session scope variables in cflock blocks.	Specifying the prefix Session when you create the variable.
This	Yes	Within a ColdFusion component or the body of a user-defined function that was created using the cffunction tag and place in an object, structure, or scope. In the containing page, through the component instance or containing object.	Within the component or function by specifying the prefix This when you create the variable.In the containing page, by specifying the component instance or object that contains the function as a prefix when you create the variable.
ThisTag	Yes	On the custom tag page.	Specifying the prefix ThisTag when you create the variable in the tag or using the cfassociate tag in a nested custom tag.
Thread	The thread name.Inside the thread that creates the variable, you can also use the keyword `thread`.	Any code in the request.	Using the keyword `thread` or the thread name as a prefix when you create the variable.You can create Thread variables only inside the thread.
thread-local (no prefix)	none	Within a thread created by the `cfthread` tag	Using no prefix when you create the variable. You can also use the keyword var before the variable name.

© 2014 Adobe Systems Incorporated. All rights reserved.

URL	No	On the target page of the URL.	The system. Contains the parameters passed in the URL query string used to access the page.
Variables(Local)	No	On the current page. Cannot be accessed by a form's action page (unless the form page is also the action page). Variables in this scope used on a page that calls a custom tag can be accessed in the custom tag by using its Caller scope; however, they are not available to any nested custom tags.	Specifying the prefix Variables, or using no prefix, when you create the variable. (To create a Variables scope variable inside a ColdFusion thread, you must use the Variables prefix.)

Using scopes

The following sections provide details on how you can create and use variables in different scopes.

Evaluating unscoped variables

If you use a variable name without a scope prefix, ColdFusion checks the scopes in the following order to find the variable:

1. Local (function-local, UDFs and CFCs only)
2. Arguments
3. Thread local (inside threads only)
4. Query (not a true scope; variables in query loops)
5. Thread
6. Variables
7. CGI
8. Cffile
9. URL
10. Form
11. Cookie
12. Client

Because ColdFusion must search for variables when you do not specify the scope, you can improve performance by specifying the scope for all variables.
To access variables in all other scopes, you must prefix the variable name with the scope identifier.

Scopes and CFX tags

ColdFusion scopes do not apply to ColdFusion Extension (CFX) tags, custom tags that you write in a programming language such as C++ or Java. The ColdFusion page that calls a CFX tag must use tag attributes to pass data to the CFX tag. The CFX tag must use the Java Request and Response interfaces or the C++ Request class to get and return data.
The Java `setVariable` Response interface method and C++ `CCFX::SetVariable` method return data to the Variables scope of the calling page. Therefore, they are equivalent to setting a Caller scope variable in a custom ColdFusion tag.

© 2014 Adobe Systems Incorporated. All rights reserved.

Using scopes as structures

ColdFusion makes all named scopes available as structures. You cannot access the function-local scope for user-defined functions (UDFs) that you define using CFScript as a structure.
You can reference the variables in named scopes as elements of a structure. To do so, specify the scope name as the structure name and the variable name as the key. For example, if you have a MyVar variable in the Request scope, you can reference it in either of the following ways:

```
Request.MyVar
Request["MyVar"]
```

Similarly, you can use CFML structure functions to manipulate the contents of the scope. For more information on using structures, see Using Arrays and Structures.

> Do not call `StructClear(Session)` to clear session variables. This deletes the `SessionID`, `CFID`, and `CFtoken` built-in variables, effectively ending the session. If you want to use `StructClear` to delete your application variables, place those variables in a structure in the Session scope, and then clear that structure. For example, place all your application variables in Session.MyVars and then call `StructClear(Session.MyVars)`_ to clear the variables._

© 2014 Adobe Systems Incorporated. All rights reserved.

Ensuring variable existence

ColdFusion generates an error if you try to use a variable value that does not exist. Therefore, before you use any variable whose value is assigned dynamically, you must ensure that a variable value exists. For example, if your application has a form, it must use some combination of requiring users to submit data in fields, providing default values for fields, and checking for the existence of field variable values before they are used.
There are several ways to ensure that a variable exists before you use it, including the following:

- You can use the `IsDefined` function to test for the variable's existence.
- You can use the `cfparam` tag to test for a variable and set it to a default value if it does not exist.
- You can use a `cfinput` tag with a `hidden` attribute to tell ColdFusion to display a helpful message to any user who does not enter data in a required field. For more information on this technique, see *Requiring users to enter values in form fields* in the Working with action pages.

Testing for a variable's existence

Before relying on a variable's existence in an application page, you can test to see if it exists by using the `IsDefined` function. To check whether a specific entry exists in an array, use the `ArrayIsDefined` function. To check whether a specific key exists in a structure, use the StructKeyExists function.
For example, if you submit a form with an unsettled check box, the action page does not get a variable for the check box. The following example from a form action page makes sure the Contractor check box Form variable exists before using it:

```
<cfif IsDefined("Form.Contractor")>
 <cfoutput>Contractor: #Form.Contractor#</cfoutput>
 </cfif>
```

You must always enclose the argument passed to the `IsDefined` function in quotation marks. For more information on the `IsDefined` function, see the *CFML Reference*.
To test whether an element exists in an array before you display its value, use a format such as the following:

```
</cfoutput>
 <cfloop index="i" from="1" to="#Arraylen(myArray)#">
  <cfif ArrayIsDefined(myArray, i)>
   #i#: #myArray[i]#<br>
  </cfif>
 </cfloop>
</cfoutput>
```

Notice that in the `ArrayIsDefined` function, unlike the `IsDefined` function, you do not surround the variable name in quotation marks.
If you attempt to evaluate a variable that you did not define, ColdFusion cannot process the page and displays an error message. To help diagnose such problems, turn on debugging in the ColdFusion Administrator or use the debugger in your editor. The Administrator debugging information shows which variables are being passed to your application pages.

Variable existence considerations

If a variable is part of a scope that is available as a structure, you might get a minor performance increase by testing the variable's existence using the `StructKeyExists` function instead of the `IsDefined` function.

© 2014 Adobe Systems Incorporated. All rights reserved.

You can also determine which Form variables exist by inspecting the contents of the `Form.fieldnames` built-in variable. This variable contains a list of all the fields submitted by the form. Remember, however, that form text fields are always submitted to the action page, and can contain an empty string if the user did not enter data.

Using the cfparam tag

You can ensure that a variable exists by using the `cfparam` tag, which tests for the variable's existence and optionally supplies a default value if the variable does not exist. The `cfparam` tag has the following syntax:

```
<cfparam name="VariableName"
 type="data_type"
 default="DefaultValue">
```

Note

For information on using the `type` attribute to validate the parameter data type, see the CFML Reference.

There are two ways to use the `cfparam` tag to test for variable existence, depending on how you want the validation test to proceed:

- With only the `name` attribute to test that a required variable exists. If it does not exist, the ColdFusion server stops processing the page and displays an error message.
- With the `name` and `default` attributes to test for the existence of an optional variable. If the variable exists, processing continues and the value is not changed. If the variable does not exist, it is created and set to the value of the `default` attribute, and processing continues.
  The following example shows how to use the `cfparam` tag to check for the existence of an optional variable and to set a default value if the variable does not already exist:

```
<cfparam name="Form.Contract" default="Yes">
```

Example: testing for variables

Using the `cfparam` tag with the `name` attribute is one way to clearly define the variables that a page or a custom tag expects to receive before processing can proceed. This can make your code more readable, as well as easier to maintain and debug.
For example, the following `cfparam` tags indicate that this page expects two form variables named StartRow and RowsToFetch:

```
<cfparam name="Form.StartRow">
 <cfparam name="Form.RowsToFetch">
```

If the page with these tags is called without either one of the form variables, an error occurs and the page stops processing. By default, ColdFusion displays an error message; you can also handle the error as described in Handling Errors.

Example: setting default values

© 2014 Adobe Systems Incorporated. All rights reserved.

The following example uses the `cfparam` tag to see if optional variables exist. If they do exist, processing continues. If they do not exist, the ColdFusion server creates them and sets them to the default values.

```
<cfparam name="Cookie.SearchString" default="temple">
<cfparam name="Client.Color" default="Gray">
<cfparam name="ShowExtraInfo" default="No">
```

You can use the `cfparam` tag to set default values for URL and Form variables, instead of using conditional logic. For example, you could include the following code on the action page to ensure that a SelectedDepts variable exists:

```
<cfparam name="Form.SelectedDepts" default="Marketing,Sales">
```

© 2014 Adobe Systems Incorporated. All rights reserved.

Validating data

It is often not sufficient that input data merely exists; it must also have the right format. For example, a date field must have data in a date format. A salary field must have data in a numeric or currency format. There are many ways to ensure the validity of data, including the following methods:

- Use the `cfparam` tag with the `type` attribute to validate a variable.
- Use the IsValid function to validate a variable.
- Use the `cfqueryparam` tag in a SQL WHERE clause to validate query parameters.
- Use `cfform` controls that have validation attributes.
- Use a form `input` tag with a `hidden` attribute to validate the contents of a form input field.

Note

Data validation using the `cfparam, cfqueryparam`, and `form` tags is done by the server. Validation using `cfform` tags and hidden fields is done using JavaScript in the user's browser, before any data is sent to the server.

For detailed information on validating data in forms and variables, see Validating data For detailed information on validating query parameters, see *Using cfqueryparam* in Enhancing security with cfqueryparam.

© 2014 Adobe Systems Incorporated. All rights reserved.

Passing variables to custom tags and UDFs

The following sections describe rules for how data gets passed to custom tags and user-defined functions that are written in CFML, and to CFX custom tags that are written in Java or C++.

Passing variables to CFML tags and UDFs

When you pass a variable to a CFML custom tag as an attribute, or to a user-defined function as an argument, the following rules determine whether the custom tag or function receives its own private copy of the variable or only gets a reference to the calling page's variable:

- Simple variables and arrays are passed as copies of the data. If your argument is an expression that contains multiple simple variables, the result of the expression evaluation is copied to the function or tag.
- Structures, queries, and `cfobject` objects are passed as references to the object.
  If the tag or function gets a copy of the calling page's data, changes to the variable in the custom tag or function do not change the value of the variable on the calling page. If the variable is passed by reference, changes to the variable in the custom tag or function also change the value of the variable in the calling page.
  To pass a variable to a custom tag, you must enclose the variable name in number signs. To pass a variable to a function, do *not* enclose the variable name in number signs. For example, the following code calls a user-defined function using three Form variables:

```
<cfoutput>
 TOTAL INTEREST: #TotalInterest(Form.Principal, Form.AnnualPercent,Form.Months)#<br>

</cfoutput>
```

The following example calls a custom tag using two variables, MyString and MyArray:

```
<cf_testTag stringval=#MyString# arrayval=#MyArray#>
```

Passing variables to CFX tags

You cannot pass arrays, structures, or `cfobject` objects to CFX tags. You can pass a query to a CFX tag by using the `query` attribute when calling the tag. ColdFusion normally converts simple data types to strings when passing them to CFX tags; however, the Java Request Interface `getIntAttribute` method lets you get a passed integer value.

#back to top

© 2014 Adobe Systems Incorporated. All rights reserved.

Using Expressions and Number Signs

In CFML, you create expressions by using number signs to indicate expressions in Adobe ColdFusion tags such as `cfoutput`, in strings, and in expressions. You also use variables in variable names and strings to create dynamic expressions, and dynamic variables.

© 2014 Adobe Systems Incorporated. All rights reserved.

Expressions-Developing guide

Expressions

ColdFusion expressions consist of *operands* and *operators*. Constants and variables are operands. Operators, such as the multiplication sign, are the verbs that act on the operands; functions are a form of operator.
The simplest expression consists of a single operand with no operators. Complex expressions have multiple operators and operands. The following are all ColdFusion expressions:

```
12
MyVariable
a++
(1 + 1)/2
"father" & "Mother"
Form.divisor/Form.dividend
Round(3.14159)
```

Operators act on the operands. Some operators, such as functions with a single argument, take a single operand. Many operators, including most arithmetic and logical operators, take two operands. The following is the general form of a two-operand expression:

```
Expression Operator Expression
```

Expressions surround the operator. Each expression can be a simple operand (variable or constant) or a *subexpression* consisting of more operators and expressions. Complex expressions are built up using subexpressions. For example, in the expression (1 + 1)/2, 1 + 1 is a subexpression consisting of an operator and two operands.

Operator types

ColdFusion has Five types of operators:

- Arithmetic
- Boolean
- Decision (or comparison)
- String
- Ternary
  Functions also can be viewed as operators because they act on operands.

Arithmetic operators

The following table describes the arithmetic operators:

Operator	Description
+ - * /	Basic arithmetic: Addition, subtraction, multiplication, and division. In division, the right operand cannot be zero.

© 2014 Adobe Systems Incorporated. All rights reserved.

++ –	Increment and decrement. Increase or decrease the variable by one.These operators can be used for pre-incrementing or decrementing (as in {{ x = ++ i}}), where the variable is changed before it is used in the expression. They can also be used for post-incrementing or decrementing (as in `x = i+`), where the value is changed after it is used in the expression. If the value of the variable `i` is initially 7, for example, the value of `x` in `x = ++i` is 8 after expression evaluation, but in `x=i`, the value of `x` is 7. In both cases, the value of `i` becomes 8.These operators cannot be used with expressions that involve functions, as in `f().a`. Also, you can use an expression such as `-x`, but `---x` and `+x` cause errors, because their meanings are ambiguous. You can use parentheses to group the operators, as in `-(--x)` or `(++x)`, however.
+= -= *= /= %=	Compound assignment operators. The variable on the right is used as both an element in the expression and the result variable. Thus, the expression `a += b` is equivalent to `a = a +b`. An expression can have only one compound assignment operator.
+ -	Unary arithmetic: Set the sign of a number.
MODor %	Modulus: Return the remainder after a number is divided by a divisor. The result has the same sign as the divisor. The value to the right of the operator should be an integer; using a non-numeric value causes an error, and if you specify a real number, ColdFusion ignores the fractional part (for example, 11 MOD 4.7 is 3).
	Integer division: Divide an integer by another integer. The result is also an integer; for example, 9\4 is 2. The right operand cannot be zero.
^	Exponentiation: Return the result of a number raised to a power (exponent). Use the caret character (^) to separate the number from the power; for example, 2^3 is 8. Real and negative numbers are allowed for both the base and the exponent. However, any expression that equates to an imaginary number, such -1^.5 results in the string "-1.#IND. ColdFusion does not support imaginary or complex numbers.

Boolean operators

Boolean, or logical, operators perform logical connective and negation operations. The operands of Boolean operators are Boolean (True/False) values. The following table describes the Boolean operators:

© 2014 Adobe Systems Incorporated. All rights reserved.

Operator	Description
NOT or !	Reverse the value of an argument. For example, NOT True is False and the inverse.
AND or &&	Return True if both arguments are True; return False otherwise. For example, True AND True is True, but True AND False is False.
OR or ||	Return True if any of the arguments is True; return False otherwise. For example, True OR False is True, but False OR False is False.
XOR	Exclusive or: Return True if one of the values is True and the other is False. Return False if both arguments are True or both are False. For example, True XOR True is False, but True XOR False is True.
EQV	Equivalence: Return True if both operands are True or both are False. The EQV operator is the opposite of the XOR operator. For example, True EQV True is True, but True EQV False is False.
IMP	Implication: The statement A IMP B is the equivalent of the logical statement "If A Then B." A IMP B is False only if A is True and B is False. It is True in all other cases.

Decision operators

The ColdFusion decision, or comparison, operators produce a Boolean True/False result. Many types of operation have multiple equivalent operator forms. For example, IS and EQ perform the same operation. The following table describes the decision operators:

Operator	Description
ISEQUALEQ	Perform a case-insensitive comparison of two values. Return True if the values are identical.
IS NOTNOT EQUALNEQ	Opposite of IS. Perform a case-insensitive comparison of two values. Return True if the values are not identical.
CONTAINS	Return True if the value on the left contains the value on the right.
DOES NOT CONTAIN	Opposite of CONTAINS. Return True if the value on the left does not contain the value on the right.

© 2014 Adobe Systems Incorporated. All rights reserved.

GREATER THANGT	Return True if the value on the left is greater than the value on the right.
LESS THANLT	Opposite of GREATER THAN. Return True if the value on the left is smaller than the value on the right.
GREATER THAN OR EQUAL TOGTEGE	Return True if the value on the left is greater than or equal to the value on the right.
LESS THAN OR EQUAL TOLTELE	Return True if the value on the left is less than or equal to the value on the right.

Note

In CFScript expressions only, you can also use the following decision operators. You cannot use them in expressions in tags. == (EQ), != (NEQ), > (GT), < (LT), >= (GTE), and <= (LTE).

Decision operator rules

The following rules apply to decision operators:

- When ColdFusion evaluates an expression that contains a decision operator other than CONTAINS or DOES NOT CONTAIN, it first determines if the data can be converted to numeric values. If they can be converted, it performs a numeric comparison on the data. If they cannot be converted, it performs a string comparison. This can sometimes result in unexpected results. For more information on this behavior, see *Evaluation and type conversion issues* in Data type conversion.
- When ColdFusion evaluates an expression with CONTAINS or DOES NOT CONTAIN it does a string comparison. The expression A CONTAINS B evaluates to True if B is a substring of A. Therefore an expression such as the following evaluates as True:

```
123.45 CONTAINS 3.4
```

- When a ColdFusion decision operator compares strings, it ignores the case. As a result, the following expression is True:

```
"a" IS "A"
```

- When a ColdFusion decision operator compares strings, it evaluates the strings from left to right, comparing the characters in each position according to their sorting order. The first position where the characters differ determines the relative values of the strings. As a result, the following expressions are True:

```
"ab" LT "aba"
"abde" LT "ac"
```

String operators

© 2014 Adobe Systems Incorporated. All rights reserved.

String operators manipulate strings of characters. The following table describes the operators:

Operator	Description
&	Concatenates strings.
&=	Compound concatenation. The variable on the right is used as both an element in the concatenation operation and the result variable. Thus, the expression `a &= b` is equivalent to `a = a & b`.An expression can have only one compound assignment operator.

Note

In a Query of Queries, you use || as the concatenation operator.

Ternary Operator

The ternary operator is a decision operator with three operands. It assigns a variable a value based on a Boolean expression. The operator has the form

```
(Boolean expression)? expression1 : expresson2
```

If the Boolean expression evaluates to `true`, the operator result is `expression1`; otherwise, it is `expression2` For example

```
<cfset c = (a GT b)? a : b >
```

If a is greater than b, c is assigned the value of a; otherwise, c is assigned the value of b.
The parentheses can contain any expression that evaluates to a Boolean value, and a and b can be any valid expression. You can nest this operator inside other expressions.

Operator precedence and evaluation ordering

The order of precedence controls the order in which operators in an expression are evaluated. The order of precedence is as follows. (Some alternative names for operators, such as EQUALS and GREATER THAN OR EQUAL TO are omitted for brevity.)

© 2014 Adobe Systems Incorporated. All rights reserved.

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

To enforce a nonstandard order of evaluation, parenthesize expressions. For example:

- 6 - 3 * 2 is equal to 0
- (6 - 3) * 2 is equal to 6
  You can nest parenthesized expressions. When in doubt about the order in which operators in an expression are evaluated, use parentheses to force the order of evaluation.

Using functions as operators

Functions are a form of operator. Because ColdFusion functions return values, you can use function results as operands. Function arguments are expressions. For example, the following are valid expressions:

- `Rand()`
- `UCase("This is a text: ") & ToString(123 + 456)`

Function syntax

The following table shows function syntax and usage guidelines:

Usage	Example
No arguments	`Function()`
Basic format	`Function(Data)`
Nested functions	`Function1(Function2(Data))`
Multiple arguments	`Function(Data1, Data2, Data3)`
String arguments	`Function('This is a demo')Function(This is a demo)`
Arguments that are expressions	`Function1(X*Y, Function2("Text"))`

All functions return values. In the following example, the `cfset` tag sets a variable to the value returned by the `Now` function:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cfset myDate = DateFormat(Now(), "mmmm d, yyyy")>
```

You can use the values returned by functions directly to create more complex expressions, as in the following example:

```
Abs(Myvar)/Round(3.14159)
```

For more information on how to insert functions in expressions, see Using number signs.

Optional function arguments

Some functions take optional arguments after their required arguments. If omitted, all optional arguments default to a predefined value. For example:

- `Replace("Eat and Eat", "Eat", "Drink")` returns "Drink and Eat"
- `Replace("Eat and Eat", "Eat", "Drink", "All")` returns "Drink and Drink"
  The difference in the results is because the `Replace` function takes an optional fourth argument that specifies the scope of replacement. The default value is "One," which explains why only the first occurrence of "Eat" was replaced with "Drink" in the first example. In the second example, a fourth argument causes the function to replace all occurrences of "Eat" with "Drink".

Expression evaluation and functions

It is important to remember that ColdFusion evaluates function attributes as expressions before it executes the function. As a result, you can use any ColdFusion expression as a function attribute. For example, consider the following lines:

```
<cfset firstVariable = "we all need">
<cfset myStringVar = UCase(firstVariable & " more sleep!")>
```

When ColdFusion server executes the second line, it does the following:

1. Identifies an expression with a string concatenation.
2. Evaluates the firstVariable variable as the string "we all need".
3. Concatenates "we all need" with the string "more sleep!" to get "we all need more sleep!".
4. Passes the string "we all need more sleep!" to the `UCase` function.
5. Executes the `UCase` function on the string argument "we all need more sleep!" to get "WE ALL NEED MORE SLEEP!".
6. Assigns the string value "WE ALL NEED MORE SLEEP!" to the variable myStringVar.
   ColdFusion completes steps 1-3 before running the function.

Using multiple assignments in one expression

You can chain assignments to assign the same value to multiple variables in a single statement. This includes chain assignments for the results of an expression. The following code displays a chain assignment:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
a=b=c=d*5
```

You can use the `var` operator in multiple assignments, but the variables with this operator must precede all others. For example:

```
//The following line is valid.
var a = var b = c = d*5
//The following line is not valid.
// a = b = var c = d*5
```

© 2014 Adobe Systems Incorporated. All rights reserved.

Using number signs

Number signs (#) have a special meaning in CFML. When the ColdFusion server encounters number signs in CFML text, such as the text in a `cfoutput` tag body, it checks to see if the text between the number signs is either a variable or a function.

> ✅ Number signs are also called pound signs.

Is so, it replaces the text and surrounding number signs with the variable value or the result of the function. Otherwise, ColdFusion generates an error.
For example, to output the current value of a variable named `Form.MyFormVariable`, you delimit (surround) the variable name with number signs:

```
<cfoutput>Value is #Form.MyFormVariable#</cfoutput>
```

In this example, the variable `Form.MyFormVariable` is replaced with the value assigned to it.
Follow these guidelines when using number signs:

- Use number signs to distinguish variables or functions from plain text.
- Surround only a single variable or function in number signs; for example, #Variables.myVar# or #Left(myString, position)#. (However, a function in number signs can contain nested functions, such as #Left(trim(myString), position)#.
- Do not place complex expressions, such as{{ 1 + 2 }}in number signs. Although this is allowed in a cfoutput block, such as <cfoutput>One plus one is #1 + 1#</cfoutput>, doing so mixes logic and presentation.
- Use number signs *only* where necessary, because unneeded number signs slow processing.
  For a description of using number signs to create variable names, see *Using number signs to construct a variable name in assignments* in Dynamic expressions and dynamic variables.

Using number signs in ColdFusion tag attribute values

You can place variables, functions, or expressions inside tag attributes by enclosing the variable or expression with number signs. For example, if the variable CookieValue has the value "MyCookie", the following line sets the `cfcookie` `value` attribute to "The value is MyCookie":

```
<cfcookie name="TestCookie" value="The value is #CookieValue#">
```

You can optionally omit quotation marks around variables used as attribute values as shown in the following example:

```
<cfcookie name = TestCookie value = #CookieValue#>
```

However, surrounding all attribute values in quotation marks is more consistent with HTML coding style.
If you use string expressions to construct an attribute value, as shown in the following example, the strings inside the expression use single quotation marks (') to differentiate the quotation marks from the quotation marks that surround the attribute value.

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cfcookie name="TestCookie2" value="The #CookieValue & 'ate the cookie!'#">
```

Note

You do not need to use number signs when you use the cfset tag to assign one variable's value to another value. For example, the following tag assigns the value of the `oldVar` variable to the new variable, `newVar`: `<cfset newVar = oldVar>`.

Using number signs in tag bodies

You can place variables or functions freely inside the bodies of the following tags by enclosing each variable or expression with number signs:

- `cfoutput`
- `cfquery`
- `cfmail`

For example:

```
<cfoutput>
 Value is #Form.MyTextField#
</cfoutput>

<cfoutput>
 The name is #FirstName# #LastName#.
</cfoutput>

<cfoutput>
 The value of Cos(0) is #Cos(0)#
</cfoutput>
```

If you omit the number signs, the text, rather than the value, appears in the output generated by the `cfoutput` state ment.
Two expressions inside number signs can be adjacent to one another, as in the following example:

```
<cfoutput>
 "Mo" and "nk" is #Left("Moon", 2)##Mid("Monkey", 3, 2)#
</cfoutput>
```

This code displays the following text:
"Mo" and "nk" is Monk
ColdFusion does not interpret the double number sign as an escaped # character.

Using number signs in strings

You can place variables or functions freely inside strings by enclosing each variable or expression with number signs; for example:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cfset TheString = "Value is #Form.MyTextField#">
<cfset TheString = "The name is #FirstName# #LastName#.">
<cfset TheString = "Cos(0) is #Cos(0)#">
```

ColdFusion automatically replaces the text with the value of the variable or the value returned by the function. For example, the following pairs of `cfset` statements produce the same result:

```
<cfset TheString = "Hello, #FirstName#!">
<cfset TheString = "Hello, " & FirstName & "!">
```

If number signs are omitted inside the string, the text, rather than the value, appears in the string. For example, the following pairs of `cfset` statements produce the same result:

```
<cfset TheString = "Hello, FirstName!">
<cfset TheString = "Hello, " & "First" & "Name!">
```

As with the `cfoutput` statement, two expressions can be adjacent to each other in strings, as in the following example:

```
<cfset TheString = "Monk is #Left("Moon", 2)##Mid("Monkey", 3, 2)#">
```

The double-quotation marks around "Moon" and "Monkey" do *not* need to be escaped (as in ""Moon"" and ""Monkey""). This is because the text between the number signs is treated as an expression; it is evaluated before its value is inserted inside the string.

Nested number signs

In a few cases, you can nest number signs in an expression. The following example uses nested number signs:

```
<cfset Sentence = "The length of the full name is #Len("#FirstName# #LastName#")#">
```

In this example, number signs are nested so that the values of the variables FirstName and LastName are inserted in the string whose length the `Len` function calculates.
Nested number signs imply a complex expression that can typically be written more clearly and efficiently without the nesting. For example, you can rewrite the preceding code example without the nested number signs, as follows:

```
<cfset Sentence2 = "The length of the full name is #Len(FirstName & " " &
LastName)#">
```

© 2014 Adobe Systems Incorporated. All rights reserved.

The following achieves the same results and can further improve readability:

```
<cfset FullName = "#FirstName# #LastName#">
<cfset Sentence = "The length of the full name is #Len(FullName)#">
```

A common mistake is to place number signs around the arguments of functions, as in:

```
<cfset ResultText = "#Len(#TheText#)#">
<cfset ResultText = "#Min(#ThisVariable#, 5 + #ThatVariable#)#">
<cfset ResultText = "#Len(#Left("Some text", 4)#)#">
```

These statements result in errors. As a general rule, *never* place number signs around function arguments.

Using number signs in expressions

Use number signs in expressions only when necessary, because unneeded number signs reduce clarity and can increase processing time. The following example shows the preferred method for referencing variables:

```
<cfset SomeVar = Var1 + Max(Var2, 10 * Var3) + Var4>
```

In contrast, the following example uses number signs unnecessarily and is less efficient than the previous statement:

```
<cfset #SomeVar# = #Var1# + #Max(Var2, 10 * Var3)# + #Var4#>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

Dynamic expressions and dynamic variables

Many ColdFusion programmers never encounter or require dynamic expressions. However, dynamic variable naming is important in situations where the variable names are not known in advance, such as in shopping cart applications.
ColdFusion also includes an `IIf` function, which is most often used without dynamic expressions. This function dynamically evaluates its arguments, and you often must use the `DE` function to prevent the evaluation. For more information on using the `IIF` function, see *Using the IIF function* in this page.

About dynamic variables

Dynamic variables are variables that are named dynamically, typically by creating a variable name from a static part and a variable part. For example, the following example dynamically constructs the variable name from a variable prefix and a static suffix:

```
<cfset "#flavor#_availability" = "out of stock">
```

Using dynamic variables in this manner does not require dynamic evaluation.

About dynamic expressions and dynamic evaluation

In a *dynamic expression*, the actual expression, not just its variable values, is determined at execution time. In other words, in a dynamic expression the structure of the expression, such as the names of the variables, not just the values of the variables, gets built at runtime.
You create dynamic expressions using *string expressions*, which are expressions contained in strings, (that is, surrounded with quotation marks). *Dynamic evaluation* is the process of evaluating a string expression. The `Evaluate` and `IIf` functions, and only these functions, perform dynamic evaluation.
When ColdFusion performs dynamic evaluation it does the following:

1. Takes a string expression and treats it as a standard expression, as if the expression was not a string.
2. Parses the expression to determine the elements of the expression and validate the expression syntax.
3. Evaluates the expression, looking up any variables and replacing them with their values, calling any functions, and performing any required operations.

This process enables ColdFusion to interpret dynamic expressions with variable parts. However, it incurs a substantial processing overhead.
Dynamic expressions were important in early versions of ColdFusion, before it supported arrays and structures, and they still can be useful in limited circumstances. However, the ability to use structures and the ability to use associative array notation to access structure elements provide more efficient and easier methods for dynamically managing data. For information on using arrays and structures, see Using Arrays and Structures.

Selecting how to create variable names

The following two examples describe cases when you need dynamic variable names:

- Form applications where the number and names of fields on the form vary dynamically. In this case, the form posts only the names and values of its fields to the action page. The action page does not know all the names of the fields, although it does know how the field names (that is, the variable names) are constructed.
- If the following are true:
  - ColdFusion calls a custom tag multiple times.
  - The custom tag result must be returned to different variables each time.
  - The calling code can specify the variable in which to return the custom tag result.
    In this case, the custom tag does not know the return variable name in advance, and gets it as an

© 2014 Adobe Systems Incorporated. All rights reserved.

attribute value.
In both cases, it might appear that dynamic expressions using the *Evaluate* function are needed to construct the variable names. However, you can achieve the same ends more efficiently by using dynamic variable naming, as shown in [Example: a dynamic shopping cart].
This does not mean that you must always avoid dynamic evaluation. However, given the substantial performance costs of dynamic evaluation, first ensure that one of the following techniques cannot serve your purpose:

- An array (using index variables)
- Associative array references containing expressions to access structure elements
- Dynamically generated variable names

Dynamic variable naming without dynamic evaluation

While ColdFusion does not always allow you to construct a variable name inline from variable pieces, but it does allow the most common uses.

Using number signs to construct a variable name in assignments

You can combine text and variable names to construct a variable name on the left side of a `cfset` assignment. For example, the following code sets the value of the variable Product12 to the string "Widget":

```
<cfset ProdNo = 12>
<cfset "Product#ProdNo#" = "Widget">
```

To construct a variable name this way, all the text on the left side of the equal sign must be in quotation marks. This usage is less efficient than using arrays. The following example has the same purpose as the previous one, but requires less processing:

```
<cfset MyArray=ArrayNew(1)>
<cfset prodNo = 12>
<cfset myArray[prodNo] = "Widget">
```

Dynamic variable limitation

When you use a dynamic variable name in quotation marks on the left side of an assignment, the name must be either a simple variable name or a complex name that uses object.property notation (such as MyStruct.#KeyName#). You cannot use an array as part of a dynamic variable name. For example, the following code generates an error:

```
<cfset MyArray=ArrayNew(1)>
<cfset productClassNo = 1>
<cfset productItemNo = 9>
<cfset "myArray[#productClassNo##productItemNo#]" = "Widget">
```

However, you can construct an array index value dynamically from variables without using quotation marks on the left side of an assignment. For example, the preceding sample code works if you replace the final line with the following line:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cfset myArray[#productClassNo# & #productItemNo#] = "Widget">
```

Dynamically constructing structure references

The ability to use associative array notation to reference structures provides a way for you to use variables to dynamically create structure references. (For a description of associative array notation, see *Structure notation*.) Associative array structure notation allows you to use a ColdFusion expression inside the index brackets. For example, if you have a productName structure with keys of the form product_1, product_2 and so on, you can use the following code to display the value of productName.product_3:

```
<cfset prodNo = 3>
<cfoutput>
 Product_3 Name: #ProductName["product_" & prodNo]#
<cfoutput>
```

For an example of using this format to manage a shopping cart, see [Example: a dynamic shopping cart].

Using dynamic evaluation

Dynamic evaluation and dynamic expressions have several features and consideratons.

ColdFusion dynamic evaluation functions

The following table describes the functions that perform dynamic evaluation and are useful in evaluating dynamic expressions:

Function	Purpose
DE	Escapes any double-quotation marks in the argument and wraps the result in double-quotation marks. The `DE` function is particularly useful with the `IIF` function, to prevent the function from evaluating a string to be output. For an example of using the `DE` function with the `IIF` function, see Using the IIF function.
Evaluate	Takes one or more string expressions and dynamically evaluates their contents as expressions from left to right. (The results of an evaluation to the left can have meaning in an expression to the right.) Returns the result of evaluating the rightmost argument. For more information on this function see About the Evaluate function.

© 2014 Adobe Systems Incorporated. All rights reserved.

IIf	Evaluates a Boolean condition expression. Depending on whether this expression is True or False, dynamically evaluates one of two string expressions and returns the result of the evaluation. The `IIF` functi on is convenient for incorporating a `cfif` tag in line in HTML. For an example of using this function, see Using the IIF function.
PrecisionEvaluate	Operates identically to the `Evaluate` function, except that it can calculate arbitrary precision decimal arithmetic. If one or more operands in an arithmetic expression are decimal numbers, such as 12947834.986532, and are too long to be represented exactly by a ColdFusion numeric data type, the function uses arbitrary-precision arithmetic to calculate the result, and return the result as an arbitrarily long string of numbers. For more information about this function, see PrecisionEvaluate in the *CFML Reference*.
SetVariable	Sets a variable identified by the first argument to the value specified by the second argument. This function is no longer required in well-formed ColdFusion pages; see SetVariable function considerations.

Function argument evaluation considerations

It is important to remember that ColdFusion always evaluates function arguments *before* the argument values are passed to a function:
For example, consider the following `DE` function:

```
<cfoutput>#DE("1" & "2")#</cfoutput>
```

You might expect this line to display """1"" & ""2""". Instead, it displays "12", because ColdFusion processes the line as follows:

1. Evaluates the expression "1" & "2" as the string "12".
2. Passes the string "12" (without the quotation marks) to the `DE` function.
3. Calls the `DE` function, which adds literal quotation marks around the 12.
   Similarly, if you use the expression DE(1 + 2), ColdFusion evaluates 1 + 2 as the integer 3 and passes it to the function. The function converts it to a string and surrounds the string in literal quotation marks: "3".

About the Evaluate function

The `Evaluate` function takes one or more string expressions, dynamically evaluates their contents as expressions from left to right, and returns the result of evaluating the rightmost argument.
The following example shows the `Evaluate` function and how it works with ColdFusion variable processing:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cfset myVar2="myVar">
<cfset myVar="27/9">
<cfoutput>
 #myVar2#<br>
 #myVar#<br>
 #Evaluate("myVar2")#<br>
 #Evaluate("myVar")#<br>
 #Evaluate(myVar2)#<br>
 #Evaluate(myVar)#<br>
</cfoutput>
```

Reviewing the code

The following table describes how ColdFusion processes this code:

Code	Description
`<cfset myVar2="myVar">` `<cfset myVar="27/9">`	Sets the two variables to the following strings:myVar27/9
`<cfoutput>#myVar2# #myVar# `	Displays the values assigned to the variables, myVar and 27/9, respectively.
`#Evaluate("myVar2")# `	Passes the string "myvar2" (without the quotation marks) to the Evaluate function, which does the following:1-Evaluates it as the variable myVar2. 2-Returns the value of the myVar2 variable, the string "myvar" (without the quotation marks).
`#Evaluate("myVar")# `	Passes the string "myvar" (without the quotation marks) to the Evaluate function, which does the following:1-Evaluates it as the variable myVar. 2-Returns the value of the myVar variable, the string "27/9" (without the quotation marks).
`#Evaluate(myVar2)# `	Evaluates the variable myVar2 as the string "myVar" and passes the string (without the quotation marks) to the Evaluate function. The rest of the processing is the same as in the previous line.
`#Evaluate(myVar)# </cfoutput>`	Evaluates the variable myVar as the string "27/9" (without the quotation marks), and passes it to the Evaluate function, which does the following:1-Evaluates the string as the expression 27/9.2-Performs the division.3-Returns the resulting value, 3

As you can see, using dynamic expressions can result in substantial expression evaluation overhead, and the code can be confusing. Therefore, you should avoid using dynamic expressions wherever a simpler technique, such as using indexed arrays or structures can serve your purposes.

© 2014 Adobe Systems Incorporated. All rights reserved.

Avoiding the Evaluate function

Using the *Evaluate* function increases processing overhead, and in most cases it is not necessary. These examples show some cases where you can consider using the Evaluate function:

Example 1

You might be inclined to use the Evaluate function in code such as the following:

```
<cfoutput>1 + 1 is #Evaluate(1 + 1)#</cfoutput>
```

Although this code works, it is not as efficient as the following code:

```
<cfset Result = 1 + 1>
<cfoutput>1 + 1 is #Result#</cfoutput>
```

Example 2

This example shows how you can use an associative array reference in place of an Evaluate function. This technique is powerful because:

- Most ColdFusion scopes are accessible as structures.
- You can use ColdFusion expressions in the indexes of associative array structure references. For more information on using associative array references for structures, see Structure notation.
  The following example uses the Evaluatefunction to construct a variable name:

  ```
  <cfoutput>
   Product Name: #Evaluate("Form.product_#i#")#
  </cfoutput>
  ```

  This code comes from an example where a form has entries for an indeterminate number of items in a shopping cart. A product name field exists for each item in the shopping cart. The field name is of the form product_1, product_2, and so on, where the number corresponds to the product entry in the shopping cart. In this example, ColdFusion does the following:

1. Replaces the variable i with its value, for example 1.
2. concatenates the variable value with "Form.product_", and passes the result (for Form.product_1) to the Eva luate function, which does the remaining steps.
3. Parses the variable product_1 and generates an executable representation of the variable. Because ColdFusion must run its parser, this step requires substantial processing, even for a simple variable.
4. Evaluates the representation of the variable, for example as "Air popper".
5. Returns the value of the variable.
   The following example has the same result as the preceding example and is more efficient:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cfoutput>
 ProductName: #Form["product_" & i]#
</cfoutput>
```

In this code, ColdFusion does the following:

6. Evaluates the expression in the associative array index brackets as the string "product_" concatenated with the value of the variable i.
7. Determines the value of the variable i; 1.
8. Concatenates the string and the variable value to get product_1.
9. Uses the result as the key value in the Form structure to get Formproduct_1. This associative array reference accesses the same value as the object.attribute format reference Form.product_1; in this case, Air popper. This code format does not use any dynamic evaluation, but it achieves the same effect, of dynamically creating a structure reference by using a string and a variable.

SetVariable function considerations

You can avoid using the `SetVariable` function by using a format such as the following to set a dynamically named variable. For example, the following lines are equivalent:

```
<cfset SetVariable("myVar" & i, myVal)>
<cfset "myVar#i#" = myVal>
```

In the second line, enclosing the myVar#i# variable name in quotation marks tells ColdFusion to evaluate the name and process any text in number signs as a variable or function. ColdFusion replaces the #i# with the value of the variable i, so that if the value of i is 12, this code is equivalent to the line

```
<cfset myVar12 = myVal>
```

For more information on this usage, see *Using number signs to construct a variable name in assignments* in this page.

Using the IIF function

The `IIf` function is a shorthand for the following code:

```
<cfif argument1>
 <cfset result = Evaluate(argument1)>
<cfelse>
 <cfset result = Evaluate(argument2)>
</cfif>
```

The function returns the value of the result variable. It is comparable to the use of the JavaScript and Java ? : operator, and can result in more compact code. As a result, the `IIF` function can be convenient even if you are not using dynamic expressions.

© 2014 Adobe Systems Incorporated. All rights reserved.

The IIF function requires the DE function to prevent ColdFusion from evaluating literal strings, as the following example shows:

```
<cfoutput>
 #IIf(IsDefined("LocalVar"), "LocalVar", DE("The variable is not defined."))#
</cfoutput>
```

If you do not enclose the string "The variable is not defined." in a DE function, the IIF function tries to evaluate the contents of the string as an expression and generates an error (in this case, an invalid parser construct error).
The IIF function is useful for incorporating ColdFusion logic in line in HTML code, but it entails a processing time penalty in cases where you do not otherwise need dynamic expression evaluation.
The following example shows using IIF to alternate table row background color between white and gray. It also shows the use of the DE function to prevent ColdFusion from evaluating the color strings.

```
<cfoutput>
 <table border="1" cellpadding="3">
  <cfloop index="i" from="1" to="10">
   <tr bgcolor="#IIF( i mod 2 eq 0, DE("white"), DE("gray") )#">
   <td>
    hello #i#
   </td>
   </tr>
  </cfloop>
 </table>
</cfoutput>
```

This code is more compact than the following example, which does not use IIF or DE:

```
<cfoutput>
<table border="1" cellpadding="3">
 <cfloop index="i" from="1" to="10">
  <cfif i mod 2 EQ 0>
   <cfset Color = "white">
  <cfelse>
   <cfset Color = "gray">
  </cfif>
  <tr bgcolor="#color#">
   <td>
    hello #i#
   </td>
  </tr>
 </cfloop>
</table>
</cfoutput>
```

Example: a dynamic shopping cart

The following example dynamically creates and manipulates variable names without using dynamic expression evaluation by using associative array notation.

© 2014 Adobe Systems Incorporated. All rights reserved.

You need to dynamically generate variable names in applications such as shopping carts, where the required output is dynamically generated and variable. In a shopping cart, you do not know in advance the number of cart entries or their contents. Also, because you are using a form, the action page only receives Form variables with the names and values of the form fields.

The following example shows the shopping cart contents and lets you edit your order and submit it. To simplify things, the example automatically generates the shopping cart contents using CFScript instead of having the user fill the cart. A more complete example would populate a shopping cart as the user selected items. Similarly, the example omits all business logic for committing and making the order.

Create the form

1. Create a file in your editor.

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<html>
<head>
 <title>Shopping Cart</title>
</head>
<cfscript>
CartItems=4;
Cart = ArrayNew(1);
for ( i=1; i LE cartItems; i=i+1)
{
 Cart[i]=StructNew();
 Cart[i].ID=i;
 Cart[i].Name="Product " & i;
 Cart[i].SKU=i*100+(2*i*10)+(3*i);
 Cart[i].Qty=3*i-2;
}
</cfscript>

<body>
Your shopping cart has the following items.<br>
You can change your order quantities.<br>
If you don't want any item, clear the item's check box.<br>
When you are ready to order, click submit.<br>
<br>
<cfform name="ShoppingCart" action="ShoppingCartAction.cfm" method="post">
<table>
 <tr>
 <td>Order?</td>
 <td>Product</td>
 <td>Code</td>
 <td>Quantity</td>
 </tr>
 <cfloop index="i" from="1" to="#cartItems#">
  <tr>
  <cfset productName= "product_" & Cart[i].ID>
  <cfset skuName= "sku_" & Cart[i].ID>
  <cfset qtyname= "qty_" & Cart[i].ID>
  <td><cfinput type="checkbox" name="itemID" value="#Cart[i].ID#" checked>
    </td>
  <td><cfinput type="text" name="#productName#" value="#Cart[i].Name#"
    passThrough = "readonly = 'True'"></td>
  <td><cfinput type="text" name="#skuName#" value="#Cart[i].SKU#"
    passThrough = "readonly = 'True'"></td>
  <td><cfinput type="text" name="#qtyName#" value="#Cart[i].Qty#">
    </td>
  </tr>
 </cfloop>
</table>
<input type="submit" name="submit" value="submit">
</cfform>

</body>
</html>
```

2. Save the file as ShoppingCartForm.cfm.

© 2014 Adobe Systems Incorporated. All rights reserved.

Reviewing the code

The following table describes the code:

Code	Description

<cfscript>
CartItems=4;
Cart = ArrayNew(1);
for (i=1; i LE cartItems; i=i+1)
{
Cart[i]=StructNew();
Cart[i].ID=i;
Cart[i].Name="Product " & i;
Cart[i].SKU=i*100+(2*i*10)+(3*i);
 	
Cart[i].Qty=3*i-2;
}
</cfscript>
	Create a shopping cart as an array of structures, with each structure containing the cart item ID, product name, SKU number, and quantity ordered for one item in the cart. Populate the shopping cart by looping CartItems times and setting the structure variables to arbitrary values based on the loop counter. A real application would set the Name, SKU, and Quantity values on other pages.

<cfform name="ShoppingCart" action="ShoppingCartAction.cfm" method="post"><table> <tr> <td>Order?</td> <td>Product</td> <td>Code</td> <td>Quantity</td> </tr>	Start the form and its embedded table. When the user clicks the submit button, post the form data to the ShoppingCartAction.cfm page. The table formats the form neatly. The first table row contains the column headers. Each following row has the data for one cart item.

© 2014 Adobe Systems Incorporated. All rights reserved.

`<cfloop index="i" from="1" to="#cartItems#">`
`  <tr>`
`  <cfset productName= "product_" & Cart[i].ID>`
`  <cfset skuName= "sku_" & Cart[i].ID>`
`  <cfset qtyname= "qty_" & Cart[i].ID>`
`  <td><cfinput type="checkbox" name="itemID" value="#Cart[i].ID#" checked>`
`  </td>`
`  <td><cfinput type="text" name="#productName#" value="#Cart[i].Name#"`
`  passThrough = "readonly = 'True'"></td>`
`  <td><cfinput type="text" name="#skuName#" value="#Cart[i].SKU#"`
`  passThrough = "readonly = 'True'"></td>`
`  <td><cfinput type="text" name="#qtyName#" value="#Cart[i].Qty#">`
`  </td>`
`  </tr>`
`  </cfloop>`
`</table>`
	Loop through the shopping cart entries to generate the cart form dynamically. For each loop, generate variables used for the form field name attributes by appending the cart item ID (Cart[i].ID) to a field type identifier, such as "sku_". Use a single name, "itemID", for all check boxes. This way, the itemID value posted to the action page is a list of all the check box field values. The check box field value for each item is the cart item ID.Each column in a row contains a field for a cart item structure entry. The `passthrough` attribute sets the product name and SKU fields to read only; note the use of single-quotation marks. (For more information on the cfinput tag `passthrough` attribute, see the *CFML Reference*.) The check boxes are selected by default.

`<input type="submit" name="submit" value="submit">` `</cfform>`	Create the Submit button and end the form.

Create the Action page

1. Create a file in your editor.
2. Enter the following text:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<html>
<head>
 <title>Your Order</title>
</head>
<body>
<cfif isDefined("Form.submit")>
 <cfparam name="Form.itemID" default="">
 <cfoutput>
  You have ordered the following items:<br>
  <br>
  <cfloop index="i" list="#Form.itemID#">
   ProductName: #Form["product_" & i]#<br>
   Product Code: #Form["sku_" & i]#<br>
   Quantity: #Form["qty_" & i]#<br>
   <br>
  </cfloop>
 </cfoutput>
</cfif>
</body>
</html>
```

3. Save the file as ShoppingCartAction.cfm
4. Open ShoppingCartform.cfm in your browser, change the check box and quantity values, and click Submit.

Reviewing the code

The following table describes the code:

Code	Description
`<cfif isDefined("Form.submit")>`	Run the CFML on this page only if it is called by submitting a form. This is not needed if the form and action pages are separate, but is required if the form and action page were one ColdFusion page.
`<cfparam name="Form.itemID" default="">`	Set the default Form.itemID to the empty string. This prevents ColdFusion from displaying an error if the user clears all check boxes before submitting the form (so no product IDs are submitted).

© 2014 Adobe Systems Incorporated. All rights reserved.

`<cfoutput>` `You have ordered the following items: ` ` `<cfloop index="i" list="#Form.itemID#">` `ProductName: #Form["product_" & i]# ` `Product Code: #Form["sku_" & i]# ` `Quantity: #Form["qty_" & i]# ` ` ` `</cfloop>` `</cfoutput>` `</cfif>`	Display the name, SKU number, and quantity for each ordered item.The form page posts Form.itemID as a list containing the value attributes of all the check boxes. These attributes contain the shopping cart item IDs for the selected cart items. Use the list values to index a loop that outputs each ordered item.Use associative array notation to access the Form scope as a structure and use expressions in the array indexes to construct the form variable names. The expressions consist of a string containing the field name's field type prefix (for example, "sku_"), concatenated with the variable i, which contains the shopping cart ItemID number (which is also the loop index variable).

#back to top

© 2014 Adobe Systems Incorporated. All rights reserved.

Using Arrays and Structures

Adobe ColdFusion supports dynamic multidimensional arrays. Using arrays can enhance your ColdFusion application code.

Adobe ColdFusion also supports structures for managing lists of key-value pairs. Because structures can contain other structures or complex data types as it values, they provide a flexible and powerful tool for managing complex data.

© 2014 Adobe Systems Incorporated. All rights reserved.

About arrays

Traditionally, an *array* is a tabular structure used to hold data, much like a spreadsheet table with clearly defined limits and dimensions.
In ColdFusion, you typically use arrays to temporarily store data. For example, if your site lets users order goods online, you can store their shopping cart contents in an array. Using an array lets you make changes easily without committing the information, which the user can change before completing the transaction, to a database.

Basic array concepts

Subsequent discussions of ColdFusion arrays are based on the following terms:

- **Array dimension** The relative complexity of the array structure.
- **Index** The position of an element in a dimension, ordinarily surrounded by brackets: my1Darray1, my2Darray11, my3Darray111.
- **Array element:** Data stored at an array index. The simplest array is a one-dimensional array, like a row in a table. A one-dimensional array has a *name* (the variable name) and a numeric index. The index number references a single entry, or cell, in the array.
  Thus, the following statement sets the value of the fifth entry in the one-dimensional array MyArray to "Robert":

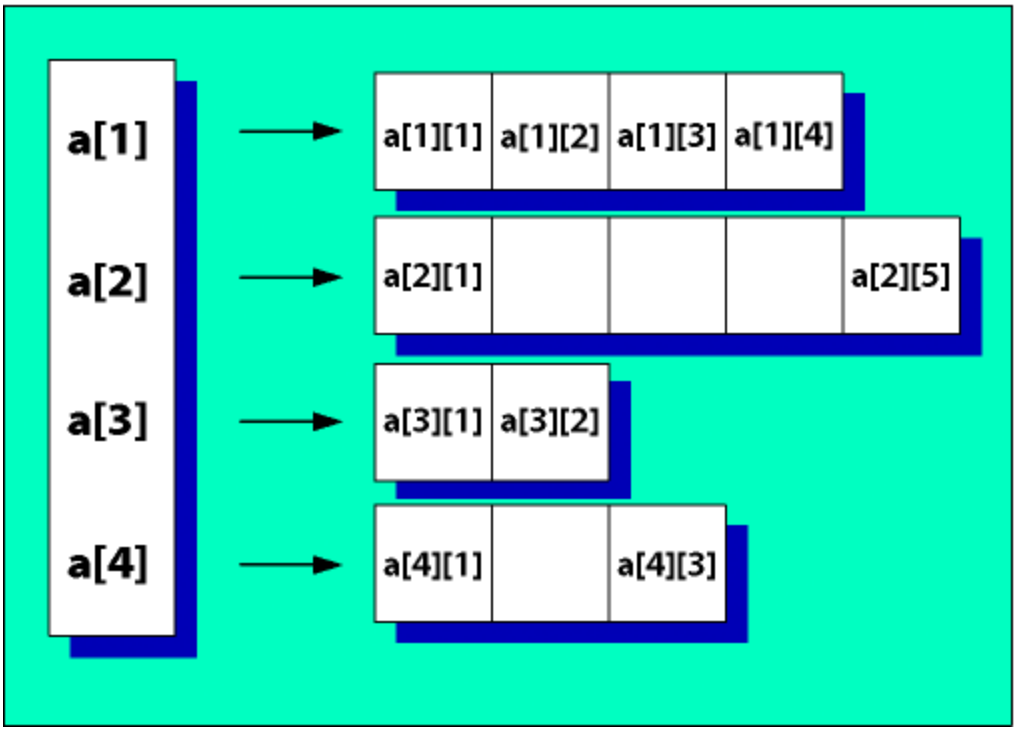
```
<cfset MyArray[5] = "Robert">
```

A basic two-dimensional (2D) array is like a simple table. A three-dimensional (3D) array is like a cube of data, and so on. ColdFusion lets you directly create arrays with up to three dimensions. You can use multiple statements to create arrays with more than three dimensions.
The syntax `my2darray13="Paul"` is the same as saying "My2dArray is a two-dimensional array and the value of the array element index 13 is Paul".

About ColdFusion arrays

ColdFusion arrays differ from traditional arrays, because they are dynamic. For example, in a conventional array, array size is constant and symmetrical, whereas in a ColdFusion array, you can have rows of differing lengths based on the data that is added or removed.
A conventional 2D array is like a fixed-size table made up of individual cells.
A ColdFusion 2D array is actually a one-dimensional array that contains a series of additional 1D arrays. Each of the arrays that make up a row can expand and contract independently of any other column.
The following figure represents a ColdFusion 2D array:

© 2014 Adobe Systems Incorporated. All rights reserved.



A ColdFusion 3D array is essentially three nested sets of 1D arrays. The differences between traditional and ColdFusion 3D arrays are similar, but much harder to show on a page.

Dynamic arrays expand to accept data that you add to them and contract as you remove data from them.

© 2014 Adobe Systems Incorporated. All rights reserved.

Basic array techniques

Referencing array elements

You reference array elements by enclosing the index with brackets: *arrayName*_x_ where *x* is the index that you want to reference. In ColdFusion, array indexes are counted starting with position 1, which means that position 1 in the firstname array is referenced as firstname1. For 2D arrays, you reference an index by specifying two coordinates: `myarray11`.
You can use ColdFusion variables and expressions inside the brackets to reference an index, as the following example shows:

```
<cfset myArray=ArrayNew(1)>
<cfset myArray[1]="First Array Element">
<cfset myArray[1 + 1]="Second Array" & "Element">
<cfset arrayIndex=3>
<cfset arrayElement="Third Array Element">
<cfset myArray[arrayIndex]=arrayElement>
<cfset myArray[++arrayIndex]="Fourth Array Element">
<cfdump var=#myArray#>
```

Note

The `IsDefined` function does not test the existence of array elements. Instead, place any code that could try to access an undefined array element in a try block and use a catch block to handle exceptions that arise if elements do not exist.

Creating arrays

In ColdFusion, you can create arrays explicitly, by using a function to declare the array and then assigning it data, or implicitly by using an assignment statement. You can create simple or complex, multidimensional arrays.

Creating arrays using functions

To create an array explicitly, you use the `arrayNew` function and specify the array dimensions, as in the following example:

```
<cfset myNewArray=ArrayNew(2)>
```

This line creates a two-dimensional array named myNewArray. You use this method to create an array with up to three dimensions.
After you create an array, you add array elements, which you can then reference by using the element indexes. For example, suppose you create a one-dimensional array called firstname:

```
<cfset firstname=ArrayNew(1)>
```

The array firstname holds no data and is of an unspecified length. Next you add data to the array:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cfset firstname[1]="Coleman">
<cfset firstname[2]="Charlie">
<cfset firstname[3]="Dexter">
```

After you add these names to the array, it has a length of 3.

Creating and using arrays implicitly

To create an array implicitly, you do not use the `ArrayNew` function. Instead, you use a new variable name on the left side of an assignment statement, and array notation on the right side of the statement, as in the following example:

```
<cfset firstnameImplicit=["Coleman","Charlie","Dexter"]>
```

This single statement is equivalent to the four statements used to create the `firstname` array in Creating arrays using functions.
When you create an array implicitly, the right side of the assignment statement has brackets ([]) surrounding the array contents and commas separating the individual array elements. The elements can be literal values, such as the strings in the example, variables, or expressions. If you specify variables, do not place the variable names in quotation marks.
You can create an empty array implicitly, as in the following example:

```
<cfset myArray = []>
```

You can also create an array implicitly by assigning a single entry, as the following example shows:

```
<cfset chPar[1] = "Charlie">
<cfset chPar[2] = "Parker">
```

ColdFusion does not allow nested implicit creation of arrays, structures, or arrays and structures. Therefore, you cannot create a multidimensional array in a single implicit statement. For example, neither of the following statements is valid:

```
<cfset myArray = [[],[]]>
<cfset jazzmen = [["Coleman","Charlie"],["Hawkins", "Parker"]]
```

To create a two-dimensional array, for example, use a format such as the following:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cfset ch = ["Coleman", "Hawkins"]>
<cfset cp = ["Charlie", "Parker"]>
<cfset dg = ["Dexter", "Gordon"]>
<cfset players = [ch, cp, dg]>
```

You cannot use a dynamic variable when you create an array implicitly. For example, the following expression generates an error:

```
<cfset i="CP">
<cfset "#i#"=["Charlie","Parker"]>
```

Creating complex multidimensional arrays

ColdFusion supports dynamic multidimensional arrays. When you declare an array with the `ArrayNew` function, you specify the number of dimensions. You can create an asymmetrical array or increase the number of dimensions by nesting arrays as array elements.
It is important to know that when you assign one array (array1) to an element of another array (array2), array1 is copied into array2. The original copy of array1 still exists, independent of array2. You can then change the contents of the two arrays independently.
The best way to understand an asymmetrical array is by looking at it. The following example creates an asymmetric, multidimensional, array, and the `cfdump` tag displays the resulting array structure. Several array elements do not yet contain data.

```
<cfset myarray=ArrayNew(1)>
<cfset myotherarray=ArrayNew(2)>
<cfset biggerarray=ArrayNew(3)>

<cfset biggerarray[1][1][1]=myarray>
<cfset biggerarray[1][1][1][10]=3>
<cfset biggerarray[2][1][1]=myotherarray>
<cfset biggerarray[2][1][1][4][2]="five deep">

<cfset biggestarray=ArrayNew(3)>
<cfset biggestarray[3][1][1]=biggerarray>
<cfset biggestarray[3][1][1][2][3][1]="This is complex">
<cfset myarray[3]="Can you see me">

<cfdump var=#biggestarray#><br>
<cfdump var=#myarray#>
```

Note

The `cfdump` tag displays the entire contents of an array. It is an excellent tool for debugging arrays and array-handling code.

Reviewing the code

© 2014 Adobe Systems Incorporated. All rights reserved.

The following table describes the code:

Code	Description

`<cfset myarray=ArrayNew(1)>`
`<cfset myotherarray=ArrayNew(2)>`
`<cfset biggerarray=ArrayNew(3)>`
	Create three empty arrays, a 1D array, a 2D array, and a 3D array.

`<cfset biggerarray[1][1][1]=myarray>` `<cfset biggerarray[1][1][1][10]=3>`	Make element 111 of the 3D `biggerarray` array be a copy of the 1D array. Assign 3 to the 11110 element of the resulting array. The biggerarray array is now asymmetric. For example, it does not have a 1121 ele ment.

`<cfset biggerarray[2][1][1]=myotherarray>`
 	
`<cfset biggerarray[2][1][1][4][2]="five deep">`
	Make element 211 of the 3D array be the 2D array, and assign the 21142 element the value `"five deep"`. The biggerarray array is now even more asymmetric.

`<cfset biggestarray=ArrayNew(3)>`
`<cfset biggestarray[3][1][1]=biggerarray>`
 	
`<cfset biggestarray[3][1][1][2][3][1]="This is complex">`
	Create a second 3D array. Make the 311 element of this array a copy of the biggerarray array, and assign element 311231. The resulting array is complex and asymmetric.

`<cfset myarray[3]="Can you see me">`	Assign a value to element 3 of myarray.
`<cfdump var=#biggestarray#> ` `<cfdump var=#myarray#>`	Use cfdump to view the structure of biggestarray and myarray. Notice that the "Can you see me" entry appears in myarray, but not in biggestarray, because biggestarray has a copy of the original myarray values and the change to myarray does not affect it.

Using implicitly created arrays

© 2014 Adobe Systems Incorporated. All rights reserved.

You can use implicitly created arrays directly in functions (including user-defined functions) and tags. For example, the following code uses two implicit arrays, one in a ColdFusion function, the other in a user-defined function:

```
<cffunction name="sumarray">
 <cfargument name="inarray" type="array">
    <cfset result = 0>
 <cfloop array="#inarray#" index="i" >
   <cfset result += i>
     </cfloop>
     <cfreturn result>
</cffunction>

<cfoutput>
Summed Implicit array [#ArrayToList([1,2,3,4,5,6])#]: #sumarray([1,2,3,4,5,6])#<br
/>
</cfoutput>
```

Adding elements to an array

You can add an element to an array by assigning the element a value or by using a ColdFusion function.

Adding an array element by assignment

You can add elements to an array by defining the value of an array element, as shown in the following `cfset` tag:

```
<cfset myarray[5]="Test Message">
```

If an element does not exist at the specified index, ColdFusion creates it. If an element exists at the specified index, ColdFusion replaces it with the new value. To prevent existing data from being overwritten, use the `ArrayInsertAt` function, as described in the next section.
If elements with lower-number indexes do not exist, they remain undefined. Assign values to undefined array elements before you can use them. For example, the following code creates an array and an element at index 4. It outputs the contents of element 4, but generates an error when it tries to output the (nonexistent) element 3.

```
<cfset myarray=ArrayNew(1)>
<cfset myarray[4]=4>
<cfoutput>
 myarray4: #myarray[4]#<br>
 myarray3: #myarray[3]#<br>
</cfoutput>
```

Adding an array element with a function

You can use the following array functions to add data to an array:

Function	Description
ArrayAppend	Creates an array element at the end of the array.

© 2014 Adobe Systems Incorporated. All rights reserved.

ArrayPrepend	Creates an array element at the beginning of the array.
ArrayInsertAt	Inserts an array element at the specified index position.

Because ColdFusion arrays are dynamic, if you add or delete an element from the array, any higher-numbered index values all change. For example, the following code creates a two element array and displays the array contents. It then uses `ArrayPrepend` to insert a new element at the beginning of the array and displays the result. The data that was originally in indexes 1 and 2 is now in indexes 2 and 3.

```
<!--- Create an array with three elements. --->
<cfset myarray=ArrayNew(1)>
<cfset myarray[1]="Original First Element">
<cfset myarray[2]="Original Second Element">
<!--- Use cfdump to display the array structure --->
<cfdump var=#myarray#>
<br>
<!--- Add a new element at the beginning of the array. --->
<cfscript>
 ArrayPrepend(myarray, "New First Element");
</cfscript>
<!--- Use cfdump to display the new array structure. --->
<cfdump var=#myarray#>
```

For more information about these array functions, see the *CFML Reference*.

Deleting elements from an array

Use the `ArrayDeleteAt` function to delete data from the array at a particular index, instead of setting the data value to zero or an empty string. If you remove data from an array, the array resizes dynamically, as the following example shows:

```
<!--- Create an array with three elements --->
<cfset firstname=ArrayNew(1)>
<cfset firstname[1]="Robert">
 <cfset firstname[2]="Wanda">
 <cfset firstname[3]="Jane">
<!--- Delete the second element from the array --->
<cfset temp=ArrayDeleteAt(firstname, 2)>
<!--- Display the array length (2) and its two entries,
 which are now "Robert" and "Jane" --->
<cfoutput>
 The array now has #ArrayLen(firstname)# indexes<br>
 The first entry is #firstname[1]#<br>
  The second entry is #firstname[2]#<br>
 </cfoutput>
```

The `ArrayDeleteAt` function removed the original second element and resized the array so that it has two entries, with the second element now being the original third element.

Copying arrays

© 2014 Adobe Systems Incorporated. All rights reserved.

You can copy arrays of simple variables (numbers, strings, Boolean values, and date-time values) by assigning the original array to a new variable name. You do not have to use `ArrayNew` to create the array first. When you assign the existing array to a new variable, ColdFusion creates an array and copies the contents of the old array to the new array. The following example creates and populates a two-element array. It then copies the original array, changes one element of the copied array and dumps both arrays. As you can see, the original array is unchanged and the copy has a new second element.

```
<cfset myArray=ArrayNew(1)>
<cfset myArray[1]="First Array Element">
<cfset myArray[2]="Second Array Element">
<cfset newArray=myArray>
<cfset newArray[2]="New Array Element 2">
<cfdump var=#myArray#><br>
<cfdump var=#newArray#>
```

If your array contains complex variables (structures, query objects, or external objects such as COM objects) assigning the original array to a new variable does not make a complete copy of the original array. The array structure is copied; however, the new array does not get its own copy of the complex data, only references to it. To demonstrate this behavior, run the following code:

```
Create an array that contains a structure.<br>
<cfset myStruct=StructNew()>
<cfset myStruct.key1="Structure key 1">
<cfset myStruct.key2="Structure key 2">
<cfset myArray=ArrayNew(1)>
<cfset myArray[1]=myStruct>
<cfset myArray[2]="Second array element">
<cfdump var=#myArray#><br>
<br>
Copy the array and dump it.<br>
<cfset myNewArray=myArray>
<cfdump var=#myNewArray#><br>
<br>
Change the values in the new array.<br>
<cfset myNewArray[1].key1="New first array element">
<cfset myNewArray[2]="New second array element">
<br>
Contents of the original array after the changes:<br>
<cfdump var=#myArray#><br>
Contents of the new array after the changes:<br>
<cfdump var=#myNewArray#>
```

The change to the new array also changes the contents of the structure in the original array.
To make a complete copy of an array that contains complex variables, use the `Duplicate` function.

© 2014 Adobe Systems Incorporated. All rights reserved.

Populating arrays with data

Array elements can store any values, including queries, structures, and other arrays. You can use assignment statements to populate an array. You can also use several of functions to populate an array with data, including `ArraySet`, `ArrayAppend`, `ArrayInsertAt`, and `ArrayPrepend`. These functions are useful for adding data to an existing array.

In particular, consider using the following techniques:

- Populating an array with the `ArraySet` function
- Populating an array with the `cfloop` tag
- Populating an array from a query

Populating an array with the ArraySet function

You can use the `ArraySet` function to populate a 1D array, or one dimension of a multidimensional array, with some initial value, such as an empty string or zero. This can be useful to create an array of a certain size, without adding data to it right away. One reason to do this is so that you can reference all the array indexes. If you reference an array index that does not contain some value, such as an empty string, you get an error.
The `ArraySet` function has the following form:

```
ArraySet (arrayname, startrow, endrow, value)
```

The following example initializes the array myarray, indexes 1 - 100, with an empty string:

```
ArraySet (myarray, 1, 100, "")
```

Populating an array with the cfloop tag

The `cfloop` tag provides a common and efficient method for populating an array. The following example uses a `cfloop` tag and the `MonthAsString` function to populate a simple 1D array with the names of the months. A second `cfloop` outputs data in the array to the browser.

```
<cfset months=arraynew(1)>

<cfloop index="loopcount" from=1 to=12>
 <cfset months[loopcount]=MonthAsString(loopcount)>
</cfloop>

<cfloop index="loopcount" from=1 to=12>
 <cfoutput>
  #months[loopcount]#<br>
 </cfoutput>
</cfloop>
```

Using nested loops for 2D and 3D arrays

To output values from 2D and 3D arrays, employ nested loops to return array data. With a one-dimensional (1D)

© 2014 Adobe Systems Incorporated. All rights reserved.

array, a single cfloop is sufficient to output data, as in the previous example. With arrays of dimension greater than one, you maintain separate loop counters for each array level.

Nesting cfloop tags for a 2D array

The following example shows how to handle nested cfloop tags to output data from a 2D array. It also uses nested cfloop tags to populate the array:

```
<cfset my2darray=arraynew(2)>
<cfloop index="loopcount" from=1 to=12>
 <cfloop index="loopcount2" from=1 to=2>
  <cfset my2darray[loopcount][loopcount2]=(loopcount * loopcount2)>
 </cfloop>
</cfloop>

<p>The values in my2darray are currently:</p>

<cfloop index="OuterCounter" from="1" to="#ArrayLen(my2darray)#">
 <cfloop index="InnerCounter" from="1"to="#ArrayLen(my2darray[OuterCounter])#">
  <cfoutput>
   <b>[#OuterCounter#][#InnerCounter#]</b>:
   #my2darray[OuterCounter][InnerCounter]#<br>
  </cfoutput>
 </cfloop>
</cfloop>
```

Nesting cfloop tags for a 3D array

For 3D arrays, you simply nest an additional cfloop tag. (This example does not set the array values first to keep the code short.)

```
<cfloop index="Dim1" from="1" to="#ArrayLen(my3darray)#">
 <cfloop index="Dim2" from="1" to="#ArrayLen(my3darray[Dim1])#">
  <cfloop index="Dim3" from="1"to="#ArrayLen(my3darray[Dim1][Dim2])#">
   <cfoutput>
    <b>[#Dim1#][#Dim2#][#Dim3#]</b>:
     #my3darray[Dim1][Dim2][Dim3]#<br>
   </cfoutput>
  </cfloop>
 </cfloop>
</cfloop>
```

Populating an array from a query

When populating an array from a query, remember the following:

- You cannot add query data to an array all at once. A looping structure is often required to populate an array from a query.
- You can reference query column data using array-like syntax. For example, myquery.col_name1 references data in the first row in the col_name column of the myquery query.
- Inside a cfloop query= loop, you do not have to specify the query name to reference the query variables. You can use a cfset tag with the following syntax to define values for array indexes:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cfset arrayName[index]=queryColumn[row]>
```

In the following example, a `cfloop` tag places four columns of data from a sample data source into an array, myarray.

```
<!--- Do the query --->
<cfquery name="test" datasource="cfdocexamples">
 SELECT Emp_ID, LastName, FirstName, Email
 FROM Employees
</cfquery>

<!--- Declare the array --->
<cfset myarray=arraynew(2)>

<!--- Populate the array row by row --->
<cfloop query="test">
 <cfset myarray[CurrentRow][1]=Emp_ID>
 <cfset myarray[CurrentRow][2]=LastName>
 <cfset myarray[CurrentRow][3]=FirstName>
 <cfset myarray[CurrentRow][4]=Email>
</cfloop>

<!--- Now, create a loop to output the array contents --->
<cfset total_records=test.recordcount>
<cfloop index="Counter" from=1 to="#Total_Records#">
 <cfoutput>
  ID: #MyArray[Counter][1]#,
  LASTNAME: #MyArray[Counter][2]#,
  FIRSTNAME: #MyArray[Counter][3]#,
  EMAIL: #MyArray[Counter][4]# <br>
 </cfoutput>
</cfloop>
```

This example uses the query object built-in variable CurrentRow to index the first dimension of the array.

© 2014 Adobe Systems Incorporated. All rights reserved.

Array functions-Developing guide

The following functions are available for creating, editing, and handling arrays:

Function	Description
ArrayAppend	Appends an array element to the end of a specified array.
ArrayAvg	Returns the average of the values in the specified array.
ArrayClear	Deletes all data in a specified array.
ArrayDeleteAt	Deletes an element from a specified array at the specified index and resizes the array.
ArrayInsertAt	Inserts an element (with data) in a specified array at the specified index and resizes the array.
ArrayIsDefined	Returns True if the specified array is defined.
ArrayIsEmpty	Returns True if the specified array is empty of data.
arrayLen	Returns the length of the specified array.
ArrayMax	Returns the largest numeric value in the specified array.
ArrayMin	Returns the smallest numeric value in the specified array.
ArrayNew	Creates an array of specified dimension.
ArrayPrepend	Adds an array element to the beginning of the specified array.
ArrayResize	Resets an array to a specified minimum number of elements.
ArraySet	Sets the elements in a 1D array in a specified range to a specified value.
ArraySort	Returns the specified array with elements sorted numerically or alphanumerically.
ArraySum	Returns the sum of values in the specified array.
ArraySwap	Swaps array values in the specified indexes.

© 2014 Adobe Systems Incorporated. All rights reserved.

ArrayToList	Converts the specified 1D array to a list, delimited with the character you specify.
IsArray	Returns True if the value is an array.
ListToArray	Converts the specified list, delimited with the character you specify, to an array.

For more information about each of these functions, see the *CFML Reference*.
If a function returns an array, you can now reference a specific element array directly in the function call statement. For example, the following line references the fifth element of the array returned by the myFunc() function:

```
myFunc()[5]
```

© 2014 Adobe Systems Incorporated. All rights reserved.

About structures

ColdFusion *structures* consist of key-value pairs. Structures let you build a collection of related variables that are grouped under a single name. You can define ColdFusion structures dynamically.
You can use structures to reference related values as a unit, rather than individually. To maintain employee lists, for example, you can create a structure that holds personnel information such as name, address, phone number, ID numbers, and so on. Then you can reference this collection of information as a structure called *employee* rather than as a collection of individual variables.
A structure *key* must be a string. The *values* associated with the key can be any valid ColdFusion value or object. It can be a string or integer, or a complex object such as an array or another structure. Because structures can contain any types of data, they provide a powerful and flexible mechanism for representing complex data.

Structure notation

ColdFusion supports three types of notation for referencing structure contents. The notation that you use depends on your requirements.

Notation	Description
Object.property	You can reference a property, prop, of an object, obj, as obj.prop. This notation, also called dot notation, is useful for simple assignments, as in this example:`depts.John="Sales"` Use this notation only when you know the property names (keys) in advance and they are strings, with no special characters, numbers, or spaces. You cannot use the dot notation when the property, or key, is dynamic.

Associative arrays	If you do not know the key name in advance, or it contains spaces, numbers, or special characters, you can use associative array notation. This notation uses structures as arrays with string indexes; for example:
	`depts["John"]="Sales"`
	`depts[employeeName] = "Sales"`
	You can use a variable (such as employeeName) as an associative array index. Therefore, enclose any literal key names in quotation marks. For information on using associative array references containing variables, see *Dynamically constructing structure references* in Dynamic expressions and dynamic variables.

© 2014 Adobe Systems Incorporated. All rights reserved.

Structure	Use structure notation only when you create structures and set their initial values, not when you are accessing or updating structure data, and only on the right side of an assignment expression. This notation has the following format:
	`{keyName=value[,keyName=value]...}`
	where the square braces ([]) and ellipses (\...) indicate optional contents that can be repeated.The following example creates a structure that uses structure notation:
	`<cfset name={firstName = "John", lastName = "Smythe"}>`

Referencing complex structures

When a structure contains another structure, you reference the data in the nested structure by extending either object.property or associative array notation. You can even use a mixture of both notations.

For example, if structure1 has a key key1 whose value is a structure that has keys struct2key1, struct2key2, and so on, you can use any of the following references to access the data in the first key of the embedded structure:

```
Structure1.key1.Struct2key1
Structure1["key1"].Struct2key1
Structure1.key1["Struct2key1"]
Structure1["key1"]["Struct2key1"]
```

The following example shows various ways you can reference the contents of a complex structure:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cfset myArray=ArrayNew(1)>
<cfset myArray[1]="2">
<cfset myArray[2]="3">
<cfset myStruct2=StructNew()>
<cfset myStruct2.struct2key1="4">
<cfset myStruct2.struct2key2="5">
<cfset myStruct=StructNew()>
<cfset myStruct.key1="1">
<cfset myStruct.key2=myArray>
<cfset myStruct.key3=myStruct2>
<cfdump var=#myStruct#><br>

<cfset key1Var="key1">
<cfset key2Var="key2">
<cfset key3Var="key3">
<cfset var2="2">

<cfoutput>
Value of the first key<br>
#mystruct.key1#<br>
#mystruct["key1"]#<br>
#mystruct[key1Var]#<br>
<br>
Value of the second entry in the key2 array<br>
#myStruct.key2[2]#<br>
#myStruct["key2"][2]#<br>
#myStruct[key2Var][2]#<br>
#myStruct[key2Var][var2]#<br>
<br>
Value of the struct2key2 entry in the key3 structure<br>
#myStruct.key3.struct2key2#<br>
#myStruct["key3"]["struct2key2"]#<br>
#myStruct[key3Var]["struct2key2"]#<br>
#myStruct.key3["struct2key2"]#<br>
#myStruct["key3"].struct2key2#<br>
<br>
</cfoutput>
```

Reviewing the code

The following table describes the code:

Code	Description

© 2014 Adobe Systems Incorporated. All rights reserved.

`<cfset myArray=ArrayNew(1)>`
`<cfset myArray[1]="2">`
`<cfset myArray[2]="3">`
`<cfset myStruct2=StructNew()>`
`<cfset myStruct2.struct2key1="4">`
`<cfset myStruct2.struct2key2="5">`
`<cfset myStruct=StructNew()>`
`<cfset myStruct.key1="1">`
`<cfset myStruct.key2=myArray>`
`<cfset myStruct.key3=myStruct2>`
	Create a structure with three entries: a string, an array, and an embedded structure.

`<cfdump var=#myStruct#> `	Display the complete structure.

`<cfset key1Var="key1">`
`<cfset key2Var="key2">`
`<cfset key3Var="key3">`
	Create variables containing the names of the myStruct keys and the number 2.

`<cfoutput>` `Value of the first key ` `#mystruct.key1# ` `#mystruct["key1"]# ` `#mystruct[key1Var]# ` ` `	Output the value of the key1 (string) entry using the following notation: • object.property notation • associative array notation with a constant • associative array notation with a variable
` ` `Value of the second entry in the key2 array ` `#myStruct.key2[2]# ` `#myStruct["key2"][2]# ` `#myStruct[key2Var][2]# ` `#myStruct[key2Var][var2]# ` ` `	Output the value of the second entry in the key2 array using the following notation: • object.property notation • associative array notation with a constant • associative array notation with a variable • associative array notation with variables for both the array and the array index

© 2014 Adobe Systems Incorporated. All rights reserved.

`Value of the struct2key2 entry in the key3 structure ` `#myStruct.key3.struct2key2# ` `#myStruct["key3"]["struct2key2"]# ` `#myStruct[key3Var]["struct2key2"]# ` `#myStruct.key3["struct2key2"]# ` `#myStruct["key3"].struct2key2# ` ` `	Output the value of second entry in the key3 embedded structure using the following notation: • object.property notation • associative array notation with two constants • associative array notation with a variable and a constant • object.property notation followed by associative array notation • associative array notation followed by object.property notation

© 2014 Adobe Systems Incorporated. All rights reserved.

Creating and using structures

The sample code in the following discussions uses a structure called *employee,* which is used to add new employees to a corporate information system.

Creating structures

In ColdFusion, you can create structures explicitly by using a function, and then populate the structure using assignment statements or functions, or you can create the structure implicitly by using an assignment statement.

Creating structures using functions

You can create structures by assigning a variable name to the structure with the `StructNew` function as follows:

```
<cfset structName = StructNew()>
```

For example, to create a structure named departments, use the following syntax:

```
<cfset departments = StructNew()>
```

This statement creates an empty structure to which you can add data.

Creating structures implicitly

You can create an empty structure implicitly, as in the following example:

```
<cfset myStruct = {}>
```

You can also create a structure by assigning data to a variable. For example, each of the following lines creates a structure named `myStruct` with one element, `name`, that has the value Adobe Systems Incorporated.

```
<cfset coInfo.name = "Adobe Systems Incorporated">
<cfset coInfo["name"] = "Adobe Systems Incorporated">
<cfset coInfo = {name = "Adobe Systems Incorporated"}>
```

When you use structure notation to create a structure, as shown in the third example, you can populate multiple structure fields. The following example shows this use:

```
<cfset coInfo={name="Adobe Systems Incorporated", industry="software"}>
```

ColdFusion does not allow nested implicit creation of structures, arrays, or structures and arrays. The following line, for example, generates an error:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cfset myStruct = {structKey1 = {innerStructKey1 = "innerStructValue1"}}>
```

Similarly, you cannot use object.property notation on the left side of assignments inside structure notation. The following statement, for example, causes an error:

```
<cfset myStruct={structKey1.innerStructKey1 = "innerStructValue1"}>
```

Instead of using these formats, use multiple statements, such as the following:

```
<cfset innerStruct1 = {innerStructKey1 = "innerStructValue1"}>
<cfset myStruct1={structKey1 = innerStruct1}>
```

You cannot use a dynamic variable when you create a structure implicitly. For example, the following expression generates an error:

```
<cfset i="coInfo">
<cfset "#i#"={name = "Adobe Systems Incorporated"}>
```

Using implicitly created structures in functions and tags

You can use implicitly created structures directly in functions (including user-defined functions) and tags. For example, the following code dumps an implicitly created structure.

```
<cfdump var="#{Name ="28 Weeks Later", Time = "7:45 PM"}#">
```

You can use array notation inside the structure notation, as shown in the following example:

```
<cfset student = {firstName="Jane", lastName="Janes", grades=[91, 78, 87]}>
```

Adding and updating structure elements

You add or update a structure element to a structure by assigning the element a value or by using a ColdFusion function. It is simpler and more efficient to use direct assignment.
You can add structure key-value pairs by defining the value of the structure key, as the following example shows:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cfset myNewStructure.key1="A new structure with a new key">
<cfdump var=#myNewStructure#>
<cfset myNewStructure.key2="Now I've added a second key">
<cfdump var=#myNewStructure#>
```

The following code uses `cfset` and object.property notation to create a structure element called departments.John, and changes John's department from Sales to Marketing. It then uses associative array notation to change his department to Facilities. Each time the department changes, it displays the results:

```
<cfset departments=structnew()>
<cfset departments.John = "Sales">
<cfoutput>
 Before the first change, John was in the #departments.John# Department<br>
</cfoutput>
<cfset Departments.John = "Marketing">
<cfoutput>
 After the first change, John is in the #departments.John# Department<br>
</cfoutput>
<cfset Departments["John"] = "Facilities">
<cfoutput>
 After the second change, John is in the #departments.John# Department<br>
</cfoutput>
```

Getting information about structures and keys

You use ColdFusion functions to find information about structures and their keys.

Getting information about structures

To find out if a given value represents a structure, use the `IsStruct` function, as follows:

```
IsStruct(variable)
```

This function returns True if *variable* is a ColdFusion structure. (It also returns True if *variable* is a Java object that implements the java.util.Map interface.)
Structures are not indexed numerically, so to find out how many name-value pairs exist in a structure, use the `StructCount` function, as in the following example:

```
StructCount(employee)
```

To discover whether a specific Structure contains data, use the `StructIsEmpty` function, as follows:

```
StructIsEmpty(structure_name)
```

© 2014 Adobe Systems Incorporated. All rights reserved.

This function returns True if the structure is empty, and False if it contains data.

Finding a specific key and its value

To determine whether a specific key exists in a structure, use the `StructKeyExists` function, as follows:

```
StructKeyExists(structure_name, "key_name")
```

Do *not* place the name of the structure in quotation marks, but you do place the key name in quotation marks. For example, the following code displays the value of the MyStruct.MyKey only if it exists:

```
<cfif StructKeyExists(myStruct, "myKey")>
 <cfoutput> #mystruct.myKey#</cfoutput><br>
</cfif>
```

You can use the `StructKeyExists` function to dynamically test for keys by using a variable to represent the key name. In this case, you do not place the variable in quotation marks. For example, the following code loops through the records of the GetEmployees query and tests the myStruct structure for a key that matches theLastName field of the query. If ColdFusion finds a matching key, it displays the Last Name from the query and the corresponding entry in the structure.

```
<cfloop query="GetEmployees">
<cfif StructKeyExists(myStruct, LastName)>
 <cfoutput>#LastName#: #mystruct[LastName]#</cfoutput><br>
</cfif>
</cfloop>
```

If the name of the key is known in advance, you can also use the ColdFusion `IsDefined` function, as follows:

```
IsDefined("structure_name.key")>
```

However, if the key is dynamic, or contains special characters, use the `StructKeyExists` function.

> **Note**
>
> Using `StructKeyExists` to test for the existence of a structure entry is more efficient than using `IsDefined`. ColdFusion scopes are available as structures and you can improve efficiency by using `StructKeyExists` to test for the existence of variables.

Getting a list of keys in a structure

To get a list of the keys in a CFML structure, you use the `StructKeyList` function, as follows:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cfset temp=StructKeyList(structure_name, [delimiter])>
```

You can specify any character as the delimiter; the default is a comma.
Use the `StructKeyArray` function to returns an array of keys in a structure, as follows:

```
<cfset temp=StructKeyArray(structure_name)>
```

Note

The `StructKeyList` and `StructKeyArray` functions do not return keys in any particular order. Use the `ListSort` or `ArraySort` functions to sort the results.

Copying structures

ColdFusion provides several ways to copy structures and create structure references. The following table lists these methods and describes their uses:

Technique	Use
Duplicate function	Makes a complete copy of the structure. All data is copied from the original structure to the new structure, including the contents of structures, queries, and other objects. As a result changes to one copy of the structure have no effect on the other structure.This function is useful when you want to move a structure completely into a new scope. In particular, if a structure is created in a scope that requires locking (for example, Application), you can duplicate it into a scope that does not require locking (for example, Request), and then delete it in the scope that requires locking.
StructCopy function	Makes a shallow copy of a structure. It creates a structure and copies all simple variable and array values at the top level of the original structure to the new structure. However, it does not make copies of any structures, queries, or other objects that the original structure contains, or of any data inside these objects. Instead, it creates a reference in the new structure to the objects in the original structure. As a result, any change to these objects in one structure also changes the corresponding objects in the copied structure.The `Duplicate` function replaces this function for most, if not all, purposes.

© 2014 Adobe Systems Incorporated. All rights reserved.

Variable assignment	Creates an additional reference, or alias, to the structure. Any change to the data using one variable name changes the structure that you access using the other variable name.This technique is useful when you want to add a local variable to another scope or otherwise change the scope of a variable without deleting the variable from the original scope.

The following example shows the different effects of copying, duplicating, and assigning structure variables:

```
Create a structure<br>
<cfset myNewStructure=StructNew()>
<cfset myNewStructure.key1="1">
<cfset myNewStructure.key2="2">
<cfset myArray=ArrayNew(1)>
<cfset myArray[1]="3">
<cfset myArray[2]="4">
<cfset myNewStructure.key3=myArray>
<cfset myNewStructure2=StructNew()>
<cfset myNewStructure2.Struct2key1="5">
<cfset myNewStructure2.Struct2key2="6">
<cfset myNewStructure.key4=myNewStructure2>
<cfdump var=#myNewStructure#><br>
<br>
A StructCopy copied structure<br>
<cfset CopiedStruct=StructCopy(myNewStructure)>
<cfdump var=#CopiedStruct#><br>
<br>
A Duplicated structure<br>
<cfset dupStruct=Duplicate(myNewStructure)>
<cfdump var=#dupStruct#><br>
<br>
A new reference to a structure<br>
<cfset structRef=myNewStructure>
<cfdump var=#structRef#><br>

<br>
Change a string, array element, and structure value in the StructCopy copy.<br>
<br>
<cfset CopiedStruct.key1="1A">
<cfset CopiedStruct.key3[2]="4A">
<cfset CopiedStruct.key4.Struct2key2="6A">
Original structure<br>
<cfdump var=#myNewStructure#><br>
Copied structure<br>
<cfdump var=#CopiedStruct#><br>
Duplicated structure<br>
<cfdump var=#DupStruct#><br>
Structure reference
<cfdump var=#structRef#><br>
<br>
Change a string, array element, and structure value in the Duplicate.<br>
<br>
<cfset DupStruct.key1="1B">
<cfset DupStruct.key3[2]="4B">
<cfset DupStruct.key4.Struct2key2="6B">
```

© 2014 Adobe Systems Incorporated. All rights reserved.

```
Original structure<br>
<cfdump var=#myNewStructure#><br>
Copied structure<br>
<cfdump var=#CopiedStruct#><br>
Duplicated structure<br>
<cfdump var=#DupStruct#><br>
Structure reference
<cfdump var=#structRef#><br>
<br>
Change a string, array element, and structure value in the reference.<br>
<br>
<cfset structRef.key1="1C">
<cfset structRef.key3[2]="4C">
<cfset structRef.key4.Struct2key2="6C">
Original structure<br>
<cfdump var=#myNewStructure#><br>
Copied structure<br>
<cfdump var=#CopiedStruct#><br>
Duplicated structure<br>
<cfdump var=#DupStruct#><br>
Structure reference
<cfdump var=#structRef#><br>
<br>
Clear the original structure<br>
<cfset foo=structclear(myNewStructure)>
Original structure:<br>
<cfdump var=#myNewStructure#><br>
Copied structure<br>
<cfdump var=#CopiedStruct#><br>
Duplicated structure<br>
<cfdump var=#DupStruct#><br>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

```
Structure reference:<br>
<cfdump var=#structRef#><br>
```

Deleting structure elements and structures

To delete a key and its value from a structure, use the `StructDelete` function, as follows:

```
StructDelete(structure_name, key [, indicateNotExisting ])
```

The *indicateNotExisting* argument tells the function what to do if the specified key does not exist. By default, the function always returns True. However, if you specify True for the *indicateNotExisting* argument, the function returns True if the key exists and False if it does not.
You can also use the `StructClear` function to delete all the data in a structure but keep the structure instance itself, as follows:

```
StructClear(structure_name)
```

If you use `StructClear` to delete a structure that you have copied using the `StructCopy` function, the specified structure is deleted, but the copy is unaffected.
If you use `StructClear` to delete a structure that has multiple references, the function deletes the contents of the structure and all references point to the empty structure, as the following example shows:

```
<cfset myStruct.Key1="Adobe">
Structure before StructClear<br>
<cfdump var="#myStruct#">
<cfset myCopy=myStruct>
<cfset StructClear(myCopy)>
After Clear:<br>
myStruct: <cfdump var="#myStruct#"><br>
myCopy: <cfdump var="#myCopy#">
```

Looping through structures

You can loop through a structure to output its contents, as the following example shows:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<!--- Create a structure and set its contents. --->
<cfset departments=structnew()>

<cfset val=StructInsert(departments, "John", "Sales")>
<cfset val=StructInsert(departments, "Tom", "Finance")>
<cfset val=StructInsert(departments, "Mike", "Education")>

<!--- Build a table to display the contents --->
<cfoutput>
<table cellpadding="2" cellspacing="2">
 <tr>
  <td><b>Employee</b></td>
  <td><b>Department</b></td>
 </tr>
 <!--- Use cfloop to loop through the departments structure.
 The item attribute specifies a name for the structure key. --->
 <cfloop collection=#departments# item="person">
  <tr>
   <td>#person#</td>
   <td>#Departments[person]#</td>
  </tr>
 </cfloop>
</table>
</cfoutput>
```

Community contributed help

Some more valid statements:

<cfset myStruct = {structKey1 = {innerStructKey1 = "innerStructValue1"}}>

<cfset myStruct={structKey1.innerStructKey1 = "innerStructValue1"}>

<cfset "#i#"={name = "Adobe Systems Incorporated"}>

You can also use a colon:

<cfset myStruct={structKey1.innerStructKey1 : "innerStructValue1"}>

© 2014 Adobe Systems Incorporated. All rights reserved.

Structure examples

Structures are useful for grouping a set of variables under a single name. The following example uses structures to collect information from a form, and to submit that information to a custom tag, named `cf_addemployee`. For information on creating and using custom tags, see Creating and Using Custom CFML Tags.

Example file newemployee.cfm

The following ColdFusion page shows how to create structures and use them to add data to a database. It calls the `cf_addemployee` custom tag, which is defined in the addemployee.cfm file.

```
<html>
<head>
<title>Add New Employees</title>
</head>

<body>
<h1>Add New Employees</h1>
<!--- Action page code for the form at the bottom of this page. --->

<!--- Establish parameters 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="">

<!--- If at least the firstname form field is passed, create
   a structure named employee and add values. --->
<cfif #Form.firstname# eq "">
 <p>Please fill out the form.</p>
<cfelse>
<cfoutput>
  <cfscript>
  employee=StructNew();
  employee.firstname = Form.firstname;
  employee.lastname = Form.lastname;
  employee.email = Form.email;
  employee.phone = Form.phone;
  employee.department = Form.department;
  </cfscript>

<!--- Display results of creating the structure. --->
  First name is #StructFind(employee, "firstname")#<br>
  Last name is #StructFind(employee, "lastname")#<br>
  EMail is #StructFind(employee, "email")#<br>
  Phone is #StructFind(employee, "phone")#<br>
  Department is #StructFind(employee, "department")#<br>
 </cfoutput>

<!--- Call the custom tag that adds employees. --->
 <cf_addemployee empinfo="#employee#">
</cfif>

<!--- The form for adding the new employee information --->
<hr>
<form action="newemployee.cfm" method="Post">
```

© 2014 Adobe Systems Incorporated. All rights reserved.

```
First Name: 
<input name="firstname" type="text" hspace="30" maxlength="30"><br>
Last Name: 
<input name="lastname" type="text" hspace="30" maxlength="30"><br>
EMail: 
<input name="email" type="text" hspace="30" maxlength="30"><br>
Phone: 
<input name="phone" type="text" hspace="20" maxlength="20"><br>
Department: 
<input name="department" type="text" hspace="30" maxlength="30"><br>

<input type="Submit" value="OK">
</form>
<br>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

```
</body>
</html>
```

Reviewing the code

The following table describes the code:

Code	Description

`<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="">`
	Set default values of all form fields so that they exist the first time this page is displayed and can be tested.

`<cfif #Form.firstname# eq "">` `<p>Please fill out the form.</p>`	Test the value of the firstname field. This field is required. The test is False the first time the page displays.If no data exists in the Form.firstname variable, display a message requesting the user to fill the form.

© 2014 Adobe Systems Incorporated. All rights reserved.

`<cfelse>` `<cfoutput>` `<cfscript>` `employee=StructNew();` `employee.firstname = Form.firstname;` `employee.lastname = Form.lastname;` `employee.email = Form.email;` `employee.phone = Form.phone;` `employee.department = Form.department;` `</cfscript>` `<!--- Display results of creating the structure. --->` `First name is #StructFind(employee, "firstname")# ` `Last name is #StructFind(employee, "lastname")# ` `EMail is #StructFind(employee, "email")# ` `Phone is #StructFind(employee, "phone")# ` `Department is #StructFind(employee, "department")# ` `</cfoutput>`	If Form.firstname contains text, the user submitted the form. Use CFScript to create a structure named employee and fill it with the form field data. Then display the contents of the structure.
`<cf_addemployee empinfo="#employee#">` `</cfif>`	Call the `cf_addemployee` custom tag and pass it a copy of the employee structure in the empinfo attribute. The `duplicate` function ensures that the custom tag gets a copy of the employee structure, not the original. Although this is not necessary in this example, it is good practice because it prevents the custom tag from modifying the calling contents of the structure in the calling page.

© 2014 Adobe Systems Incorporated. All rights reserved.

`<form action="newemployee.cfm"` `method="Post">` `First Name: ` `<input name="firstname"` `type="text" hspace="30"` `maxlength="30"> ` `Last Name: ` `<input name="lastname" type="text"` `hspace="30" maxlength="30"> ` `EMail: ` `<input name="email" type="text"` `hspace="30" maxlength="30"> ` `Phone: ` `<input name="phone" type="text"` `hspace="20" maxlength="20"> ` `Department: ` `<input name="department"` `type="text" hspace="30"` `maxlength="30"> ` `<input type="Submit" value="OK">` `</form>`	The data form. When the user clicks OK, the form posts the data to this ColdFusion page.

Example file addemployee.cfm

The following file is an example of a custom tag used to add employees. Employee information is passed through the employee structure (the `empinfo` attribute). For databases that do not support automatic key generation, also add the Emp_ID.

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cfif StructIsEmpty(attributes.empinfo)>
 <cfoutput>
  Error. No employee data was passed.<br>
 </cfoutput>
 <cfexit method="ExitTag">
<cfelse>
 <!--- Add the employee --->
 <cfquery name="AddEmployee" datasource="cfdocexamples">
  INSERT INTO Employees
   (FirstName, LastName, Email, Phone, Department)
  VALUES (
  '#attributes.empinfo.firstname#' ,
  '#attributes.empinfo.lastname#' ,
  '#attributes.empinfo.email#' ,
  '#attributes.empinfo.phone#' ,
  '#attributes.empinfo.department#' )
 </cfquery>
</cfif>
<cfoutput>
 <hr>Employee Add Complete
</cfoutput>
```

Reviewing the code

The following table describes the code:

Code	Description
`<cfif StructIsEmpty(attributes.empinfo)>` `<cfoutput>` ` Error. No employee data was passed. ` `</cfoutput>` `<cfexit method="ExitTag">`	If the custom tag was called without an `empinfo` attribute, displays an error message and exit the tag.

© 2014 Adobe Systems Incorporated. All rights reserved.

<cfelse> <!--- Add the employee ---> <cfquery name="AddEmployee" datasource="cfdocexamples">INSERT INTO Employees (FirstName, LastName, Email, Phone, Department) VALUES ('#attributes.empinfo.firstname#' , '#attributes.empinfo.lastname#' , '#attributes.empinfo.email#' , '#attributes.empinfo.phone#' , '#attributes.empinfo.department#') </cfquery> </cfif>	Add the employee data passed in the empinfo structure to the Employees table of the cfdocexamples database. Use direct references to the structure entries, not Struct Find functions.If the database does not support automatic generation of the Emp_ID key, add an Emp_ID entry to the form and add it to the query.
<cfoutput> <hr>Employee Add Complete </cfoutput>	Display a completion message. This code does not have to be inside the `cfelse` block because the cfexit tag prevents it from being run if the empinfo structure is empty.

© 2014 Adobe Systems Incorporated. All rights reserved.

Structure functions - Developing guide

You can use the following functions to create and manage structures in ColdFusion applications. The table describes the purpose of each function and provides specific, but limited, information that can assist you in determining whether to use the function instead of other technique.
All functions except `StructDelete` throw an exception if a referenced key or structure does not exist.
For more information on these functions, see the *CFML Reference.*

Function	Description
`Duplicate`	Returns a complete copy of the structure.
`IsStruct`	Returns True if the specified variable is a ColdFusion structure or a Java object that implements the java.util.Map interface.
`StructAppend`	Appends one structure to another.
`StructClear`	Removes all data from the specified structure.
`StructCopy`	Returns a "shallow" copy of the structure. All embedded objects are references to the objects in the original structure. The `Duplicate` function has replaced this function for most purposes.
`StructCount`	Returns the number of keys in the specified structure.
`StructDelete`	Removes the specified item from the specified structure.
`StructFind`	Returns the value associated with the specified key in the specified structure. This function is redundant with accessing structure elements using associative array notation.
`StructFindKey`	Searches through a structure for the specified key name and returns an array containing data on the found key or keys.
`StructFindValue`	Searches through a structure for the specified simple data value (for example, a string or number) and returns an array containing information on the value location in the structure.
`StructGet`	Returns a reference to a substructure contained in a structure at the specified path. This function is redundant with using direct reference to a structure. If you accidentally use this function on a variable that is not a structure, it replaces the value with an empty structure.

© 2014 Adobe Systems Incorporated. All rights reserved.

StructInsert	Inserts the specified key-value pair into the specified structure. Unlike a direct assignment statement, this function generates an error by default if the specified key exists in the structure.
StructIsEmpty	Indicates whether the specified structure contains data. Returns True if the structure contains no data, and False if it does contain data.
StructKeyArray	Returns an array of keys in the specified structure.
StructKeyExists	Returns True if the specified key is in the specified structure. You can use this function in place of the `IsDefined` function to check for the existence of variables in scopes that are available as structures.
StructKeyList	Returns a list of keys in the specified structure.
StructNew	Returns a new structure.
StructSort	Returns an array containing the key names of a structure in the order determined by the sort criteria.
StructUpdate	Updates the specified key with the specified value. Unlike a direct assignment statement, this function generates an error if the structure or key does not exist.

#back to top

© 2014 Adobe Systems Incorporated. All rights reserved.

Extending ColdFusion Pages with CFML Scripting

Adobe ColdFusion offers a server-side scripting language, CFScript, that provides ColdFusion functionality in script syntax. This JavaScript-like language gives developers the same control flow as ColdFusion, but without tags. You can also use CFScript to write user-defined functions that you can use anywhere that a ColdFusion expression is allowed.

© 2014 Adobe Systems Incorporated. All rights reserved.

About CFScript

CFScript is a language within a language. It is a scripting language that is like JavaScript but is simpler to use. Also, unlike JavaScript, CFScript only runs on the ColdFusion server; it does not run on the client system. CFScript code can use all the ColdFusion functions and expressions, and has access to all ColdFusion variables that are available its scope.
CFScript provides a compact and efficient way to write ColdFusion logic. Typical uses of CFScript include the following:

- Simplifying and speeding variable setting
- Building compact JavaScript-like flow control structures
- Creating user-defined functions
  Because you use functions and expressions directly in CFScript, you do not have to surround each assignment or function in a `cfset` tag. Also, CFScript assignments are often faster than `cfset` tags.
  CFScript provides a set of decision and flow-control structures that are more familiar than ColdFusion tags to most programmers.
  In addition to variable setting, other operations tend to be slightly faster in CFScript than in tags.
  You can use CFScript to create user-defined functions, or UDFs (also known as custom functions). You call UDFs in the same manner that you call standard ColdFusion functions. UDFs are to ColdFusion built-in functions what custom tags are to ColdFusion built-in tags. Typical uses of UDFs include data manipulation and mathematical calculation routines.
  You cannot include ColdFusion tags in CFScript. However, some functions and CFScript statements are equivalent to commonly used tags. For more information, see *Tag equivalents in CFScript* in Language Enhancements in ColdFusion 9.

Comparing tags and CFScript

The following examples show how you can use CFML tags and CFScript to do the same thing. Each example takes data submitted from a form and places it in a structure; if the form does not have a last name and department field, it displays a message.

Using CFML tags

```
<cfif IsDefined("Form.submit")>
 <cfif (Form.lastname NEQ "") AND (Form.department NEQ "")>
  <cfset employee=structnew()>
  <cfset employee.firstname=Form.firstname>
  <cfset employee.lastname=Form.lastname>
  <cfset employee.email=Form.email>
  <cfset employee.phone=Form.phone>
  <cfset employee.department=Form.department>
  <cfoutput>
   Adding #Form.firstname# #Form.lastname#<br>
  </cfoutput>
 <cfelse>
  <cfoutput>
   You must enter a Last Name and Department.<br>
  </cfoutput>
  </cfif>
</cfif>
```

Using CFScript

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cfscript>
 if (IsDefined("Form.submit")) {
  if ((Form.lastname NEQ "") AND (Form.department NEQ "")) {
   employee=StructNew();
   employee.firstname=Form.firstname;
   employee.lastname=Form.lastname;
   employee.email=Form.email;
   employee.phone=Form.phone;
   employee.department=Form.department;
   WriteOutput("Adding #Form.firstname# #Form.lastname# <br>");
   }
  else
   WriteOutput("You must enter a Last Name and Department.<br>");
  }
</cfscript>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

Language Enhancements in ColdFusion 9

The language enhancements in ColdFusion 9 include new language constructs, extended tag support, new keywords, script functions implemented as CFCs, and support for new operations.

New tag equivalents in CFScript

The following table lists the tags that have equivalents in CFScript:

Tag	Equivalent in CFScript
cfabort	abort
cfcomponent	component
cfcontinue	continue
cfdirectory Only for `<Cfdirectory action=list/>`	The directory functions DirectoryCreate, DirectoryDelete, DirectoryList, and DirectoryRename.
cfdump	writedump
cfexit	exit
cffinally	finally
cfimport	import
cfinclude	include
cfinterface	interface
cflocation	location
cflog	writelog
cfparam	param
cfprocessingdirective	pageencoding
cfproperty	property
cfrethrow	rethrow
cfthread	thread
cfthrow	throw

© 2014 Adobe Systems Incorporated. All rights reserved.

`cftrace`	trace
`cftransaction`	transaction

Script Functions added in ColdFusion 9

The following table has the list of script functions introduced in ColdFusion 9.

Function	Equivalent ColdFusion Tag
`ftp`	cfftp
http	cfhttp
mail	cfmail
pdf	cfpdf
query	cfquery
storedproc	cfstoredproc

For details of the Script Functions, see the section Script Functions Implemented as CFCs in the *CFML Reference*.

Reserved words introduced in ColdFusion 9

- import
- finally
- local (inside function declaration)
- interface
- pageencoding

© 2014 Adobe Systems Incorporated. All rights reserved.

What is supported in CFScript

Tag equivalents in CFScript

Tag	CFScript equivalent
cfabort	abort
cfbreak	break. CFScript also has a continue statement that has no equivalent CFML tag.
cfcase	case
cfcatch	catch
cfcomponent	component
cfcontinue	continue
cfcookie	Direct assignment of Cookie scope memory-only variables. You cannot use direct assignment to set persistent cookies that are stored on the user system.
cfdefaultcase	default
cfdirectory Only for `<Cfdirectory action=list/>`	The directory functions DirectoryCreate, DirectoryDelete, DirectoryList, and DirectoryRename.
cfdump	writedump
cfelse	else
cfelseif	elseif
cfexit	exit
cffile	The file functions FileDelete, FileSeek, FileSkipBytes, and FileWriteLine.
cffinally	finally
cffunction	function
cfimage	The Image functions.
cfif	if

© 2014 Adobe Systems Incorporated. All rights reserved.

cfimport	importImport in `cfscript` is only equivalent of `<cfimport path="">`. You cannot use `<cfimport taglib="">` in `cfscript`.
cfinclude	include
cfinterface	interface
cflocation	location
cflock	lock
cflog	writelog
cfloop	• Indexed cfloop: for loops • Conditional cfloop: while loops and do while loops • Structure cfloop: for in loop. (There is no equivalent for queries, lists, or objects.)
cfobject	createobject, new
cfoutput	writeoutput
cfparam	param
cfprocessingdirective	pageencoding
cfproperty	property
cfrethrow	rethrow
cfreturn	return
cfsavecontent	savecontent
cfset	var{{var x =1; }} is equivalent of `<cfset var x =1>` Assignment statement `x =1;` is equivalent of `<cfset x =1>` local.x=1; is equivalent of `<cfset var x =1>`
cfswitch	switch
cfthread	thread
cfthrow	throw
cftrace	trace

© 2014 Adobe Systems Incorporated. All rights reserved.

cftransaction	transaction
cftry	try

Example

The following example loops through a query in CFScript:

```
...
<cfscript>
// Loop through the qGetEmails RecordSet
for (x = 1; x <= qGetEmails.RecordCount; x=x+1) {
  This_id = qGetEmails.Emails_id[x];
  This_Subject = qGetEmails.Subject[x];
  This_RecFrom = qGetEmails.RecFrom[x];
  This_SentTo = qGetEmails.SentTo[x];
  This_dReceived = qGetEmails.dReceived[x];
   This_Body = qGetEmails.Body[x];
... // More code goes here.
}
</cfscript>
```

Reserved words

In addition to the names of ColdFusion functions and words reserved by ColdFusion expressions (such as NOT, AND, IS, and so on), the following words are reserved in CFScript. Do not use these words as variables or identifiers in your scripting code:

break	do	import	var
case	else	in	while
catch	finally	interface	
try	for	pageencoding	
continue	function	return	
default	if	switch	

Script functions

For a list of script functions, see *Script Functions added in ColdFusion 9*.

© 2014 Adobe Systems Incorporated. All rights reserved.

The CFScript language

The CFScript language syntax is similar to other scripting languages, and has the same types of elements.

Identifying CFScript

You enclose CFScript regions inside `<cfscript>` and `</cfscript>` tags. No other CFML tags are allowed inside a `cfscript` region. The following lines show a minimal script:

```
<cfscript>
a = 2;
</cfscript>
```

Variables

CFScript variables can be of any ColdFusion type, such as numbers, strings, arrays, queries, and objects. The CFScript code can read and write any variables that are available in the page that contains the script. These variables include all shared scopes, such as session, application, and server variables.

Expressions and operators

CFScript supports all CFML expressions. CFML expressions include operators (such as +, -, EQ, and so on), as well as all CFML functions.
You can use several comparison operators in CFScript only, not in CFML tags. (You can also use the corresponding CFML operators in CFScript.) The following table lists the CFScript-only operators and the equivalent operator that you can use in CFML tags or CFScript:

CFScript operator	CFML operator	CFScript operator	CFML operator
==	EQ	!=	NEQ
<	LT	<=	LTE
>	GT	>=	GTE

For information about CFML expressions, operators, and functions, see Using Expressions and Number Signs.

Statements

CFScript supports the following statements:

assignment	for-in	try-catch
function call	while	function (function definition)
if-else	do-while	var (in custom functions only)
switch-case-default	break	return (in custom functions only)
for	continue	

© 2014 Adobe Systems Incorporated. All rights reserved.

The following rules apply to statements:

- You must put a semicolon at the end of a statement.
- Line breaks are ignored. A single statement can cross multiple lines.
- White space is ignored. For example, it does not matter whether you precede a semicolon with a space character.
- Use curly brackets to group multiple statements into one logical statement unit.
- Unless otherwise indicated, you can use any ColdFusion expression in the body of a statement.

Note

For information on the `function`, `var`, and `return` statements, see Defining components and functions in CFScript.

Statement blocks

Curly bracket characters ({ and }) group multiple CFScript statements so that they are treated as a single unit or statement.
This syntax enables you to create code blocks in conditional statements, such as the following:

```
if(score GT 0)
{
 result = "positive";
 Positives = Positives + 1;
}
```

In this example, both assignment statements are executed if the score is greater than 0. If they were not in the code block, only the first line would execute.
You do not have to place curly bracket characters on their own lines in the code. For example, you could place the open curly bracket in the preceding example on the same line as the `if` statement, and some programmers use this style. However, putting at least the ending brace on its own line makes it easier to read the code and separate code blocks.

Comments

CFScript has two forms of comments: single line and multiline.
A single-line comment begins with two forward slashes (//) and ends at the line end; for example:

```
//This is a single-line comment.
//This is a second single-line comment.
```

A multiline comment starts with a /* marker and continues until it reaches a */ marker; for example:

```
/*This is a multiline comment.
  You do not need to start each line with a comment indicator.
  This line is the last line in the comment. */
```

© 2014 Adobe Systems Incorporated. All rights reserved.

The following rules apply to comments:

- Comments do not have to start at the beginning of a line. They can follow active code on a line. For example, the following line is valid:

```
MyVariable = 12; // Set MyVariable to the default value.
```

- The end of a multiline comment can be followed on the same line by active code. For example, the following line is valid, although it is poor coding practice:

```
End of my long comment */ foo = "bar";
```

- You can use multiline format for a comment on a single line, for example:

```
/*This is a single-line comment using multiline format. */
```

- You cannot nest /* and */ markers inside other comment lines.
- CFML comments (<!--- and --->) do not work in CFScript.

Differences from JavaScript

Although CFScript and JavaScript are similar, they have several key differences. The following list identifies CFScript features that differ from JavaScript:

- CFScript uses ColdFusion expressions, which are not a superset or a subset of JavaScript expressions. In particular, ColdFusion expressions do not support bitwise operators, and the ColdFusion MOD or % operator operates differently from the corresponding JavaScript % operator: In ColdFusion, the operator does integer arithmetic and ignores fractional parts. ColdFusion expressions also support the EQV, IMP, CONTAINS, and DOES NOT CONTAIN operators that are not supported in JavaScript.
- Variable declarations (`var` keyword) are only used in user-defined functions and threads.
- CFScript is not case sensitive.
- All statements end with a semicolon, and line breaks in the code are ignored.
- Assignments are statements, not expressions, and therefore cannot be used in situations that require evaluating the assignment operation.
- JavaScript objects, such as Window and Document, are not available.
- Only the ColdFusion server processes CFScript. There is no client-side CFScript.

CFScript limitation

You cannot include ColdFusion tags in CFScript. However, you can include `cfscript` blocks inside other ColdFusion tags, such as `cfoutput`.

© 2014 Adobe Systems Incorporated. All rights reserved.

Using CFScript statements

CFScript includes the following types of statements:

- Assignment statements and functions
- Conditional processing statements
- Looping statements

Using assignment statements and functions

CFScript assignment statements are the equivalent of the `cfset` tag. These statements have the following form:

```
lval = expression;
```

eval is any ColdFusion variable reference; for example:

```
x = "positive";
y = x;
a[3]=5;
structure.member=10;
ArrayCopy=myArray;
```

You can use ColdFusion function calls, including UDFs, directly in CFScript. For example, the following line is a valid CFScript statement:

```
StructInsert(employee,"lastname",FORM.lastname);
```

Using conditional processing statements

CFScript includes the following conditional processing statements:

- `if` and `else` statements, which serve the same purpose as the `cfif`, `cfelseif`, and `cfelse` tags
- `switch`, `case`, and `default` statements, which are the equivalents of the `cfswitch`, `cfcase`, and `cfdefaultcase` tags

Using if and else statements

The `if` and `else` statements have the following syntax:

```
if(expr) statement [else statement]
```

In its simplest form, an if statement looks as follows:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
if(value EQ 2700)
 message = "You've reached the maximum";
```

A simple if-else statement looks like the following:

```
if(score GT 1)
 result = "positive";
else
 result = "negative";
```

CFScript does not include an elseif statement. However, you can use an `if` statement immediately after an `else` st atement to create the equivalent of a `cfelseif` tag, as the following example shows:

```
if(score GT 1)
 result = "positive";
else if(score EQ 0)
 result = "zero";
else
 result = "negative";
```

As with all conditional processing statements, you can use curly brackets to enclose multiple statements for each condition, as follows:

```
if(score GT 1) {
 result = "positive";
 message = "The result was positive.";
 }
else {
 result = "negative";
 message = "The result was negative.";
 }
```

Note

Often, you can make your code clearer by using braces even where they are not required.

Using switch and case statements

The `switch` statement and its dependent `case` and `default` statements have the following syntax:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
switch (expression) {
  case constant: [case constant:]... statement(s) break;
  [case constant: [case constant:]... statement(s) break;]...
  [default: statement(s)] }
```

Use the following rules and recommendations for `switch` statements:

- You cannot mix Boolean and numeric constant values in a `switch` statement.
- Each constant value must be a constant (that is, not a variable, a function, or other expression).
- Multiple `case` *constant:* statements can precede the statement or statements to execute if any of the cases are true. This lets you specify several matches for one code block.
- No two constant values can be the same.
- The statements following the colon in a `case` statement block do not have to be in curly brackets. If a constant value equals the `switch` expression, ColdFusion executes all statements through the `break` state ment.
- The `break` statement at the end of the `case` statement tells ColdFusion to exit the `switch` statement. ColdFusion does not generate an error message if you omit a `break` statement. However, if you omit it, ColdFusion executes all the statements in the following case statement, *even if that case is false*. In nearly all circumstances, this is not what you want to do.
- You can have only one `default` statement in a `switch` statement block. ColdFusion executes the statements in the `default` block if none of the `case` statement constants equals the expression value.
- The `default` statement does not have to follow all `case` statements, but it is good programming practice to do so. If any `case` statements follow the `default` statement, you must end the `default` block code with a b `reak` statement.
- The `default` statement is not required. However, use one if the `case` constants do not include all possible values of the expression.
  The following `switch` statement takes the value of a name variable:

1. If the name is John or Robert, it sets both the male variable and the found variable to True.
2. If the name is Mary, it sets the male variable to False and the found variable to True.
3. Otherwise, it sets the found variable to False.

```
switch(name) {
 case "John": case "Robert":
  male=True;
  found=True;
  break;
 case "Mary":
  male=False;
  found=True;
  break;
 default:
  found=False;
} //end switch
```

Using looping statements

CFScript provides a richer selection of looping constructs than those supplied by CFML tags. It enables you to create efficient looping constructs like those in most programming and scripting languages. CFScript provides the

© 2014 Adobe Systems Incorporated. All rights reserved.

following looping constructs:

- For
- While
- Do-while
- For-in
  CFScript also includes the `continue` and `break` statements that control loop processing.

Using for loops

The for loop has the following format:

```
for (initial-expression; test-expression; final-expression) statement
```

The i_nitial-expression_ and *final-expression* can be one of the following:

- A single assignment expression; for example, x=5 or loop=loop+1
- Any ColdFusion expression; for example, SetVariable("a",a+1)
- Empty
  The *test-expression* can be one of the following:

- Any ColdFusion expression; for example:

```
A LT 5
index LE x
status EQ "not found" AND index LT end
```

- Empty

> **Note**
>
> The test expression is re-evaluated before each repeat of the loop. If code inside the loop changes any part of the test expression, it can affect the number of iterations in the loop.

The *statement* can be a single semicolon terminated statement or a statement block in curly brackets.
When ColdFusion executes a for loop, it does the following:

1. Evaluates the *initial expression.*
2. Evaluates the *test-expression.*
3. If the *test-expression* is False, exits the loop and processing continues following the *statement._If the _test-expression*is True:
   a. Executes the *statement* (or statement block).
   b. Evaluates the *final-expression.*
   c. Returns to Step 2.
   For loops are most commonly used for processing in which an index variable is incremented each time through the loop, but it is not limited to this use.
   The following simple for loop sets each element in a 10-element array with its index number.

© 2014 Adobe Systems Incorporated. All rights reserved.

```
for(index=1;
 index LTE 10;
 index = index + 1)
 a[index]=index;
```

The following, more complex, example demonstrates two features:

- The use of curly brackets to group multiple statements into a single block.
- An empty condition statement. All loop control logic is in the statement block.

```
<cfscript>
strings=ArrayNew(1);
ArraySet(strings, 1, 10, "lock");
strings[5]="key";
indx=0;
for( ; ; ) {
 indx=indx+1;
 if(Find("key",strings[indx],1)) {
  WriteOutput("Found key at " & indx & ".<br>");
  break;
  }
 else if (indx IS ArrayLen(strings)) {
  WriteOutput("Exited at " & indx & ".<br>");
  break;
  }
}
</cfscript>
```

This example shows one important issue that you must remember when creating loops: always ensure that the loop ends. If this example lacked the `else if` statement, and there was no "key" in the array, ColdFusion would loop forever or until a system error occurred; you would have to stop the server to end the loop.
The example also shows two issues with index arithmetic: in this form of loop you must make sure to initialize the index, and keep track of where the index is incremented. In this case, because the index is incremented at the top of the loop, initialize it to 0 so it becomes 1 in the first loop.

Using while loops

The while loop has the following format:

```
while (expression) statement
```

The `while` statement does the following:

1. Evaluates the *expression.*
2. If the *expression*is True, it does the following:
    a. Executes the *statement*, which can be a single semicolon-terminated statement or a statement block in curly brackets.
    b. Returns to step 1.

© 2014 Adobe Systems Incorporated. All rights reserved.

If the *expression* is False, processing continues with the next statement.The following example uses a `while`loop to populate a 10-element array with multiples of five.

```
a = ArrayNew(1);
loop = 1;
while (loop LE 10) {
 a[loop] = loop * 5;
 loop = loop + 1;
 }
```

As with other loops, make sure that at some point the{{ while}} *expression* is False and be careful to check your index arithmetic.

Using do-while loops

The do-while loop is like a while loop, except that it tests the loop condition after executing the loop statement block. The do-while loop has the following format:

```
do statement while (expression);
```

The `do while` statement does the following:

1. Executes the *statement*, which can be a single semicolon-terminated statement or a statement block in curly brackets.
2. Evaluates the *expression.*
3. If the *expression* is true, it returns to step 1.
   If the *expression* is False, processing continues with the next statement.
   The following example, like the while loop example, populates a 10-element array with multiples of 5:

```
a = ArrayNew(1);
loop = 1;
do {
a[loop] = loop * 5;
loop = loop + 1;
}
while (loop LE 10);
```

Because the loop index increment follows the array value assignment, the example initializes the loop variable to 1 and tests to make sure that it is less than or equal to 10.
The following example generates the same results as the previous two examples, but it increments the index before assigning the array value. As a result, it initializes the index to 0, and the end condition tests that the index is less than 10.

© 2014 Adobe Systems Incorporated. All rights reserved.

```
a = ArrayNew(1);
loop = 0;
do {
 loop = loop + 1;
 a[loop] = loop * 5;
}
while (loop LT 10);
```

The following example loops through a query:

```
<cfquery ... name="myQuery">
... sql goes here...
</cfquery>
<cfscript>
if (myQuery.RecordCount gt 0) {
  currRow=1;
  do {
    theValue=myQuery.myField[CurrRow];
    currRow=currRow+1;
  } while (currRow LTE myQuery.RecordCount);
}
</cfscript>
```

Using for-in loops

The for-in loop loops over the elements in a ColdFusion structure. It has the following format:

```
for (variable in structure) statement
```

The *variable* can be any ColdFusion identifier; it holds each structure key name as ColdFusion loops through the structure. The *structure* must be the name of an existing ColdFusion structure. The *statement* can be a single semicolon terminated statement or a statement block in reference.
The following example creates a structure with three elements. It then loops through the structure and displays the name and value of each key. Although the curly brackets are not required here, they make it easier to determine the contents of the relatively long `WriteOutput` function. In general, you can make structured control flow, especially loops, clearer by using curly brackets.

```
myStruct=StructNew();
myStruct.productName="kumquat";
mystruct.quality="fine";
myStruct.quantity=25;
for (keyName in myStruct) {
 WriteOutput("myStruct." & Keyname & " has the value: " &
   myStruct[keyName] &"<br>");
}
```

© 2014 Adobe Systems Incorporated. All rights reserved.

> **Note**
>
> Unlike the `cfloop` tag, CFScript for-in loops do not provide built-in support for looping over queries and lists.

Using continue and break statements

The continue and break statements enable you to control the processing inside loops:

- The `continue` statement tells ColdFusion to skip to the beginning of the next loop iteration.
- The `break` statement exits the current loop or `case` statement.

Using continue

The `continue` statement ends the current loop iteration, skips any code following it in the loop, and jumps to the beginning of the next loop iteration. For example, the following code loops through an array and display's each value that is not an empty string:

```
for ( loop=1; loop LE 10; loop = loop+1) {
 if(a[loop] EQ "") continue;
 WriteOutput(loop);
}
```

(To test this code snippet, you must first create an array, a, with 10 or more elements, some of which are not empty strings.)
The `continue` statement is useful if you loop over arrays or structures and you want to skip processing for array elements or structure members with specific values, such as the empty string.

Using break

The `break` statement exits the current loop or `case` statement. Processing continues at the next CFScript statement. You end `case` statement processing blocks with a `break` statement. You can also use a test case with a `break` statement to prevent infinite loops, as shown in the following example. This script loops through an array and prints the array indexes that contain the value key. It uses a conditional test and a `break` statement to make sure that the loop ends when at the end of the array.

© 2014 Adobe Systems Incorporated. All rights reserved.

```
strings=ArrayNew(1);
ArraySet(strings, 1, 10, "lock");
strings[5]="key";
strings[9]="key";
indx=0;
for( ; ; ) {
 indx=indx+1;
 if(Find("key",strings[indx],1)) {
  WriteOutput("Found a key at " & indx & ".<br>");
  }
 else if (indx IS ArrayLen(strings)) {
  WriteOutput("Array ends at index " & indx & ".<br>");
  break;
 }
}
```

for-in construct (for arrays)

You can loop over arrays in CFScript using for-in construct.

h7. Example

```
public String foo(array a)
{
    for(var item in a)
    {
      writedump(item);
    }
}
```

for-in construct (for query)

Similar to looping over array and struct, using for-in construct, you can loop over query object in CFScript.

h7. Example
In this example, query resultset is available in the variable `arts` and the for-in loop is used to loop over the resultset. The variable `row` used in the for-in construct is a struct that contains query columns as keys. You can use `arts.currentrow` to reference the current row.

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cfquery name="arts" datasource="cfartgallery">
 select * from art
</cfquery>
<cfscript>
 cols = listToArray(listsort(arts.columnlist, "textnocase"));
 for(row in arts)
 {
  for(col in cols)
   writeoutput(arts.currentrow & " ..." & col & ": " & row[col] & "<br>");
  writeoutput("<hr>");
 }
</cfscript>
```

Note

You have to prefix queryname to access the query result variables such as `recordcount` or `currentrow` (as shown in the example).

var declaration within for loop

Note

This feature applies only if you have installed ColdFusion 9 Update 1.

You can use `var` inline with for-in construct to bind variable to the local scope for both structs and arrays.

h7. Example

```
public String foo(struct s)
{
    for(var item in s)
    {
      writedump(item & ": " & s[item]);
    }
    writedump(local);
}
```

For arrays example, see for-in construct (for arrays).

For-in constructs support Java arrays

Apart from native ColdFusion arrays, for-in constructs now support Java arrays as shown in the following example:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cfscript>
                a =
CreateObject("java","java.util.Arrays").AsList(ToString("CF,10,Zeus").split(","));
                for (var1 in a) {
                                WriteOutput(var1);
                }
</cfscript>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

Defining components and functions in CFScript

ColdFusion supports the syntax for defining CFCs, including interfaces, functions, properties, and parameters entirely in CFScript. Currently, however, only certain ColdFusion tags are supported as CFScript functions. This section describes the component definition syntax.
For information on tags as functions see *Tag equivalents in CFScript* in Language Enhancements in ColdFusion 9.

Basic Syntax

Syntax for defining a component is as follows:

```
/**
* ColdFusion treats plain comment text as a hint.
* You can also use the @hint metadata name for hints.
* Set metadata, including, optionally, attributes, (including custom
* attributes) in the last entries in the comment block, as follows:
*@metadataName metadataValue
...
*/
component attributeName="attributeValue" ... {
body contents
}
```

The following example shows a simple component definition

```
/**
* Simple Component.
*/
component {
/**
* Simple function.
*/
public void function foo() {
WriteOutput("Method foo() called<br>");
}
}
```

When you define a component entirely in CFScript, you do not have to use a `cfscript` tag on the page. In this case, the component keyword can be preceded only by comments (including metadata assignments) and import operators. Adobe recommends this format as a best practice. You specify component properties as follows:

```
/**
/*@default defaultValue
* @attrib1Name attrib1Value
* ...
*/
property [type]propName;
```

If the type precedes the property name, you do not need to use the "type" keyword, only the name of the specific

© 2014 Adobe Systems Incorporated. All rights reserved.

type. In either format, you must set the name attribute value. All other property attributes, such as type, are optional. As with cfproperty tags, place the property operators at the top of the component definition, immediately following the opening brace.
The syntax to define a function is similar to the component definition:

```
/**
*Comment text, treated as a hint.
*Set metadata, including, optionally, attributes, in the last entries
*in the comment block, as follows:
*@metadataName metadataValue
...
*/
access returnType function functionName(arg1Type arg1Name="defaultValue1"
arg1Attribute="attributeValue...,arg2Type
arg2Name="defaultValue2" arg2Attribute="attributeValue...,...)
functionAttributeName="attributeValue" ... {
body contents
}
```

You specify all function arguments, including the argument type, default value, and attributes in the function definition.
The following example shows a function definition:

```
/**
* @hint "This function displays its name and parameter."
*/
public void function foo(String n1=10)
description="does nothing" hint="overrides hint" {
WriteOutput("Method foo() called<br> Parameter value is " & n1);
}
```

Specifying the `required` keyword makes the argument mandatory. If the `required` keyword is not present then the argument becomes optional.
For example:

```
public function funcname(required string argument1)
```

Interface definitions follow the same pattern as components, with the same general rules and limitations that apply to the interfaces you define using cfinterface tags. The following simple code defines an interface with a single function that takes one string argument, with a default argument value of "Hello World!":

```
interface {
function method1(string arg1="Hello World!");
function method2 (string arg1="Goodbye World!");
...
}
```

© 2014 Adobe Systems Incorporated. All rights reserved.

The following example shows the definition of a simple component with a single function:

```
/**
* Component defined in CFScript
* @output true
*/
component extends="component_01" {
/**
* Function that displays its arguments and returns a string.
* @returnType string
*/
public function method_03(argOne,argTwo) {
WriteOutput("#arguments.argOne# ");
WriteOutput("#arguments.argTwo# ");
return("Arguments written.");
}
}
```

Setting attributes

The definition syntax provides two ways to set attributes:

- At the end of a comment that immediately precedes the element, in the following format

```
/**
*Comment
*@attributeName1 attributeValue
*@attributeName2 attributeValue
*...
*/
```

- In the element declaration using standard attribute-value assignment notation, as in the following line:

```
component extends="component_01"
```

Attribute values set in the element declaration take precedence over the values set in the comment section. Therefore, if you set an attribute, such as a hint in both locations, ColdFusion ignores the value in the comment section and uses only the one in the element declaration.

Specifying page encoding

You can specify the character encoding of a component by specifying a `pageencoding` processing directive at the top of the component body. Character encoding using `pageencoding` processing directive is not supported for CFM files. The following code snippet shows how to use the directive:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
// this is a component
/**
*@hint "this is a hint for component"
*/
component displayname="My Component" {
pageencoding "Cp1252" ;
//
// The rest of the component definition goes here.
//
}
```

Note that as of ColdFusion 11, one should no longer need to use the pageencoding statement to specify the page encoding as the ColdFusion server should be able to identify it automatically.

Note

Currently, you cannot use CFScript to specify the suppresswhitespace processing directive.

Accessing component metadata

To access metadata of a component, function, parameter, or property, use the GetMetadata function. Component metadata includes the metadata of its properties and functions, including attributes and function parameters.
For detailed information about the structure and contents of the metadata, see GetMetaData in the CFML Reference.
The following trivial code shows the use of component metadata:

```
//Create an instance of a component.
theComponent=createObject("Component" "myComponent");
// Get the component metadata.
theMetadata = getMetadata(theComponent);
// The component properties are in an array. Display the name
// of the first property in the array.
writeoutput("Property name: " & theMetadata.properties[1].name);
```

Support for creating custom metadata

Note

To use this feature, you must install ColdFusion 9 Update 1.

You can specify custom metadata for function arguments in script syntax in either of the following ways:

- With arguments, as space-separated list of key-value pairs.
- In annotations, using `@arg1.custommetadata "custom value"`.

Example

custom.cfm

© 2014 Adobe Systems Incorporated. All rights reserved.

```
cfscript>
                writeoutput(new custom().foo(10));
</cfscript>
```

custom.cfc

```
/**
 * custom metadata for a cfc defined using annotation as well as key-value pairs
 * @cfcMetadata1 "cfc metadata1"
 */
component cfcMetadata2 = "cfc metadata2"
{
                /**
                 * custom metadata for a property defined using annotation as well
as key-value pairs
                 * @propMetadata1 "property metadata1"
                 */
                property type="numeric" name="age" default="10"
propMetadata2="property metadata2";

                /**
                 * custom metadata for a function/argument using both annotation and
key-value pairs
                 * @arg1.argmdata1 "arg metadata1"
                 * output true
                 * @fnMetadata1 "function metadata1"
                 */
                public string function foo(required numeric arg1=20 argmdata2="arg
metadata2") fnMetadata2="function metadata2"
                {
                                writedump(getmetadata(this));
                                return arg1;
                }
}
```

Explicit function-local scope

ColdFusion has an explicit function-local scope. Variables in this scope exist only during the execution of the function and are available only to the function. To declare a function-local scope variable either specify the Local scope name when assigning the variable, or use the var keyword. Also, you can now use the var keyword anywhere in a function definition, not just at the top.

> **Note**
>
> Because it is now a scope name, do not use local as a variable or argument name. If you do so, ColdFusion ignores the variable or argument.

The following code shows the use of the function local scope:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cffunction name="foo" output="true">
 <cfset var x = 5>
 <cfset local.y=local.x*4>
 <cfset var z=[local.x,local.y]>
 <cfset local.u.v={z="2"}>
 <cfset zz="in Variables Scope">
 <cfdump var="#local#">
</cffunction>;
```

For more information about function local scope see Using ColdFusion Variables.

Using system level functions

In the `<cfscript>` mode, you can now use the following basic language constructs:

- throw
- writedump
- writelog
- location
- trace
  You can call these functions by passing the argument as name="value" pairs or positional notations. For positional notations, the sequence of arguments must be followed. The sequence of arguments for each construct is mentioned in the *CFML Reference Guide*.
  Example of passing arguments as name=value pair:

```
<cfscript>
 writedump(var=myquery, label="query", show="name", format = "text");
</cfscript>
```

Example of passing arguments as Positional arguments:

```
<cfscript>
 writedump(myquery, "", "html", true)
</cfscript>
```

You do not need to specify all the parameters while using positional arguments. For instance, if you want to specify the first and third argument, you can add an empty string for the second argument. The exception to this usage is when there is a boolean type for the second argument where you have to specify `true` or `false`.

Note

You cannot mix positional and named arguments while calling a function. For example, if you need to use only the `var` and `output` attributes with the `writedump` construct, you can use `writedump(myquery,"browser")`.

© 2014 Adobe Systems Incorporated. All rights reserved.

Import and new operations using cfimport

CFScript supports import and new operations. "New" is now a keyword. However, it is not a reserved word, so you can use it as a variable name.
You can use the cfimport tag or import script operator to import a CFC. The import operation puts the contents of the specified component in the current name space, and caches the resolved component path in memory. The import action is effective on the current page only. If you import CFCs in Application.cfm, the CFC is not imported on other pages of the application.
You refer to the imported component directly without using a dot-delimited pathname. Execution time for cached components is faster than with CFCs that you do not import.

> **Note**
>
> The cfobject and cfobject tags and the CreateObject function also cache the resolved component path. They do not, however, invoke an initializer function.

In script the import statement has the following syntax:

```
import "cfc_filepath"
```

Quotation marks are optional for most paths. Surround the path in quotation marks if any directory or the CFC name has a hyphen.
The cfimport tag now supports importing CFCs and takes a path attribute to specify the path to the CF file to import. Use the import function or cfimport tag with a path attribute on top of the page only. Using them elsewhere has the same effect as putting them on top of the page. Therefore, standard coding practice places the import tags or operators at the top of the file. The cfimport tag can precede a cfcomponent tag. The import CFScript statement must follow the component statement.
The ColdFusion Administrator Sever Settings > Caching page now has a Component Cache option, and a Clear Component Cache button. To prevent ColdFusion from caching resolved component paths, clear the Component Cache option. Click the Clear Component Cache button to remove any resolved component paths from the cache.

> **Note**
>
> In all cases, ColdFusion automatically imports the com.adobe.coldfusion.* name space for CFCs. You do not have to import this path explicitly.

The new operator creates an instance of a CFC. It is equivalent to the cfobject tags and CreateObject function. You can use new as a CFScript operator, or in assignment statements outside a CFScript block, such as in a cfset tag. ColdFusion does not have a corresponding cfnew tag.
The new operation has the following syntax:

```
cfObject=new cfcPath(constructorParam1,...)
```

or

```
cfObject=new cfcPath(arg1=constructorParam1Value,...)
```

© 2014 Adobe Systems Incorporated. All rights reserved.

If the folder name or CFC name has hyphen, use the following syntax:

```
cfObject=new "cfc-path"(constructorParam1,...)
```

If you use the import operator to import the directory that contains the CFC, the cfcPath value is the CFC filename. The constructor parameters can be positional or in name="value" format. When you use the new operator, ColdFusion does the following:

1. Looks for an initmethod constructor method in the CFC. If found, ColdFusion instantiates the component and runs initmethod.
2. If it does not find an initmethod constructor method, it looks for an init constructor method. If found, ColdFusion instantiates the component and runs initmethod.
3. If neither method exists, the new operation instantiates the component but does not call a constructor.

Note

Only the new operator automatically invokes the `initmethod` or `init` function. The new operator returns the value returned by `init` or `initmethod` and if the return is void it returns the instance of the CFC. The `cfobject` tags and the `CreateObject` function do not invoke the function and you must explicitly call any custom initialization code.

© 2014 Adobe Systems Incorporated. All rights reserved.

Handling exceptions

ColdFusion provides two statements for exception handling in CFScript: `try` and `catch`. These statements are equivalent to the CFML `cftry` and `cfcatch` tags.

> **Note**
>
> For a discussion of exception handling in ColdFusion, see Handling Errors.

Exception handling syntax and rules

Exception-handling code in CFScript has the following format:

```
try {
 Code where exceptions will be caught
}
catch(exceptionType exceptionVariable) {
 Code to handle exceptions of type exceptionType
 that occur in the try block
}
...
catch(exceptionTypeN exceptionVariableN) {
 Code to handle exceptions of type
 exceptionTypeN that occur in the try block
}
finally {
 Code that will execute whether there is an exception or not.
}
```

> **Note**
>
> In CFScript, `catch` and `finally` statements follow the `try` block; you do not place them inside the `try` block. This structure differs from that of the `cftry` tag, which must include the `cfcatch` and *`cffinally`* tags in its body.

When you have a `try` statement, you must have a `catch` statement. In the `catch` block, the *exceptionVariable* variable contains the exception type. This variable is the equivalent of the `cfcatch` tag `cfcatch.Type` built-in variable.
The finally block is optional. Its code always runs, and runs after the code in the try block and any catch block.

Exception handling example

The following code shows exception handling in CFScript. It uses a `CreateObject` function to create a Java object. The catch statement executes only if the `CreateObject` function generates an exception. The displayed information includes the exception message; the except.Message variable is the equivalent of calling the Java `getMessage` method on the returned Java exception object. The message in the finally block appears after the catch block message.

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cfscript>
 try {
  emp = CreateObject("Java", "Employees");
 }
 catch(any excpt) {
  WriteOutput("The application was unable to perform a required operation.<br>
 Please try again later.<br>If this problem persists, contact
 Customer Service and include the following information:<br>
    #excpt.Message#<br>");
 }
 finally {
 writeoutput("<br>Thank you for visiting our web site.<br>come back soon!");
 }
</cfscript>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

CFScript example

The following example uses these CFScript features:

- Variable assignment
- Function calls
- For loops
- If-else statements
- `WriteOutput` functions
- Switch statements
  The example uses CFScript without any other ColdFusion tags. It creates a structure of course applicants. This structure contains two arrays; the first has accepted students, the second has rejected students. The script also creates a structure with rejection reasons for some (but not all) rejected students. It then displays the accepted applicants followed by the rejected students and their rejection reasons.

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<html>
<head>
 <title>CFScript Example</title>
</head>
<body>
<cfscript>

 //Set the variables

 acceptedApplicants[1] = "Cora Cardozo";
 acceptedApplicants[2] = "Betty Bethone";
 acceptedApplicants[3] = "Albert Albertson";
 rejectedApplicants[1] = "Erma Erp";
 rejectedApplicants[2] = "David Dalhousie";
 rejectedApplicants[3] = "Franny Farkle";
 applicants.accepted=acceptedApplicants;
 applicants.rejected=rejectedApplicants;

 rejectCode=StructNew();
 rejectCode["David Dalhousie"] = "score";
 rejectCode["Franny Farkle"] = "too late";

 //Sort and display accepted applicants

 ArraySort(applicants.accepted,"text","asc");
 WriteOutput("The following applicants were accepted:<hr>");
 for (j=1;j lte ArrayLen(applicants.accepted);j=j+1) {
  WriteOutput(applicants.accepted[j] & "<br>");
 }
 WriteOutput("<br>");

 //sort and display rejected applicants with reasons information

 ArraySort(applicants.rejected,"text","asc");
 WriteOutput("The following applicants were rejected:<hr>");
 for (j=1;j lte ArrayLen(applicants.rejected);j=j+1) {
  applicant=applicants.rejected[j];
  WriteOutput(applicant & "<br>");
  if (StructKeyExists(rejectCode,applicant)) {
   switch(rejectCode[applicant]) {
    case "score":
     WriteOutput("Reject reason: Score was too low.<br>");
     break;
    case "late":
     WriteOutput("Reject reason: Application was late.<br>");
     break;
    default:
     WriteOutput("Rejected with invalid reason code.<br>");
   } //end switch
  } //end if
  else {
   WriteOutput("Reject reason was not defined.<br>");
  } //end else
  WriteOutput("<br>");
 } //end for
</cfscript>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

Reviewing the code

The following table describes the code:

Code	Description

`<cfscript>`
` //Set the variables`
` acceptedApplicants[1] = "Cora Cardozo";`
` acceptedApplicants[2] = "Betty Bethone";`
` acceptedApplicants[3] = "Albert Albertson";`
` rejectedApplicants[1] = "Erma Erp";`
` rejectedApplicants[2] = "David Dalhousie";`
` rejectedApplicants[3] = "Franny Farkle";`
` applicants.accepted=acceptedApplicants;`
` applicants.rejected=rejectedApplicants;`
 	
` rejectCode=StructNew();`
` rejectCode["David Dalhousie"] = "score";`
` rejectCode["Franny Farkle"] = "too late";`
	Creates two one-dimensional arrays, one with the accepted applicants and another with the rejected applicants. The entries in each array are in random order.Creates a structure and assign each array to an element of the structure.Creates a structure with rejection codes for rejected applicants. The `rejectedCode` structure does not have entries for all rejected applicants, and one of its values does not match a valid code. The structure element references use associative array notation in order to use key names that contain spaces.

`ArraySort(applicants.accepted,"text","asc");` `WriteOutput("The following applicants were accepted:<hr>");` `for (j=1;j lte ArrayLen(applicants.accepted);j=j+1) {` ` WriteOutput(applicants.accepted[j] & " ");` `}` `WriteOutput(" ");`	Sorts the accepted applicants alphabetically.Displays a heading.Loops through the accepted applicants and writes their names. Curly brackets enhance clarity, although they are not needed for a single statement loop.Writes an additional line break at the end of the list of accepted applicants.
`ArraySort(applicants.rejected,"text","asc");` `WriteOutput("The following applicants were rejected:<hr>");`	Sorts `rejectedApplicants` array alphabetically and writes a heading.

© 2014 Adobe Systems Incorporated. All rights reserved.

`for (j=1;j lte ArrayLen(applicants.rejected);j=j+1) {` ` applicant=applicants.rejected[j];` ` WriteOutput(applicant & " ");`	Loops through the rejected applicants.Sets the applicant variable to the applicant name. This makes the code clearer and enables you to easily reference the `rejectCode` array later in the block.Writes the applicant name.
`if (StructKeyExists(rejectCode,applicant)) {` ` switch(rejectCode[applicant]) {` ` case "score":` `  WriteOutput("Reject reason: Score was too low. ");` `  break;` ` case "late":` `  WriteOutput("Reject reason: Application was late. ");` `  break;` ` default:` `  WriteOutput("Rejected with invalid reason code. ");` ` } //end switch` `} //end if`	Checks the `rejectCode` structure for a rejection code for the applicant. If a code exists, enters a switch statement that examines the rejection code value. If the rejection code value matches one of the known codes, displays an expanded explanation of the meaning. Otherwise (the default case), displays an indication that the rejection code is not valid.Comments at the end of blocks help clarify the control flow.
`else {` ` WriteOutput("Reject reason was not defined. ");` `} //end else`	If there is no entry for the applicant in the `rejectCode` structure, displays a message indicating that the reason was not defined.
`WriteOutput(" ");` `} //end for` `</cfscript>`	Displays a blank line after each rejected applicant.Ends the for loop that handles each rejected applicant.Ends the CFScript.

© 2014 Adobe Systems Incorporated. All rights reserved.

Using closures

A closure is an inner function. The inner function can access the variables in the outer function. You can access the inner function by accessing the outer function. See the example below.

```
<cfscript>
function helloTranslator(String helloWord)
 {
  return function(String name)
   {
    return "#helloWord#, #name#";
    };
 }
helloInHindi=helloTranslator("Namaste");
helloInFrench=helloTranslator("Bonjour");
writeoutput(helloInHindi("Anna"));
//closure is formed.
//Prints Namaste, Anna.
writeoutput("<br>");
writeoutput(helloInFrench("John"));
//Prints Bonjour, John.
</cfscript>
```

In the above example, the outer function returns a closure. Using the `helloHindi` variable, the outer function is accessed. It sets the `helloWord` argument. Using this function pointer, the closure is called. For example, `helloInHindi("Anna")`. Observe that even after the execution of outer function, the closure can access the variable sets by the outer function.
In this case, using closure, two new functions are created. One adds `Namaste` to the name. And the second one adds `Bonjour` to the name. `helloInHindi` and `helloInFrench` are closures. They have the same function body; however, store different environments.
The inner function is available for execution after the outer function is returned. A closure is formed when the inner function is available for execution.
As seen in the example, even after the outer function is returned, the inner function can access the variables in the outer function. Closure retains the reference to the environment at the time it is created. For example, the value of a local variable in the outer function. It makes closure an easy to use and handy feature.
To see more details on closure, see http://jibbering.com/faq/notes/closures.

Closure in ColdFusion

A closure can be of the following categories:

- Defined inline without giving a name. They can be used in the following ways:
  - They can be assigned to a variable, array item, struct, and variable scope. It can be returned directly from a function.

    Example

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cfscript>
function operation(string operator)
 {
  return function(numeric x, numeric y)
  {
       if(operator eq "add")
    {
    return x + y;
    }
   else if(operator eq "subtract")
    {
    return x - y;
    }
  };
 }
myval_addition=operation("add");
myval_substraction=operation("subtract");
writeoutput(myval_addition(10,20));
writeoutput("<br>");
writeoutput(myval_substraction(10,20));
</cfscript>
```

In the above example, the outer function sets the operator. `myval_addition` and `myval_substraction` are two closures. They process the data based on the condition sets by the outer function.

- Defined inline as a function and tag argument.**Example**

```
<cfscript>
function operation(numeric x, numeric y, function logic)
 {
  var result=logic(x,y);
  return result;
 }
add = operation(10,20, function(numeric N1, numeric N2)
 {
  return N1+N2;
 });
subtract = operation(10,20, function(numeric N1, numeric N2)
 {
  return N1-N2;
 });
</cfscript>
<cfdump var="#add#">
<cfdump var="#substract#">
```

In the above example, the function `operation` has an argument `logic`, which is a closure. While calling `operation`, an inline closure is passed as an argument. This anonymous closure contains the logic to process the numbers - `addition` or `subtraction`. In this case, the logic is dynamic and passed as a closure to the function.

A Closure can be assigned to a variable

© 2014 Adobe Systems Incorporated. All rights reserved.

You can assign a closure to a variable.
Example

```
var c2 = function () {..}
```

Note

When assigning Closures to a variable, only script style of syntax is supported.

A closure can be used as a return type

You can use a closure as a return type.

Note

As a best practice, if the return type is a closure, provide the Function keyword with initial capitalization.

Example

```
Function function exampleClosure(arg1)
{
 function exampleReturned(innerArg)
 {
  return innerArg + arg1;
 }
 /*
 return a reference to the inner function defined.
 */
return exampleReturned;
}
```

Calling closure with key-value pair

You can call a closure by passing a key-value pair as you do for a function call.
Example

```
var c2 = function(arg1, arg1) {..}
c2(arg1=1, arg2=3);
```

Closure can be assigned to a variable outside function

You can assign a closure to a variable outside the function.
Example

© 2014 Adobe Systems Incorporated. All rights reserved.

```
hello = function (arg1)
{
 writeoutput("Hello " & arg1);
};
hello("Mark");
```

Calling closure with argument collection

Example

```
var c2 = function(arg1, arg1) {..}
argsColl = structNew();
argsColl.arg1= 1;
argsColl.arg2= 3;
c2(argumentCollection = argsColl);
```

Closures and functions

A closure retains a copy of variables visible at the time of its creation. The global variables (like ColdFusion specific scopes) and the local variables (including declaring or outer function's local and arguments scope) are retained at the time of a closure creation. Functions are static.
The following table details the scope of closure based on the way they are defined:

Scenario where closure is defined	Scope
In a CFC function	Closure argument scope, enclosing function local scope and argument scope, this scope, variable scope, and super scope
In a CFM function	Closure argument scope, enclosing function local scope and argument scope, this scope, variable scope, and super scope
As function argument	Closure argument scope, variable scope, and this scope and super scope (if defined in CFC component).

In closure, following is the order of search for an unscoped variable:

1. Closure's local scope
2. Closure's arguments scope
3. Outer function's local scope if available
4. Owner function's local scope if available
5. ColdFusion built-in scope

© 2014 Adobe Systems Incorporated. All rights reserved.

> **Note**
>
> A closure cannot call any user-defined function, because the function's context is not retained, though the closure's context is retained. It gives erroneous results. For example, when a closure is cached, it can be properly called for later use, while a function cannot.

Closure functions

The following are the closure functions:

isClosure

Description

Determines whether a name represents a closure.

Returns

`True`, if name can be called as a closure; `False`, otherwise.

Category

Decision functions

Syntax

```
isClosure(closureName__)
```

See also

Other decision functions.

History

ColdFusion 10: Added this function.

Parameters

Parameter	Description
closureName	Name of a closure. Must not be in quotation marks.Results in an error if not a defined variable or function name.

Usage

Use this function to determine whether the name represents a closure.

Example

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cfscript>
 isClosure(closureName)
 {
  // do something
 }
 else
 {
  // do something
 }
</cfscript>
```

Modifications to the function isCustomFunction

Though closure is a function object, it is not considered as a custom function.
The function now returns:

- `True`: If name can be called as a custom function.
- `False`: If name can be called as a closure.

Usage scenarios

The following scenario explains how you can effectively use ColdFusion closures.

Example - filtering of arrays using closures

The following example filters employees based on location, age, and designation. A single function is used for filtering. The filtering logic is provided to the function as closures. That's filtering logic changes dynamically.

Example

1. Create the `employee.cfc`file that defines the variables.

```
/**
* @name employee
* @displayname ColdFusion Closure Example
* @output false
* @accessors true
*/
component
{
property string Name;
property numeric Age;
property string designation;
property string location;
property string status;
}
```

1. Create the employee array. This CFC also contains the `filterArray() }}function. A closure, {{filter`, is an argument of the function. While accessing this function, the filtering logic is passed as a closure.

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<!---filter.cfc--->
<cfcomponent>
<cfscript>
//Filter the array based on the logic provided by the closure.
function filterArray(Array a, function filter)
 {
  resultarray = arraynew(1);
   for(i=1;i<=ArrayLen(a);i++)
   {
    if(filter(a[i]))
    ArrayAppend(resultarray,a[i]);
   }
  return resultarray;
 }
function getEmployee()
 {
//Create the employee array.
 empArray = Arraynew(1);
 ArrayAppend(empArray,new employee(Name="Ryan", Age=24, designation="Manager",
location="US"));
 ArrayAppend(empArray,new employee(Name="Ben", Age=34, designation="Sr
Manager",  location="US"));
 ArrayAppend(empArray,new employee(Name="Den", Age=24, designation="Software
Engineer", location="US"));
 ArrayAppend(empArray,new employee(Name="Ran", Age=28, designation="Manager",
location="IND"));
 ArrayAppend(empArray,new employee(Name="Ramesh", Age=31,
designation="Software Engineer", location="IND"));
 return empArray;
 }
</cfscript>
</cfcomponent>
```

2. Create the CFM page that accesses the {{filterArray()}}function with a closure which provides the filtering logic. The {{filterArray()}}function is used to filter the employee data in three ways: location, age, and designation. Each time the function is accessed, the filtering logic is changed in the closure.

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<!---arrayFilter.cfm--->
<cfset filteredArray = arraynew(1)>
<cfset componentArray = [3,6,8,2,4,7,9]>
<cfscript>
obj = CreateObject("component", "filter");
// Filters employees from India
filteredArray = obj.filterArray(obj.getEmployee(), function(a)
 {
 if(a.getLocation()=="IND")
  return 1;
 else
  return 0;
 });
writedump(filteredArray);
//Filters employees from india whos age is above thirty
filteredArray = obj.filterArray(obj.getEmployee(), closure(a)
 {
 if((a.getLocation()=="IND") && (a.getAge()>30))
  return 1;
 else
  return 0;
 });
writedump(filteredArray);
// Filters employees who are managers
filteredArray = obj.filterArray( obj.getEmployee(), function(a)
 {
 if((a.getdesignation() contains "Manager"))
  return 1;
 else
  return 0;
 });
writedump(filteredArray);
</cfscript>
```

#back to top

© 2014 Adobe Systems Incorporated. All rights reserved.

Using Regular Expressions in Functions

Regular expressions let you perform string matching operations using Adobe ColdFusion functions; in particular. regular expressions work with the following functions:

- `REFind`
- `REFindNoCase`
- `REMatch`
- `REMatchNoCase`
- `REReplace`
- `REReplaceNoCase`

  Regular expressions used in the `cfinput` and `cftextinput` tags are JavaScript regular expressions, which have a slightly different syntax than ColdFusion regular expressions. For information on JavaScript regular expressions, see Building Dynamic Forms with cfform Tags.

© 2014 Adobe Systems Incorporated. All rights reserved.

About regular expressions

In traditional string matching, as used by the ColdFusion `Find` and `Replace` functions, you provide the string pattern to search for and the string to search. The following example searches a string for the pattern " BIG " and returns a string index if found. The *string index* is the location in the search string where the string pattern begins.

```
<cfset IndexOfOccurrence=Find(" BIG ", "Some BIG string")>
<!--- The value of IndexOfOccurrence is 5 --->
```

You must provide the exact string pattern to match. If the exact pattern is not found, `Find` returns an index of 0. Because you must specify the exact string pattern to match, matches for dynamic data can be difficult, if not impossible, to construct.
The next example uses a regular expression to perform the same search. This example searches for the first occurrence in the search string of any string pattern that consists entirely of uppercase letters enclosed by spaces:

```
<cfset IndexOfOccurrence=REFind(" [A-Z]+ ", "Some BIG string")>
<!--- The value of IndexOfOccurrence is 5 --->
```

The regular expression " A-Z+ " matches any string pattern consisting of a leading space, followed by any number of uppercase letters, followed by a trailing space. Therefore, this regular expression matches the string " BIG " and any string of uppercase letters enclosed in spaces.
By default, the matching of regular expressions is case-sensitive. You can use the `REFindNoCase` and `REReplaceNoCase` functions for case-insensitive matching.
Because you often process large amounts of dynamic textual data, regular expressions are invaluable in writing complex ColdFusion applications.

Using ColdFusion regular expression functions

ColdFusion supplies four functions that work with regular expressions:

- `REFind`
- `REFindNoCase`
- `REMatch`
- `REMatchNoCase`
- `REReplace`
- `REReplaceNoCase`

  `REFind` and `REFindNoCase` use a regular expression to search a string for a pattern and return the string index where it finds the pattern. For example, the following function returns the index of the first instance of the string " BIG ":

```
<cfset IndexOfOccurrence=REFind(" BIG ", "Some BIG BIG string")>
<!--- The value of IndexOfOccurrence is 5 --->
```

To find the next occurrence of the string " BIG ", you must call the `REFind` function a second time. For an example of iterating over a search string to find all occurrences of the regular expression, see Returning matched subexpressions.
`REReplace` and `REReplaceNoCase` use regular expressions to search through a string and replace the string

© 2014 Adobe Systems Incorporated. All rights reserved.

pattern that matches the regular expression with another string. You can use these functions to replace the first match, or to replace all matches.
For detailed descriptions of the ColdFusion functions that use regular expressions, see the *CFML Reference*.

Basic regular expression syntax

The simplest regular expression contains only literal characters. The literal characters must match exactly the text being searched. For example, you can use the regular expression function `REFind` to find the string pattern " BIG ", just as you can with the `Find` function:

```
<cfset IndexOfOccurrence=REFind(" BIG ", "Some BIG string")>
<!--- The value of IndexOfOccurrence is 5 --->
```

In this example, `REFind` must match the exact string pattern " BIG ".
To use the full power of regular expressions, combine literal characters with character sets and special characters, as in the following example:

```
<cfset IndexOfOccurrence=REFind(" [A-Z]+ ", "Some BIG string")>
<!--- The value of IndexOfOccurrence is 5 --->
```

The literal characters of the regular expression consist of the space characters at the beginning and end of the regular expression. The character set consists of that part of the regular expression in brackets. This character set specifies to find a single uppercase letter from A to Z, inclusive. The plus sign + after the brackets is a special character specifying to find one or more occurrences of the character set.
If you removed the + from the regular expression in the previous example, " A-Z " matches a literal space, followed by any single uppercase letter, followed by a single space. This regular expression matches " B " but not " BIG ". The `REFind` function returns 0 for the regular expression, meaning that it did not find a match.
You can construct complicated regular expressions containing literal characters, character sets, and special characters. Like any programming language, the more you work with regular expressions, the more you can accomplish with them. The examples here are fairly basic. For more examples, see Regular expression examples.

© 2014 Adobe Systems Incorporated. All rights reserved.

Regular expression syntax

Regular expression syntax has several basic rules and methods.

Using character sets

The pattern within the brackets of a regular expression defines a character set that is used to match a single character. For example, the regular expression "[A-Za-z] " specifies to match any single uppercase or lowercase letter.

In the character set, a hyphen indicates a range of characters, for example [A-Z] will match any one capital letter.

In a character set a ^ character negates the following characters. For example [^A-Z] matches any single character that is *not* a capital letter.

The regular expression " B[IAU]G " matches the strings "BIG", "BAG", and "BUG", but does not match the string "BOG".

If you specified the regular expression as "B[IA]GN", the concatenation of character sets creates a regular expression that matches the corresponding concatenation of characters in the search string. This regular expression matches a space, followed by "B", followed by an "I" or "A", followed by a "G" or "N". The regular expression matches "BIG", "BAG", "BIN", and "BAN".

The regular expression [A-Z][a-z]* matches any sequence of letters that starts with an uppercase letter and is followed by zero or more lowercase letters. The special character * after the closing square bracket specifies to match zero or more occurrences of the character set.

> **Note**
>
> The * only applies to the character set that immediately precedes it, not to the entire regular expression.

A + after the closing square bracket specifies to find one or more occurrences of the character set. You interpret the regular expression {{"A-Z+" }}as matching one or more uppercase letters enclosed by spaces. Therefore, this regular expression matches " BIG " and also matches " LARGE ", " HUGE ", " ENORMOUS ", and any other string of uppercase letters surrounded by spaces.

Considerations when using special characters

Since a regular expression followed by an * can match zero instances of the regular expression, it can also match the empty string. For example,

```
<cfoutput>
 REReplace("Hello","[T]*","7","ALL") - #REReplace("Hello","[T]*","7","ALL")#<BR>
</cfoutput>
```

results in the following output:

```
REReplace("Hello","[T]*","7","ALL") - 7H7e7l7l7o7
```

© 2014 Adobe Systems Incorporated. All rights reserved.

The regular expression T* can match empty strings. It first matches the empty string before "H" in "Hello". The "ALL" argument tells REReplace to replace all instances of an expression. The empty string before "e" is matched, and so on, until the empty string before "o" is matched.
This result might be unexpected. The workarounds for these types of problems are specific to each case. In some cases you can use [T]+, which requires at least one "T", instead of [T]*. Alternatively, you can specify an additional pattern after [T]*.
In the following examples the regular expression has a "W" at the end:

```
<cfoutput>
 REReplace("Hello World","[T]*W","7","ALL")
  #REReplace("Hello World","[T]*W","7","ALL")#<BR>
</cfoutput>
```

This expression results in the following more predictable output:

```
REReplace("Hello World","[T]*W","7","ALL") - Hello 7orld
```

Finding repeating characters

In some cases, you might want to find
a repeating pattern of characters in a search string. For example,
the regular expression "a{2,4}" specifies to match two to four occurrences
of "a". Therefore, it would match: "aa", "aaa", "aaaa", but not
"a" or "aaaaa". In the following example, the REFind function returns an index of 6:

```
<cfset IndexOfOccurrence=REFind("a{2,4}", "hahahaaahaaaahaaaaahhh")>
<!--- The value of IndexOfOccurrence is 6--->
```

The regular expression "[0-9]{3,}" specifies to match any integer
number containing three or more digits: "123", "45678", and so on.
However, this regular expression does not match a one-digit or two-digit
number.
You use the following syntax to find repeating characters:

1. *m,n*
   Where *m* is 0 or greater and *n* is greater than or equal to *m*. Match *m* through *n* (inclusive) occurrences. The expression {0,1} is equivalent to the special character ?.
2. *m*
   Where *m* is 0 or greater. Match at least *m* occurrences. The syntax {,*n*} is not allowed.The expression {1,} is equivalent to the special character +, and {0,} is equivalent to *.
3. }*m*{ Where *m* is 0 or greater. Match exactly *m* occurrences.

Case sensitivity in regular expressions

ColdFusion supplies case-sensitive and case-insensitive functions for working with regular expressions. REFind an d REReplace perform case-sensitive matching and REFindNoCase and REReplaceNoCase perform case-insensitive matching.

© 2014 Adobe Systems Incorporated. All rights reserved.

You can build a regular expression that models case-insensitive behavior, even when used with a case-sensitive function. To make a regular expression case insensitive, substitute individual characters with character sets. For example, the regular expression [Jj][Aa][Vv][Aa], when used with the case-sensitive functions `REFind` or `REReplace`, matches all of the following string patterns:

- JAVA
- java
- Java
- jAva
- All other combinations of case

Using subexpressions

Parentheses group parts of regular expressions into *subexpressions* that you can treat as a single unit. For example, the regular expression "ha" specifies to match a single occurrence of the string. The regular expression "(ha)+" matches one or more instances of "ha".
In the following example, you use the regular expression "B(ha)+" to match the letter "B" followed by one or more occurrences of the string "ha":

```
<cfset IndexOfOccurrence=REFind("B(ha)+", "hahaBhahahaha")>
<!--- The value of IndexOfOccurrence is 5 --->
```

You can use the special character | in a subexpression to create a logical "OR". You can use the following regular expression to search for the word "jelly" or "jellies":

```
<cfset IndexOfOccurrence=REFind("jell(y|ies)", "I like peanut butter and jelly">
<!--- The value of IndexOfOccurrence is 26--->
```

Using special characters

Regular expressions define the following list of special characters:

```
+ * ? . [ ^ $ ( ) { | \
```

In some cases, you use a special character as a literal character. For example, if you want to search for the plus sign in a string, you have to escape the plus sign by preceding it with a backslash:

```
"\+"
```

The following table describes the special characters for regular expressions:

Special Character	Description

© 2014 Adobe Systems Incorporated. All rights reserved.

	A backslash followed by any special character matches the literal character itself, that is, the backslash escapes the special character.For example, "+" matches the plus sign, and " " matches a backslash.
\.	A period matches any character, including newline. To match any character except a newline, use [^#chr(13)##chr(10)#], which excludes the ASCII carriage return and line feed codes. The corresponding escape codes are \r and \n.
[ ]	A one-character character set that matches any of the characters in that set. For example, "[akm]" matches an "a", "k", or "m". A hyphen in a character set indicates a range of characters; for example, a-z matches any single lowercase letter. If the first character of a character set is the caret (^), the regular expression matches any character except those in the set. It does not match the empty string.For example, akm matches any character except "a", "k", or "m". The caret loses its special meaning if it is not the first character of the set.
^	If the caret is at the beginning of a regular expression, the matched string must be at the beginning of the string being searched.For example, the regular expression "^ColdFusion" matches the string "ColdFusion lets you use regular expressions" but not the string "In ColdFusion, you can use regular expressions." In a character set (ie: within square brackets), a caret character negates the following characters. [^A] matches any character that is *not* an upper case A.
$	If the dollar sign is at the end of a regular expression, the matched string must be at the end of the string being searched.For example, the regular expression "ColdFusion$" matches the string "I like ColdFusion" but not the string "ColdFusion is fun."
?	A character set or subexpression followed by a question mark matches zero or one occurrence of the character set or subexpression. For example, xy?z matches either "xyz" or "xz".
|	The OR character allows a choice between two regular expressions. For example, jell(yies) matches either "jelly" or "jellies".

© 2014 Adobe Systems Incorporated. All rights reserved.

+	A character set or subexpression followed by a plus sign matches one or more occurrences of the character set or subexpression. For example, [a-z]+ matches one or more lowercase characters.
*	A character set or subexpression followed by an asterisk matches zero or more occurrences of the character set or subexpression. For example, [a-z]* matches zero or more lowercase characters.
()	Parentheses group parts of a regular expression into subexpressions that you can treat as a single unit.For example, (ha)+ matches one or more instances of "ha".
(?x)	If at the beginning of a regular expression, it specifies to ignore whitespace in the regular expression and lets you use ## for end-of-line comments. You can match a space by escaping it with a backslash.For example, the following regular expression includes comments, preceded by ##, that are ignored by ColdFusion:`reFind("(?x) one ##first option` {{two ##second option}} {{three\ point\ five ## note escaped spaces}} `", "three point five")`
(?m)	If at the beginning of a regular expression, it specifies the multiline mode for the special characters ^ and $.When used with ^, the matched string can be at the start of the entire search string or at the start of new lines, denoted by a linefeed character or chr(10), within the search string. For $, the matched string can be at the end the search string or at the end of new lines. Multiline mode does not recognize a carriage return, or chr(13), as a new line character. The following example searches for the string "two" across multiple lines: {{#reFind("(?m)^two", "one#chr(10)#two")#}}This example returns 4 to indicate that it matched "two" after the chr(10) linefeed. Without (?m), the regular expression would not match anything, because ^ only matches the start of the string.The character (?m) does not affect \A or \Z, which always match the start or end of the string, respectively. For information on \A and \Z, see Using escape sequences.
(?i)	If at the beginning of a regular expression for `REFind()`, it specifies to perform a case-insensitive compare. For example, the following line would return an index of 1: {{#reFind("(?i)hi", "HI")#}}If you omit the (?i), the line would return an index of zero to signify that it did not find the regular expression.

© 2014 Adobe Systems Incorporated. All rights reserved.

(?=...)	If at the beginning of a regular expression, it specifies to use positive lookahead when searching for the regular expression. If you prefix a subexpression with this, ColdFusion uses positive lookahead for that subexpression. Positive lookahead tests for the parenthesized subexpression like regular parenthesis, but does not include the contents in the match - it merely tests to see if it is there in proximity to the rest of the expression. For example, consider the expression to extract the protocol from a URL: `<cfset regex = "http(?=://)"><cfset string = "http://"><cfset result = reFind(regex, string, 1, "yes")>`{{mid(string, result.pos1, result.len1)}}This example results in the string "http". The lookahead parentheses ensure that the "://" is there, but does not include it in the result. If you did not use lookahead, the result would include the extraneous "://".Lookahead parentheses do not capture text, so backreference numbering will skip over these groups. For more information on backreferencing, see Using backreferences.
(?!...)	If at the beginning of a regular expression, it specifies to use negative lookahead. Negative is just like positive lookahead, as specified by (?=...), except that it tests for the absence of a match.Lookahead parentheses do not capture text, so backreference numbering will skip over these groups. For more information on backreferencing, see Using backreferences.
(?:...)	If you prefix a subexpression with "?:", ColdFusion performs all operations on the subexpression except that it will not capture the corresponding text for use with a back reference.

You must be aware of the following considerations when using special characters in character sets, such as a-z:

- To include a hyphen (-) in the brackets of a character set as a literal character, you cannot escape it as you can other special characters because ColdFusion always interprets a hyphen as a range indicator. Therefore, if you use a literal hyphen in a character set, make it the last character in the set.
- To include a closing square bracket (]) in the character set, escape it with a backslash, as in [1-3]A-z]. You do not have to escape the] character outside the character set designator.

Using escape sequences

Escape sequences are special characters in regular expressions preceded by a backslash (). You typically use escape sequences to represent special characters within a regular expression. For example, the escape sequence \t represents a tab character within the regular expression, and the \d escape sequence specifies any digit, as [0-9] does. ColdFusion escape sequences are case sensitive.
The following table lists the escape sequences that ColdFusion supports:

© 2014 Adobe Systems Incorporated. All rights reserved.

Escape Sequence	Description
\b	Specifies a boundary defined by a transition from an alphanumeric character to a nonalphanumeric character, or from a nonalphanumeric character to an alphanumeric character.For example, the string " Big" contains boundary defined by the space (nonalphanumeric character) and the "B" (alphanumeric character). The following example uses the \b escape sequence in a regular expression to locate the string "Big" at the end of the search string and not the fragment "big" inside the word "ambiguous".reFindNoCase("\bBig\b", "Don't be ambiguous about Big."){{<!--- The value of IndexOfOccurrence is 26 --->}}When used inside a character set (for example [\b]), it specifies a backspace
\B	Specifies a boundary defined by no transition of character type. For example, two alphanumeric characters in a row or two nonalphanumeric characters in a row; opposite of \b.
\A	Specifies a beginning of string anchor, much like the ^ special character.However, unlike ^, you cannot combine \A with (?m) to specify the start of newlines in the search string.
\Z	Specifies an end of string anchor, much like the $ special character.However, unlike $, you cannot combine \Z with (?m) to specify the end of newlines in the search string.
\n	Newline character
\r	Carriage return
\t	Tab
\f	Form feed
\d	Any digit, similar to [0-9]
\D	Any nondigit character, similar to [^0-9]
\w	Any alphanumeric character, or the underscore (_), similar to [[:word:]]

© 2014 Adobe Systems Incorporated. All rights reserved.

\W	Any nonalphanumeric character, except the underscore similar to [^[:word:]]
\s	Any whitespace character including tab, space, newline, carriage return, and form feed. Similar to [\t\n\r\f].
\S	Any nonwhitespace character, similar to [^ \t\n\r\f]
x	A hexadecimal representation of character, where d is a hexadecimal digit
\ddd	An octal representation of a character, where d is an octal digit, in the form \000 to \377

Using character classes

In character sets within regular expressions, you can include a character class. You enclose the character class inside brackets, as the following example shows:

```
REReplace ("Adobe Web Site","[[:space:]]","*","ALL")
```

This code replaces all the spaces with *, producing this string:

```
Adobe*Web*Site
```

You can combine character classes with other expressions within a character set. For example, the regular expression [space:123] searches for a space, 1, 2, or 3. The following example also uses a character class in a regular expression:

```
<cfset IndexOfOccurrence=REFind("[[:space:]][A-Z]+[[:space:]]",
 "Some BIG string")>
<!--- The value of IndexOfOccurrence is 5 --->
```

The following table shows the character classes that ColdFusion supports. Regular expressions using these classes match any Unicode character in the class, not just ASCII or ISO-8859 characters.

Character class	Matches
:alpha:	Any alphabetic character.
:upper:	Any uppercase alphabetic character.

© 2014 Adobe Systems Incorporated. All rights reserved.

:lower:	Any lowercase alphabetic character
:digit:	Any digit. Same as \d.
:alnum:	Any alphabetic or numeric character.
:xdigit:	Any hexadecimal digit. Same as [0-9A-Fa-f].
:blank:	Space or a tab.
:space:	Any whitespace character. Same as \s.
:print:	Any alphanumeric, punctuation, or space character.
:punct:	Any punctuation character
:graph:	Any alphanumeric or punctuation character.
:cntrl:	Any character not part of the character classes [:upper:], [:lower:], [:alpha:], [:digit:], [:punct:], [:graph:], [:print:], or [:xdigit:].
:word:	Any alphabetic or numeric character, plus the underscore (_). Same as \w
:ascii:	The ASCII characters, in the Hexadecimal range 0 - 7F

© 2014 Adobe Systems Incorporated. All rights reserved.

Using backreferences

You use parenthesis to group components of a regular expression into subexpressions. For example, the regular expression "(ha)+" matches one or more occurrences of the string "ha".
ColdFusion performs an additional operation when using subexpressions; it automatically saves the characters in the search string matched by a subexpression for later use within the regular expression. Referencing the saved subexpression text is called *backreferencing*.
You can use backreferencing when searching for repeated words in a string, such as "the the" or "is is". The following example uses backreferencing to find all repeated words in the search string and replace them with an asterisk:

```
REReplace("There is is coffee in the the kitchen",
 "[ ]+([A-Za-z]+)[ ]+\1"," * ","ALL")
```

Using this regular expression, ColdFusion detects the two occurrences of "is" as well as the two occurrences of "the", replaces them with an asterisk enclosed in spaces, and returns the following string:

```
There * coffee in * kitchen
```

You interpret the regular expression [ ]([A-Za-z])[ ]+\1 as follows:
Use the subexpression ([A-Za-z]) to search for character strings consisting of one or more letters, enclosed by one or more spaces, [ ], followed by the same character string that matched the first subexpression, \1.
You reference the matched characters of a subexpression using a slash followed by a digit *n* (*n*) where the first subexpression in a regular expression is referenced as \1, the second as \2, and so on. The next section includes an example using multiple backreferences.

Using backreferences in replacement strings

You can use backreferences in the replacement string of both the `REReplace` and `REReplaceNoCase` functions. For example, to replace the first repeated word in a text string with a single word, use the following syntax:

```
REReplace("There is is a cat in in the kitchen",
 "([A-Za-z ]+)\1","\1")
```

This results in the sentence:

```
"There is a cat in in the kitchen"
```

You can use the optional fourth parameter to `REReplace`, *scope*, to replace all repeated words, as in the following code:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
REReplace("There is is a cat in in the kitchen",
 "([A-Za-z ]+)\1","\1","ALL")
```

This results in the following string:

```
"There is a cat in the kitchen"
```

The next example uses two backreferences to reverse the order of the words "apples" and "pears" in a sentence:

```
<cfset astring = "apples and pears, apples and pears, apples and pears">
<cfset newString = REReplace("#astring#", "(apples) and (pears)",
 "\2 and \1","ALL")>
```

In this example, you reference the subexpression (apples) as \1 and the subexpression (pears) as \2. The `REReplace` function returns the string:

```
"pears and apples, pears and apples, pears and apples"
```

Note

To use backreferences in either the search string or the replace string, you must use parentheses within the regular expression to create the corresponding subexpression. Otherwise, ColdFusion throws an exception.

Using backreferences to perform case conversions in replacement strings

The `REReplace` and `REReplaceNoCase` functions support special characters in replacement strings to convert replacement characters to uppercase or lowercase. The following table describes these special characters:

Special character	Description
\u	Converts the next character to uppercase.
\l	Converts the next character to lowercase.
\U	Converts all characters to uppercase until encountering \E.
\L	Converts all characters to lowercase until encountering \E.

© 2014 Adobe Systems Incorporated. All rights reserved.

\E	End \U or \L.

To include a literal \u, or other code, in a replacement string, escape it with another backslash; for example
u.
For example, the following statement replaces the uppercase string "HELLO" with a lowercase "hello". This example uses backreferences to perform the replacement.

```
REReplace("HELLO", "([[:upper:]]*)", "Don't shout\scream \L\1")
```

The result of this example is the string "Don't shout\scream hello".

Escaping special characters in replacement strings

You use the backslash character, \, to escape backreference and case-conversion characters in replacement strings. For example, to include a literal "\u" in a replacement string, escape it, as in "
\u".

Omitting subexpressions from backreferences

By default, a set of parentheses will both group the subexpression and capture its matched text for later referral by backreferences. However, if you insert "?:" as the first characters of the subexpression, ColdFusion performs all operations on the subexpression except that it will not capture the corresponding text for use with a back reference. This is useful when alternating over subexpressions containing differing numbers of groups would complicate backreference numbering. For example, consider an expression to insert a "Mr." in between Bonjour|Hi|Hello and Bond, using a nested group for alternating between Hi & Hello:

```
<cfset regex = "(Bonjour|H(?:i|ello))( Bond)">
<cfset replaceString = "\1 Mr.\2">
<cfset string = "Hello Bond">
#REReplace(string, regex, replaceString)#
```

This example returns "Hello Mr. Bond". If you did not prohibit the capturing of the Hi/Hello group, the \2 backreference would end up referring to that group instead of " Bond", and the result would be "Hello Mr.ello".

© 2014 Adobe Systems Incorporated. All rights reserved.

Returning matched subexpressions

The `REFind` and `REFindNoCase` functions return the location in the search string of the first match of the regular expression. Even though the search string in the next example contains two matches of the regular expression, the function only returns the index of the first:

```
<cfset IndexOfOccurrence=REFind(" BIG ", "Some BIG BIG string")>
<!--- The value of IndexOfOccurrence is 5 --->
```

To find all instances of the regular expression, you must call the `REFind` and `REFindNoCase` functions multiple times.
Both the `REFind` and `REFindNoCase` functions take an optional third parameter that specifies the starting index in the search string for the search. By default, the starting location is index 1, the beginning of the string.
To find the second instance of the regular expression in this example, you call `REFind` with a starting index of 8:

```
<cfset IndexOfOccurrence=REFind(" BIG ", "Some BIG BIG string", 8)>
<!--- The value of IndexOfOccurrence is 9 --->
```

In this case, the function returns an index of 9, the starting index of the second string " BIG ".
To find the second occurrence of the string, you must know that the first string occurred at index 5 and that the string's length was 5. However, `REFind` only returns starting index of the string, not its length. So, you either must know the length of the matched string to call `REFind` the second time, or you must use subexpressions in the regular expression.
The `REFind` and `REFindNoCase` functions let you get information about matched subexpressions. If you set these functions fourth parameter, `ReturnSubExpression`, to True, the functions return a CFML structure with two arrays, `pos` and `len`, containing the positions and lengths of text strings that match the subexpressions of a regular expression, as the following example shows:

```
<cfset sLenPos=REFind(" BIG ", "Some BIG BIG string", 1, "True")>
<cfoutput>
 <cfdump var="#sLenPos#">
</cfoutput><br>
```

Element one of the `pos` array contains the starting index in the search string of the string that matched the regular expression. Element one of the `len` array contains length of the matched string. For this example, the index of the first " BIG " string is 5 and its length is also 5. If the regular expression does not occur, the `pos` and `len` arrays each contain one element with a value of 0.
You can use the returned information with other string functions, such as `mid`. The following example returns that part of the search string matching the regular expression:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cfset myString="Some BIG BIG string">
<cfset sLenPos=REFind(" BIG ", myString, 1, "True")>
<cfoutput>
 #mid(myString, sLenPos.pos[1], sLenPos.len[1])#
</cfoutput>
```

Each additional element in the `pos` array contains the position of the first match of each subexpression in the search string. Each additional element in `len` contains the length of the subexpression's match.
In the previous example, the regular expression " BIG " contained no subexpressions. Therefore, each array in the structure returned by `REFind` contains a single element.
After executing the previous example, you can call `REFind` a second time to find the second occurrence of the regular expression. This time, you use the information returned by the first call to make the second:

```
<cfset newstart = sLenPos.pos[1] + sLenPos.len[1] - 1>
<!--- subtract 1 because you need to start at the first space --->
<cfset sLenPos2=REFind(" BIG ", "Some BIG BIG string", newstart, "True")>
<cfoutput>
 <cfdump var="#sLenPos2#">
</cfoutput><br>
```

If you include subexpressions in your regular expression, each element of `pos` and `len` after element one contains the position and length of the first occurrence of each subexpression in the search string.
In the following example, the expression [A-Za-z]+ is a subexpression of a regular expression. The first match for the expression ([A-Za-z])[ ], is "is is".

```
<cfset sLenPos=REFind("([A-Za-z]+)[ ]+\1",
 "There is is a cat in in the kitchen", 1, "True")>
<cfoutput>
 <cfdump var="#sLenPos#">
</cfoutput><br>
```

The entries sLenPos.pos[1] and sLenPos.len[1] contain information about the match of the entire regular expression. The array elements sLenPos.pos2 and sLenPos.len2 contain information about the first subexpression ("is"). Because `REFind` returns information on the first regular expression match only, the `sLenPos` structure does not contain information about the second match to the regular expression, "in in".
The regular expression in the following example uses two subexpressions. Therefore, each array in the output structure contains the position and length of the first match of the entire regular expression, the first match of the first subexpression, and the first match of the second subexpression.

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cfset sString = "apples and pears, apples and pears, apples and pears">
<cfset regex = "(apples) and (pears)">
<cfset sLenPos = REFind(regex, sString, 1, "True")>
<cfoutput>
 <cfdump var="#sLenPos#">
</cfoutput>
```

For a full discussion of subexpression usage, see the sections on `REFind` and `REFindNoCase` in the ColdFusion functions chapter in the *CFML Reference*.

Specifying minimal matching

The regular expression quantifiers ?, *, +, {min,} and {min,max} specify one or both of a minimum and maximum number of instances of a given expression to match. By default, ColdFusion locates the greatest number characters in the search string that match the regular expression. This behavior is called *maximal matching*.
For example, you use the regular expression "(\.*)" to search the string "one two". The regular expression "(\.*)", matches both of the following:

- one
- one two
  By default, ColdFusion always tries to match the regular expression to the largest string in the search string. The following code shows the results of this example:

```
<cfset sLenPos=REFind("<b>(.*)</b>", "<b>one</b> <b>two</b>", 1, "True")>
<cfoutput>
 <cfdump var="#sLenPos#">
</cfoutput><br>
```

Thus, the starting position of the string is 1 and its length is 21, which corresponds to the largest of the two possible matches.
However, sometimes you might want to override this default behavior to find the shortest string that matches the regular expression. ColdFusion includes minimal-matching quantifiers that let you specify to match on the smallest string. The following table describes these expressions:

Expression	Description
*?	minimal-matching version of *
+?	minimal-matching version of +
??	minimal-matching version of ?
{min,}?	minimal-matching version of {min,}
{min,max}?	minimal-matching version of {min,max}

© 2014 Adobe Systems Incorporated. All rights reserved.

{n}?	(no different from {n}, supported for notational consistency)

If you modify the previous example to use the minimal-matching syntax, the code is as follows:

```
<cfset sLenPos=REFind("<b>(.*?)</b>", "<b>one</b> <b>two</b>", 1, "True")>
<cfoutput>
 <cfdump var="#sLenPos#">
</cfoutput><br>
```

Thus, the length of the string found by the regular expression is 10, corresponding to the string "one".

© 2014 Adobe Systems Incorporated. All rights reserved.

Regular expression examples

The following examples show some regular expressions and describe what they match:

Expression	Description
`[\?&]value=`	A URL parameter value in a URL.
`[A-Z]:(\\[A-Z0-9_]+)+`	An uppercase DOS/Windows path in which (a) is not the root of a drive, and (b) has only letters, numbers, and underscores in its text.
`^[A-Za-z][A-Za-z0-9_]*`	A ColdFusion variable with no qualifier.
`([A-Za-z][A-Za-z0-9_]*)(\.[A-Za-z][A-Za-z0-9_]*)?`	A ColdFusion variable with no more than one qualifier; for example, Form.VarName, but not Form.Image.VarName.
`(\+|-)?[1-9][0-9]*`	An integer that does not begin with a zero and has an optional sign.
`(\+|-)?[0-9]+(\.[0-9]*)?`	A real number.
`(\+|-)?[1-9]\.[0-9]*E(\+|-)?[0-9]+`	A real number in engineering notation.
a{2,4}	Two to four occurrences of “a”: aa, aaa, aaaa.
(ba){3,}	At least three “ba” pairs: bababa, babababa, and so on.

Regular expressions in CFML

The following examples of CFML show some common uses of regular expression functions:

Expression	Returns
`REReplace (CGI.Query_String, "CFID=[0-9]+[&]*", "")`	The query string with parameter CFID and its numeric value stripped out.
`REReplace("I Love Jellies", "[[:lower:]]","x","ALL"`	I Lxxx Jxxxxxx
`REReplaceNoCase("cabaret","[A-Z]", "G","ALL")`	GGGGGGG
`REReplace (Report,"\$[0-9,]*\.[0-9]*","$***.**")", "")`	The string value of the variable Report with all positive numbers in the dollar format changed to "$***.**".

© 2014 Adobe Systems Incorporated. All rights reserved.

`REFind ("[Uu]\.?[Ss]\.?[Aa}\.?", Report )`	The position in the variable Report of the first occurrence of the abbreviation USA. The letters can be in either case and the abbreviation can have a period after any letter.
`REFindNoCase("a+c","ABCAACCDD")`	4
`REReplace("There is is coffee in the the kitchen","([A-Za-z]+)[ ]+\1","*","ALL")`	There * coffee in * kitchen
`REReplace(report, "<[^>]*>", "", "All")`	Removes all HTML tags from a string value of the report variable.

© 2014 Adobe Systems Incorporated. All rights reserved.

Types of regular expression technologies

Many types of regular expression technologies are available to programmers. JavaScript, Perl, and POSIX are all examples of different regular expression technologies. Each technology has its own syntax specifications and is not necessarily compatible with other technologies.
ColdFusion supports regular expressions that are Perl compliant with a few exceptions:

- A period, ., always matches newlines.
- In replacement strings, use \n instead of $n for backreference variables. ColdFusion escapes all $ in the replacement string.
- You do not have to escape backslashes in replacement strings. ColdFusion escapes them, except in case conversion sequences or escaped versions (for example, \u or \\\\u).
- Embedded modifiers such as (?i) always operate on the entire expression, even if they are inside a group.
- \` and the combinations \u\L and \l\U are not supported in replacement strings.
  The following Perl statements are not supported:

- Lookbehind (?<=) (<?!)
- \\\\x
- \N
- \p
- \C

An excellent reference on regular expressions is *Mastering Regular Expressions*, by Jeffrey E. F. Friedl, O'Reilly & Associates, Inc., 1997, ISBN: 1-56592-257-3, available at www.oreilly.com.

© 2014 Adobe Systems Incorporated. All rights reserved.

ColdFusion Language Enhancements

- Script support for tags
- Using custom tags in scripts
- Query tags as ColdFusion functions
- JSON serialization
- Member functions for data structure
- Support for Elvis operator (?:)
- Passing array index to callback functions in ArrayEach
- Support for ListEach
- Support for QueryGetRow
- Preferences for built-in functions
- Support for database queries

ColdFusion has gone through various language enhancements that will provide a better development experience for ColdFusion developers. The core CFML language enhancements in ColdFusion 11 includes new language constructs, extended tag support, enhanced script functions, and support for new operations.

This document provides an overview on the language enhancements and changes made in ColdFusion 11.

Script support for tags

See Script support for tags.

Adobe blog post
Script support for tags

Using custom tags in scripts

See Using custom tags in scripts.

Query tags as ColdFusion functions

See QueryExecute.

JSON serialization

JSON serialization allows you to convert ColdFusion data into a JSON (JavaScript Object Notation) representation of the data. This feature was made available in ColdFusion 8.

See SerializeJSON.

Adobe blog post
Improved JSON Serialization

Member functions for data structure

See Using the member functions.

© 2014 Adobe Systems Incorporated. All rights reserved.

Support for Elvis operator (?:)

See Elvis operator.

Passing array index to callback functions in ArrayEach

Currently, ColdFusion supports passing objects in callback function. Now, from ColdFusion 11, you can pass the index of an array to the callback function. See ArrayEach.

Support for ListEach

A new function, ListEach, has been introduced. See ListEach.

Support for QueryGetRow

A new function, QueryGetRow, has been introduced. See QueryGetRow.

Preferences for built-in functions

See Built-in functions as first class citizen.

Adobe blog post
Language Enhancements in ColdFusion Splendor - Promoting built-in CF function to first class

Support for database queries

You can start using the <cfquery> tag in client-side CFML just like how you are currently using it in server-side CFML code. Note that not all of the <cfquery> features are supported in this release.

See Mobile Application Development

Added script support for cfimport based prefix custom tag. Now you can use prefix based custom tag in script block without any issue.

```
test.cfm
<cfscript>
  cfimport(taglib=" ../importFolder/" prefix="myTags");
  myTags:customTag();
</cfscript>

customTag.cfm
<cfoutput > Output from custom tag </cfoutput>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

Built-in functions as first class citizen

The ColdFusion built-in functions will be treated as ‘first-class’ functions so that any built-in function can be passed as an argument.

For instance, this is valid:

```
<cfscript>
 function convertCaseForArray(Array array, function convertor)
 {
  for (var i=1; i <= arrayLen(array); i++){
   array[i] = convertor(array[i]);
  }
  return array;
 }

 // lcase built-in function is being passed as callback.
 resultantArray = convertCaseForArray(['One', 'Two','Three'], lcase);

 writedump(resultantArray);
</cfscript>
```

Now, you can treat the built-in CFML functions like ucase() as objects, being able to assign them to variables, and pass them as arguments.

© 2014 Adobe Systems Incorporated. All rights reserved.

Building Blocks of ColdFusion Applications

To build robust ColdFusion Applications, you can create ColdFusion Elements, write and call user-defined Functions, create or use the custom CFML tags and/or build using the custom CFXAPI tags.

- Creating ColdFusion Elements
- Writing and Calling User-Defined Functions
- Building and Using ColdFusion Components
- Creating and Using Custom CFML Tags
- Building Custom CFXAPI Tags
- Using the member functions

#back to top

© 2014 Adobe Systems Incorporated. All rights reserved.

Creating ColdFusion Elements

You can create ColdFusion elements to organize your code. When you create any of these elements, you write your code once and use it, without copying it, in many places.

© 2014 Adobe Systems Incorporated. All rights reserved.

About CFML elements that you create

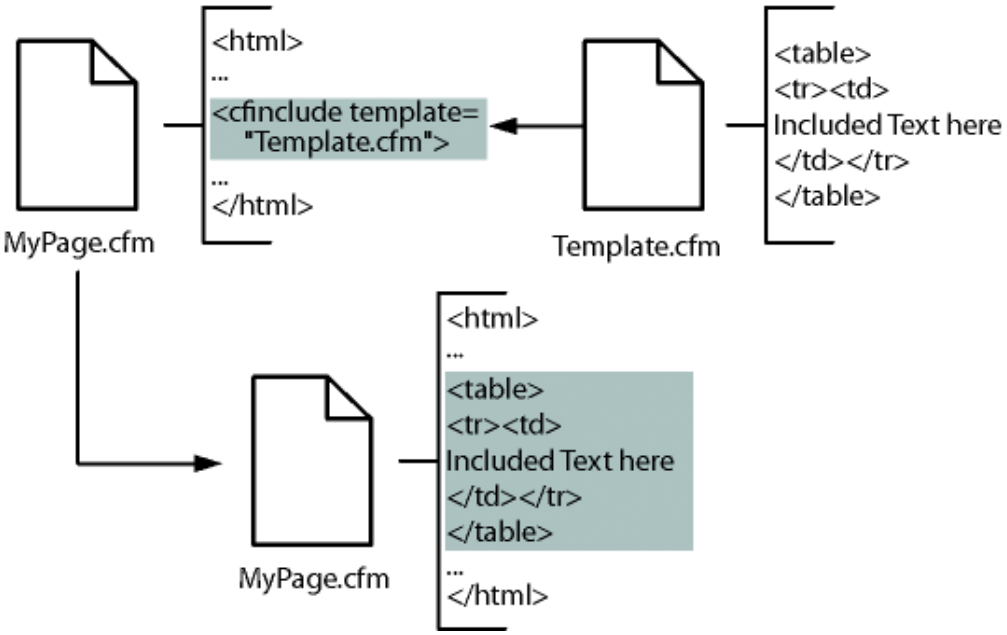
Adobe ColdFusion provides you with several techniques and elements to create sections of code that you can use multiple times in an application. Many of the elements also let you extend the built-in capabilities of ColdFusion. ColdFusion provides the following techniques and elements:

- ColdFusion pages you include using the `cfinclude` tag
- User-defined functions (UDFs)
- ColdFusion components
- Custom CFML tags
- CFX (ColdFusion Extension) tags
  ColdFusion can also use elements developed using other technologies, including the following:
- JSP tags from JSP tag libraries. For information on using JSP tags, see Integrating JEE and Java Elements in CFML Applications.
- Java objects, including objects in the Java run-time environment and JavaBeans. For information on using Java objects, see Integrating JEE and Java Elements in CFML Applications.
- Microsoft COM (Component Object Model) objects. For information on using COM objects, see Integrating COM and CORBA Objects in CFML Applications.
- CORBA (Common Object Request Broker Architecture) objects. For information on using CORBA objects, see Integrating COM and CORBA Objects in CFML Applications.
- Web services. For information on using web services, see Using Web Services.

© 2014 Adobe Systems Incorporated. All rights reserved.

Including pages with the cfinclude tag

The `cfinclude` tag adds the contents of a ColdFusion page to another ColdFusion page, as if the code on the included page was part of the page that uses the `cfinclude` tag. It lets you pursue a "write once use multiple times" strategy for ColdFusion elements that you incorporate in multiple pages. Instead of copying and maintaining the same code on multiple pages, you can store the code in one page and then reference it in many pages. For example, the `cfinclude` tag is commonly used to place a header and footer on multiple pages. This way, if you change the header or footer design, you only change the contents of a single file.
The model of an included page is that it is part of your page; it just resides in a separate file. The `cfinclude` tag cannot pass parameters to the included page, but the included page has access to all the variables on the page that includes it. The following image shows this model:



Using the cfinclude tag

When you use the `cfinclude` tag to include one ColdFusion page in another ColdFusion page, the page that includes another page is referred to as the *calling* page. When ColdFusion encounters a `cfinclude` tag it replaces the tag on the calling page with the output from processing the included page. The included page can also set variables in the calling page.
The following line shows a sample `cfinclude` tag:

```
<cfinclude template = "header.cfm">
```

> **Note**
>
> You cannot break CFML code blocks across pages. For example, if you open a `cfoutput` block in a ColdFusion page, close the block on the same page; you cannot include the closing portion of the block in an included page.

ColdFusion searches for included files as follows:

- The `template` attribute specifies a path relative to the directory of the calling page.
- If the template value is prefixed with a forward slash, ColdFusion searches for the included file in directories

© 2014 Adobe Systems Incorporated. All rights reserved.

that you specify on the Mappings page of the ColdFusion Administrator.

> A page must not include itself. Doing so causes an infinite processing loop. To resolve the problem, stop the ColdFusion_ server._

Include code in a calling page

1. Create a ColdFusion page named header.cfm that displays your logo. Your page can consist of just the following lines, or it can include many lines to define an entire header:

```
<img src="mylogo.gif">
<br>
```

   (For this example to work, you must also place your logo as a GIF file in the same directory as the header.cfm file.)

2. Create a ColdFusion page with the following content:

```
<html>
<head>
 <title>Test for Include</title>
</head>
<body>
 <cfinclude template="header.cfm">
</body>
</html>
```

3. Save the file as includeheader.cfm and view it in a browser.
   The header appears along with the logo.

Recommended uses

Consider using the `cfinclude` tag in the following cases:

- For page headers and footers
- To divide a large page into multiple logical chunks that are easier to understand and manage
- For large "snippets" of code that are used in many places but do not require parameters or fit into the model of a function or tag

© 2014 Adobe Systems Incorporated. All rights reserved.

User-defined functions with ColdFusion Elements

User-defined functions (UDFs) let you create application elements in a format in which you pass in arguments and get a return a value. You can define UDFs using CFScript or the `cffunction` tag. The two techniques have several differences, of which the following are the most important:

- If you use the `cffunction` tag, your function can include CFML tags.
- If you write your function using CFScript, you cannot include CFML tags.
  You can use UDFs in your application pages just as you use standard ColdFusion functions. When you create a function for an algorithm or procedure that you use frequently, you can then use the function wherever you need the procedure, just as you would use a ColdFusion built-in function. For example, the following line calls the function MyFunct and passes it two arguments:

```
<cfset returnValue=MyFunct(Arg1, Arg2)>
```

You can group related functions in a ColdFusion component. For more information, see Using ColdFusion components.
As with custom tags, you can easily distribute UDFs to others. For example, the Common Function Library Project at www.cflib.org is an open-source collection of CFML user-defined functions.

Recommended uses

Typical uses of UDFs include, but are not limited to, the following:

- Data manipulation routines, such as a function to reverse an array
- String and date and time routines, such as a function to determine whether a string is a valid IP address
- Mathematical calculation routines, including standard trigonometric and statistical operations or calculating loan amortization
- Routines that call functions externally, for example using COM or CORBA, such as routines to determine the space available on a Windows file system drive
  Consider using UDFs in the following circumstances:

- You want to pass in arguments, process the results, and return a value. UDFs can return complex values, including structures that contain multiple simple values.
- You want to provide logical units, such as data manipulation functions.
- Your code must be recursive.
- You distribute your code to others.
  If you can create either a UDF or a custom CFML tag for a particular purpose, first consider creating a UDF because running it requires less system overhead than using a custom tag.

For more information

For more information on user-defined functions, see Writing and Calling User-Defined Functions.

© 2014 Adobe Systems Incorporated. All rights reserved.

About user-defined functions-Developing guide

You can create your own custom functions, known as *user-defined functions*, or UDFs. You then use them in your application pages the same way you use standard ColdFusion functions. You can also organize functions you create by grouping related functions into ColdFusion components. For more information, see Building and Using ColdFusion Components.

When you create a function for an algorithm or procedure that you use frequently, you can then use the function wherever you require the procedure. If you must change the procedure, you change only one piece of code. You can use your function anywhere that you can use a ColdFusion expression: in tag attributes, between number (#) signs in output, and in CFScript code. Typical uses of UDFs include, but are not limited to the following:

- Data manipulation routines, such as a function to reverse an array
- String and date/time routines, such as a function to determine whether a string is a valid IP address
- Mathematical calculation routines, including standard trigonometric and statistical operations or calculating loan amortization
- Routines that call functions externally, for example using COM or CORBA, including routines to determine the space available on a Windows file system drive

  For information about selecting among user-defined functions, ColdFusion components, and custom tags, see Creating ColdFusion Elements.

Note

The Common Function Library Project at www.cflib.org is an open source collection of CFML user-defined functions.

© 2014 Adobe Systems Incorporated. All rights reserved.

Using ColdFusion components

ColdFusion components (CFCs) are ColdFusion templates that contain related functions and arguments that each function accepts. The CFC contains the CFML tags necessary to define its functions and arguments and return a value. ColdFusion components are saved with a .cfc extension.
CFCs combine the power of objects with the simplicity of CFML. By combining related functions into a single unit, they provide an object or class shell from which functions can be called.
ColdFusion components can make their data private, so that it is available to all functions (also called methods) in the component, but not to any application that uses the component.
ColdFusion components have the following features:

- They are designed to provide related services in a single unit.
- They can provide web services and make them available over the Internet.
- They can provide ColdFusion services that Flash clients can call directly.
- They have several features that are familiar to object-oriented programmers, including data hiding, inheritance, packages, and introspection.

Recommended uses

Consider using ColdFusion components when doing the following:

- Creating web services. (To create web services in ColdFusion, you must use components.)
- Creating services that are callable by Flash clients.
- Creating libraries of related functions, particularly if they must share data.
- Using integrated application security mechanisms based on roles and the requestor location.
- Developing code in an object-oriented manner, in which you use methods on objects and can create objects that extend the features of existing objects.

For more information

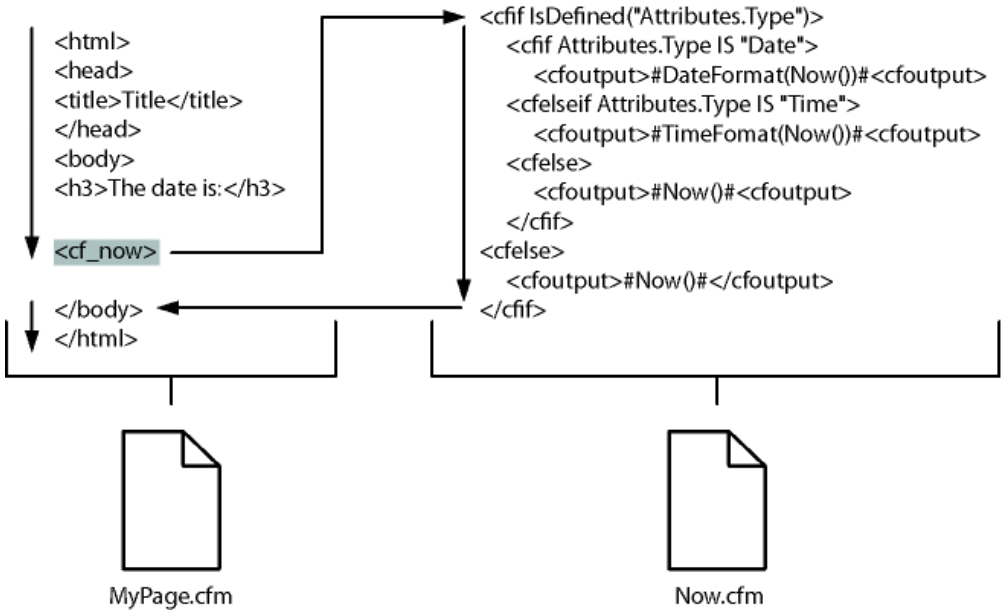
For more information on using ColdFusion components, see Building and Using ColdFusion Components

© 2014 Adobe Systems Incorporated. All rights reserved.

Using custom CFML tags

Custom tags written in CFML behave like ColdFusion tags. They can do all of the following:

- Take arguments.
- Have tag bodies with beginning and ending tags.
- Do specific processing when ColdFusion encounters the beginning tag.
- Do processing that is different from the beginning tag processing when ColdFusion encounters the ending tag.
- Have any valid ColdFusion page content in their bodies, including both ColdFusion built-in tags and custom tags (referred to as nested tags), or even JSP tags or JavaScript.
- Be called recursively; that is, a custom tag can, if designed properly, call itself in the tag body.
- Return values to the calling page in a common scope or the Variables scope of the calling page, but custom tags do not return values directly, the way functions do.
  Although a custom tag and a ColdFusion page that you include using the `cfinclude` tag are both ColdFusion pages, they differ in how they are processed. When a page calls a custom tag, it hands processing off to the custom tag page and waits until the custom tag page completes. When the custom tag finishes, it returns processing (and possibly data) to the calling page; the calling page can then complete its processing. The following image shows this process. The arrows indicate the flow of ColdFusion processing the pages.



Calling custom CFML tags

Unlike built-in tags, you can run custom CFML tags in the following three ways:

- Call a tag directly.
- Call a tag using the `cfmessagebox` tag.
- Use the `cfimport` tag to import a custom tag library directory.
  To call a CFML custom tag directly, precede the filename with `cf_`, omit the .cfm extension, and place the name in angle brackets (<>). For example, use the following line to call the custom tag defined by the file mytag.cfm:

```
<cf_myTag>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

If your tag takes a body, end it with the same tag name preceded with a forward slash, as follows:

```
</cf_myTag>
```

For information on using the `cfmodule` and `cfimport` tags to call custom CFML tags, see Creating and Using Custom CFML Tags.

Recommended uses

ColdFusion custom tags let you abstract complex code and programming logic into simple units. These tags let you maintain a CFML-like design scheme for your code. You can easily distribute your custom tags and share tags with others. For example, the ColdFusion Developer Exchange includes a library of custom tags that perform a wide variety of often-complex jobs; see www.adobe.com/go/learn_cfu_exchange_en.
Consider using CFML custom tags in the following circumstances:

- You need a tag-like structure, which has a body and an end tag, with the body contents changing from invocation to invocation.
- You want to associate specific processing with the beginning tag, the ending tag, or both tags.
- To use a logical structure in which the tag body uses "child" tags or subtags. This structure is like the `cfform` tag, which uses subtags for the individual form fields.
- You do not need a function format in which the calling code uses a direct return value.
- Your code must be recursive.
- Your functionality is complex.
- To distribute your code in a convenient form to others.
  If you can create either a UDF or a custom CFML tag for a purpose, first consider creating a UDF because running it requires less system overhead than using a custom tag.

For more information

For more information on custom CFML tags, see Creating and Using Custom CFML Tags.

© 2014 Adobe Systems Incorporated. All rights reserved.

Using CFX tags

ColdFusion Extension (CFX) tags are custom tags that you write in Java or C+. Generally, you create a CFX tag to do something that is not possible in CFML. CFX tags also let you use existing Java or C+ code in your ColdFusion application. Unlike CFML custom tags, CFX tags cannot have bodies or ending tags.
CFX tags can return information to the calling page in a page variable or by writing text to the calling page.
CFX tags can do the following:

- Have any number of custom attributes.
- Create and manipulate ColdFusion queries.
- Dynamically generate HTML that your page returns to the client.
- Set variables within the ColdFusion page from which they are called.
- Throw exceptions that result in standard ColdFusion error messages.

Calling CFX tags

To use a CFX tag, precede the class name with cfx_ and place the name in angle brackets. For example, use the following line to call the CFX tag defined by the MyCFXClass class and pass it one attribute.

```
<cfx_MyCFXClass myArgument="arg1">
```

Recommended uses

CFX tags provide one way of using C++ or Java code. However, you can also create Java classes and COM objects and access them using the `cfobject` tag. CFX tags, however, provide some built-in features that the `cfobject` tag does not have:

- CFX tags are easier to call in CFML code. You use CFX tags directly in CFML code as you would any other tag, and you can pass arguments using a standard tag format.
- ColdFusion provides predefined classes for use in your Java or C++ code that facilitate CFX tag development. These classes include support for request handling, error reporting, and query management.
CFX tags are useful in the following circumstances:

- You already have existing application functionality written in C++ or Java that you want to incorporate into your ColdFusion application.
- You cannot build the functionality you need using ColdFusion elements.
- You want to provide the new functionality in a tag format, as opposed to using the `cfobject` tag to import native Java or COM objects.
- You want to use the Java and C++ classes provided by ColdFusion for developing your CFX code.

For more information

For more information on CFX tags, see Building Custom CFXAPI Tags.

© 2014 Adobe Systems Incorporated. All rights reserved.

Selecting among ColdFusion code reuse methods

The following table lists common reasons to employ code reuse methods and indicates the techniques to consider for each purpose. The letter **P** indicates that the method is preferred. (There can be more than one preferred method.) The letter **A** means that the method provides an alternative that is useful in some circumstances.
This table does not include CFX tags. You use CFX tags only when it is best to code your functionality in C++ or Java. For more information about using CFX tags, see Using CFX tags.

Purpose	cfinclude tag	Custom tag	UDF	Component
Provide code, including CFML, HTML, and static text, that must be used in multiple pages.	P			
Deploy headers and footers.	P			
Include one page in another page.	P			
Divide pages into smaller units.	P			
Use variables from a calling page.	A	P	P	
Implement code that uses recursion.		P	P	P
Distribute your code to others.		P	P	P
Operate on a body of HTML or CFML text.		P		
Use subtags.		P		
Provide a computation, data manipulation, or other procedure.		A	P	

© 2014 Adobe Systems Incorporated. All rights reserved.

Provide a single functional element that takes any number of input values and returns a (possibly complex) result.		A	P	
Use variables with names that change from use to use.		A	P	P
Provide accessibility from Flash clients.		A	A	P
Use built-in user security features.			A	P
Encapsulate multiple related functions and properties.				P
Create web services.				P
Implement object-oriented coding methodologies.				P

#back to top

© 2014 Adobe Systems Incorporated. All rights reserved.

Writing and Calling User-Defined Functions

Creating custom functions for algorithms or procedures that you call frequently lets you organize and reuse the functions in your Adobe ColdFusion application pages.

© 2014 Adobe Systems Incorporated. All rights reserved.

About user-defined functions

You can create your own custom functions, known as *user-defined functions*, or UDFs. You then use them in your application pages the same way you use standard ColdFusion functions. You can also organize functions you create by grouping related functions into ColdFusion components. For more information, see Building and Using ColdFusion Components.

When you create a function for an algorithm or procedure that you use frequently, you can then use the function wherever you require the procedure. If you must change the procedure, you change only one piece of code. You can use your function anywhere that you can use a ColdFusion expression: in tag attributes, between number (#) signs in output, and in CFScript code. Typical uses of UDFs include, but are not limited to the following:

- Data manipulation routines, such as a function to reverse an array
- String and date/time routines, such as a function to determine whether a string is a valid IP address
- Mathematical calculation routines, including standard trigonometric and statistical operations or calculating loan amortization
- Routines that call functions externally, for example using COM or CORBA, including routines to determine the space available on a Windows file system drive

For information about selecting among user-defined functions, ColdFusion components, and custom tags, see Creating ColdFusion Elements.

> The Common Function Library Project at www.cflib.org is an open source collection of CFML user-defined functions.

© 2014 Adobe Systems Incorporated. All rights reserved.

Creating user-defined functions

Creating user-defined functions

Before you create a UDF, determine where you want to define it, and whether you want to use CFML or CFScript to create it.

Determining where to create a user-defined function

You can define a function in the following places:

- In a ColdFusion component. If you organize your functions in ColdFusion components, you use the functions as described in Using ColdFusion components.
- On the page where it is called. You can even define it below the place on the page where it is called, but this poor coding practice can result in confusing code.
- On a page that you include using a `cfinclude` tag. The `cfinclude` tag must be executed before the function gets called. For example, you can define all the functions for your application's on a single page and place a `cfinclude` tag at the top of pages that use the functions.
- On any page that places the function name in a scope common with the page on which you call the function. For more information on UDF scoping, see *Specifying the scope of a function* in Using UDFs effectively.
- On the Application.cfc or Application.cfm page. For more information, see Designing and Optimizing a ColdFusion Application.
  For recommendations on selecting where you define functions, see the sections *Using Application.cfm and function include files* and *Specifying the scope of a function* in Using UDFs effectively.

About creating functions using CFScript

You use the `function` statement to define the function in CFScript. CFScript function definitions have the following features and limitations:

- The function definition syntax is familiar to anyone who uses JavaScript or most programming languages.
- CFScript is efficient for writing business logic, such as expressions and conditional operations.
- CFScript function definitions cannot include CFML tags.
  The following is a CFScript definition for a function that returns a power of 2:

```
<cfscript>
function twoPower(exponent) {
 return 2^exponent;
}
</cfscript>
```

For more information on how to use CFScript to define a function, see Defining components and functions in CFScript.

Defining functions in CFScript

You define functions using CFScript in a similar manner defining JavaScript functions. You can define multiple functions in a single CFScript block.

> **Note**
>
> For more information on using CFScript, see Extending ColdFusion Pages with CFML Scripting.

© 2014 Adobe Systems Incorporated. All rights reserved.

CFScript function definition syntax

A CFScript function definition has the following syntax:

```
function functionName( [argName1[, argName2...]] )
 {
  CFScript Statements
 }
```

The following table describes the function variables:

Function variable	Description
functionName	The name of the function. You cannot use the name of a standard ColdFusion function or any name that starts with "cf". You cannot use the same name for two different function definitions. Function names cannot include periods.
argName1...	Names of the arguments required by the function. The number of arguments passed into the function must equal or exceed the number of arguments in the parentheses at the start of the function definition. If the calling page omits any of the required arguments, ColdFusion generates a mismatched argument count error.

The body of the function definition must be in curly brackets, even if it is empty.
The following two statements are allowed only in function definitions:

Statement	Description

© 2014 Adobe Systems Incorporated. All rights reserved.

`var` *variableName* = *expression*;	Creates and initializes a variable that is local to the function (function variable). This variable has meaning only inside the function and is not saved between calls to the function. It has precedence in the function body over any variables with the same name that exist in any other scopes. You never prefix a function variable with a scope identifier, and the name cannot include periods. The initial value of the variable is the result of evaluating the expression. The expression can be any valid ColdFusion expression, including a constant or even another UDF.All `var` statements must be at the top of the function declaration, before any other statements. Initialize all variables when you declare them. You cannot use the same name for a function variable and an argument.Each var statement can initialize only one variable.Use the var statement to initialize all function-only variables, including loop counters and temporary variables.
`return` *expression*;	Evaluates expression (which can be a variable), returns its value to the page that called the function, and exits the function. You can return any ColdFusion variable type.

A simple CFScript example

The following example function adds the two arguments and returns the result:

```
<cfscript>
function Sum(a,b) {
 var sum = a + b;
 return sum;
}
</cfscript>
```

In this example, a single line declares the function variable and uses an expression to set it to the value that it returns. This function can be simplified so that it does not use a function variable, as follows:

```
function MySum(a,b) {Return a + b;}
```

Always use curly brackets around the function definition body, even if it is a single statement.

Note

ColdFusion does not COPY any of the function arguments into the local scope of a function. However, if an unscoped variable is called, it is searched first in argument scope and then local scope.

© 2014 Adobe Systems Incorporated. All rights reserved.

About creating functions by using tags

You use the `cffunction` tag to define a UDF in CFML. The `cffunction` tag syntax has the following features and limitations:

- Developers who have a background in CFML or HTML, but no scripting or programming experience are more familiar with the syntax.
- You can include any ColdFusion tag in your function definition. Therefore, you can create a function, for example, that accesses a database.
- You can embed CFScript code inside the function definition.
- The `cffunction` tag provides attributes that enable you to easily limit the execution of the tag to authorized users or specify how the function can be accessed.
  The following code uses the `cffunction` tag to define the exponentiation function:

```
<cffunction name="twoPower" output=True>
 <cfargument name="exponent">
 <cfreturn 2^exponent>
</cffunction>
```

For more information on how to use the `cffunction` tag to define a function, see Defining components and functions in CFScript.

Defining functions by using the cffunction tag

The `cffunction` and `cfargument` tags let you define functions in CFML without using CFScript.
For information on ColdFusion components, see Building and Using ColdFusion Components. For more information on the `cffunction` tag, see the *CFML Reference*.

The cffunction tag function definition format

A `cffunction` tag function definition has the following format:

```
<cffunction name="functionName" [returnType="type" roles="roleList"
  access="accessType" output="Boolean"]>
 <cfargument name="argumentName" [Type="type" required="Boolean"
  default="defaultValue">]
 <!--- Function body code goes here. --->
 <cfreturn expression>
</cffunction>
```

where brackets ([]) indicate optional arguments. You can have any number of `cfargument` tags.
The `cffunction` tag specifies the name you use when you call the function. You can optionally specify other function characteristics, as the following table describes:{{}}

Attribute	Description
name	The function name.

© 2014 Adobe Systems Incorporated. All rights reserved.

`returnType`	(Optional) The type of data that the function returns. The valid standard type names are: any, array, binary, Boolean, date, guid, numeric, query, string, struct, uuid, variableName, xml, and void. If you specify any other name, ColdFusion requires the argument to be a ColdFusion component with that name. ColdFusion throws an error if you specify this attribute and the function tries to return data with a type that ColdFusion cannot automatically convert to the one you specified. For example, if the function returns the result of a numeric calculation, a returnType attribute of string or numeric is valid, but array is not.
`roles`	(Optional) A comma-delimited list of security roles that can run this method. If you omit this attribute, ColdFusion does not restrict user access to the function.If you use this attribute, the function executes only if the current user is logged in using the `cflogin user` tag and is a member of one or more of the roles specified in the attribute. Otherwise, ColdFusion throws an unauthorized access exception. For more information on user security, see Securing Applications.
`output`	(Optional) Determines how ColdFusion processes displayable output in the function body.If you do not specify this option, ColdFusion treats the body of the function as normal CFML. As a result, text and the result of any `cfoutput` tags in the function definition body are displayed each time the function executes.If you specify `true` or `yes`, the body of the function is processed as if it is in a `cfoutput` tag. ColdFusion displays variable values and expression results if you surround the variables and expressions with number signs (#).If you specify `false` or `no`., the function is processed as if it is in a `cfsilent` tag. The function does not display any output. The code that calls the function is responsible for displaying any function results.

Use `cfargument` tags for required function arguments. All `cfargument` tags must precede any other CFML code in a `cffunction` tag body. Therefore, place the `cfargument` tags immediately following the `cffunction` openin g tag. The `cfargument` tag takes the following attributes:

Attribute	Description
`name`	The argument name.

© 2014 Adobe Systems Incorporated. All rights reserved.

`type`	(Optional) The data type of the argument. The type of data that is passed to the function. The valid standard type names are any, array, binary, Boolean, date, guid, numeric, query, string, struct, uuid, and variableName. If you specify any other name, ColdFusion requires the argument to be a ColdFusion component with that name. ColdFusion throws an error if you specify this attribute and the function is called with data of a type that ColdFusion cannot automatically convert to the one you specified. For example, if the argument `type` attribute is numeric, you cannot call the function with an array.
`required`	(Optional) A Boolean value that specifies whether the argument is required. If set to `true` and the argument is omitted from the function call, ColdFusion throws an error. The default value is `false`. The required attribute is not required if you specify a `default` attrib ute.Because you do not identify arguments when you call a function, all `cfargument` tags that specify required arguments must precede any `cfargument` ta gs that specify optional arguments in the `cffunction` definition.
`default`	(Optional) The default value for an optional argument if no argument value is passed. If you specify this attribute, ColdFusion ignores the `required` attribute.

Note

The `cfargument` tag is not required for optional arguments. This feature is useful if a function can take an indeterminate number of arguments. If you do not use the `cfargument` tag for an optional argument, reference it by using its position in the Arguments scope array. For more information see Working with arguments and variables in functions.

Using a CFML tag in a user-defined function

The most important advantage of using the `cffunction` tag over defining a function in CFScript is that you can include CFML tags in the function. Thus, UDFs can encapsulate activities, such as database lookups, that require ColdFusion tags. Also, you can use the `cfoutput` tag to display output on the calling page with minimal coding.

Note

To improve performance, avoid using the `cfparam` tag in ColdFusion functions. Instead, use the `cfset` tag.

The following example function looks up and returns an employee department ID. It takes one argument, the employee ID, and looks up the corresponding department ID in the cfdocexamples Employee table:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cffunction name="getDeptID" >
 <cfargument name="empID" required="true" type="numeric">
 <cfset var cfdocexamples="">
 <cfquery dataSource="cfdocexamples" name="deptID">
  SELECT Dept_ID
  FROM Employee
  WHERE Emp_ID = #empID#
 </cfquery>
 <cfreturn deptID.Dept_ID>
</cffunction>
```

Rules for function definitions

The following rules apply to functions that you define using CFScript or the `cffunction` tag:

- The function name must be unique. It must be different from any existing variable, or UDF, except that you can use the ColdFusion advanced security function names.
- You can have a user-defined function with the same name as a built-in function for a CFC but not for CFM.
- You cannot use the following names to create user-defined functions:
  - writedump
  - writelog
  - location
  - throw
  - trace
- The function name must not start with the letters **cf** in any form. (For example, CF_MyFunction, cfmyFunction, and cfxMyFunction are not valid UDF names.)
- You cannot redefine or overload a function. If a function definition is active, ColdFusion generates an error if you define a second function with the same name.
- You cannot nest function definitions; that is, you cannot define one function inside another function definition.
- The function can be recursive, that is, the function definition body can call the function.
- The function does not have to return a value.
  You can use tags or CFScript to create a UDF. Each technique has advantages and disadvantages.

© 2014 Adobe Systems Incorporated. All rights reserved.

Calling user-defined functions

Calling user-defined functions

You can call a function anywhere that you can use an expression, including in number signs (#) in a `cfoutput` tag, in a CFScript, or in a tag attribute value. One function can call another function, and you can use a function as an argument to another function.
You can call a UDF in two ways:

- With unnamed, positional arguments, as you would call a built-in function
- With named arguments, as you would use attributes in a tag
  You can use either technique for any function. However, if you use named arguments, use the same argument names to call the function as you use to define the function. You cannot call a function with a mixture of named and unnamed arguments.
  One example of a user-defined function is a TotalInterest function that calculates loan payments based on a principal amount, annual percentage, and loan duration in months. (The definition of this function, see A user-defined function example). You can call the function without argument names on a form action page, as follows:

```
<cfoutput>
 Interest: #TotalInterest(Form.Principal, Form.Percent, Form.Months)#
</cfoutput>
```

You can call the function with argument names, as follows:

```
<cfoutput>
Interest: #TotalInterest(principal=Form.Principal, annualPercent=Form.Percent,
 months=Form.Months)#
</cfoutput>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

Working with arguments and variables in functions

Working with arguments and variables in functions

Good argument naming practice

Use an argument name that represents its use. For example, the following code is unlikely to result in confusion:

```
<cfscript>
 function SumN(Addend1,Addend2)
 { return Addend1 + Addend2; }
</cfscript>
<cfset x = 10>
<cfset y = 12>
<cfoutput>#SumN(x,y)#</cfoutput>
```

The following, similar code is more likely to result in programming errors:

```
<cfscript>
 function SumN(x,y)
 { return x + y; }
</cfscript>
<cfset x = 10>
<cfset y = 12>
<cfoutput>#SumN(x,y)#<cfoutput>
```

Passing arguments

ColdFusion passes the following data types to the function by value:

- Integers
- Real numbers
- Strings (including lists)
- Date-time objects
- Arrays
  As a result, any changes that you make in the function to these arguments do not affect the variable that was used to call the function, even if the calling code is on the same ColdFusion page as the function definition.
  ColdFusion passes queries, structures, and external objects such as COM objects into the function by reference. As a result, any changes to these arguments in the function also change the value of the variable in the calling code.
  For an example of the effects of passing arguments, see *Passing complex data* in Handling complex data types.

Passing complex data

Structures, queries, and complex objects such as COM objects are passed to UDFs by reference, so the function uses the same copy of the data as the caller. Arrays are passed to user-defined functions by value, so the function gets a new copy of the array data, and the array in the calling page is unchanged by the function. As a result, always handle arrays differently from all other complex data types.

Passing structures, queries, and objects

© 2014 Adobe Systems Incorporated. All rights reserved.

For your function to modify the copy of a structure, query, or object, in the caller, pass the variable as an argument. Because the function gets a reference to the structure in the caller, the caller variable reflects all changes in the function. You do not have to return the structure to the caller. After the function returns, the calling page accesses the changed data by using the structure variable that it passed to the function.
If you do not want a function to modify the copy of a structure, query, or object, in the caller, use the `Duplicate` fun ction to make a copy and pass the copy to the function.

Passing arrays

If you want your function to modify the caller's copy of the array, the simplest solution is to pass the array to the function and return the changed array to the caller in the function `return` statement. In the caller, use the same variable name in the function argument and return variable.
The following example shows how to directly pass and return arrays. In this example, the `doubleOneDArray` functi on doubles the value of each element in a one-dimensional array.

```
<cfscript>
//Initialize some variables
//This creates a simple array.
a=ArrayNew(1);
a[1]=2;
a[2]=22;
//Define the function.
function doubleOneDArray(OneDArray) {
 var i = 0;
 for ( i = 1; i LE arrayLen(OneDArray); i = i + 1)
  { OneDArray[i] = OneDArray[i] * 2; }
 return OneDArray;
}
//Call the function.
a = doubleOneDArray(a);
</cfscript>
<cfdump var="#a#">
```

This solution is simple, but it is not always optimal:

- This technique requires ColdFusion to copy the entire array twice, once when you call the function and once when the function returns. Doing so is inefficient for large arrays and can reduce performance, particularly if the function is called frequently.
- You can use the return value for other purposes, such as a status variable.
  If you do not use the `return` statement to return the array to the caller, you can pass the array as an element in a structure and change the array values inside the structure. Then the calling page can access the changed data by using the structure variable it passed to the UDF.
  The following code shows how to rewrite the previous example using an array in a structure. It returns True as a status indicator to the calling page and uses the structure to pass the array data back to the calling page.

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cfscript>
//Initialize some variables.
//This creates a simple array as an element in a structure.
arrayStruct=StructNew();
arrayStruct.Array=ArrayNew(1);
arrayStruct.Array[1]=2;
arrayStruct.Array[2]=22;
//Define the function.
function doubleOneDArrayS(OneDArrayStruct) {
 var i = 0;
 for ( i = 1; i LE arrayLen(OneDArrayStruct.Array); i = i + 1)
  { OneDArrayStruct.Array[i] = OneDArrayStruct.Array[i] * 2; }
 return True;
}
//Call the function.
Status = doubleOneDArrayS(arrayStruct);
WriteOutput("Status: " & Status);
</cfscript>
</br>
<cfdump var="#arrayStruct#">
```

Use the same structure element name for the array (in this case Array) in the calling page and the function.

About the Arguments scope

All function arguments exist in their own scope, the Arguments scope.
The Arguments scope exists for the life of a function call. When the function returns, the scope and its variables are destroyed.
However, destroying the Argument scope does not destroy variables, such as structures or query objects, that ColdFusion passes to the function by reference. The variables on the calling page that you use as function arguments continue to exist; if the function changes the argument value, the variable in the calling page reflects the changed value.
The Arguments scope is special, in that you can treat the scope as either an array *or* a structure. This dual nature of the Arguments scope is useful because it makes it easy to use arguments in any of the following circumstances:

- You define the function using CFScript.
- You define the function using the `cffunction` tag.
- You pass arguments using argument name=value format.
- You pass arguments as values only.
- The function takes optional, undeclared arguments.

The contents of the Arguments scope

The following rules apply to the Arguments scope and its contents:

- The scope contains all the arguments passed into a function.
- If you use `cffunction` to define the function, the scope always contains an entry "slot" for each declared argument, even if you do not pass the argument to the function when you call it. If you do not pass a declared (optional) argument, the scope entry for that argument is empty.When you call a function that you defined using CFScript, Pass the function a value for each argument declared in the function definition. Therefore, the Arguments scope for a CFScript call does not have empty slots.
  The following example shows these rules. Assume that you have a function declared, as follows:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cffunction name="TestFunction">
 <cfargument name="Arg1">
 <cfargument name="Arg2">
</cffunction>
```

You can call this function with a single argument, as in the following line:

```
<cfset TestFunction(1)>
```

The resulting Arguments scope looks like the following:

As an array		As a structure	
Entry	**Value**	**Entry**	**Value**
1	1	Arg1	1
2	undefined	Arg2	undefined

In this example, the following functions return the value 2 because the scope contains two defined arguments:

```
ArrayLen(Arguments)
StructCount(Arguments)
```

However, the following tests return the value `false`, because the contents of the second element in the Arguments scope is undefined.

```
Isdefined("Arguments.Arg2")
testArg2 = Arguments[2]>
Isdefined("testArg2")
```

Note

The `IsDefined` function does not test the existence of array elements. Use the function `ArrayContains` to search the array elements.

Using the Arguments scope as an array

The following rules apply to referencing Arguments scope as an array:

- If you call the function using unnamed arguments, the array index is the position of the argument in the function call.

© 2014 Adobe Systems Incorporated. All rights reserved.

- If you use names to pass the arguments, the array indexes correspond to the order in which the arguments are declared in the function definition.
- If you use names to pass arguments, and do not pass all the arguments defined in the function, the Arguments array has an empty entry at the index corresponding to the argument that was not passed. This rule applies only to functions created using the `cffunction` tag.
- If you use a name to pass an optional argument that is not declared in the function definition, the array index of the argument is the sum of the following:
  - The number of arguments defined with names in the function.
  - The position of the optional argument among the arguments passed in that do not have names defined in the function.
    However, using argument names in this manner is not good programming practice because you cannot ensure that you always use the same optional argument names when calling the function.
    To demonstrate these rules, define a simple function that displays the contents of its Arguments array and call the function with various argument combinations, as the following example shows:

```
<cffunction name="TestFunction" >
 <cfargument name="Arg1">
 <cfargument name="Arg2">
 <cfloop index="i" from="1" to="#ArrayLen(Arguments)#">
  <cfoutput>Argument #i#: #Arguments[i]#<br></cfoutput>
 </cfloop>
</cffunction>

<strong>One Unnamed argument</strong><br>
<cfset TestFunction(1)>
<strong>Two Unnamed arguments</strong><br>
<cfset TestFunction(1, 2)>
<strong>Three Unnamed arguments</strong><br>
<cfset TestFunction(1, 2, 3)>
<strong>Arg1:</strong><br>
<cfset TestFunction(Arg1=8)>
<strong>Arg2:</strong><br>
<cfset TestFunction(Arg2=9)>
<strong>Arg1=8, Arg2=9:</strong><br>
<cfset TestFunction(Arg1=8, Arg2=9)>
<strong>Arg2=6, Arg1=7</strong><br>
<cfset TestFunction(Arg2=6, Arg1=7)>
<strong>Arg1=8, Arg2=9, Arg3=10:</strong><br>
<cfset TestFunction(Arg1=8, Arg2=9, Arg3=10)>
<strong>Arg2=6, Arg3=99, Arg1=7</strong><br>
<cfset TestFunction(Arg2=6, Arg3=99, Arg1=7)>
```

Note

Although you can use the Arguments scope as an array, the `IsArray(Arguments)` function always returns `false` and the `cfdump` tag displays the scope as a structure.

Using the Arguments scope as a structure

The following rule applies when referencing Arguments scope as a structure:

- Use the argument names as structure keys. For example, if your function definition includes a Principal argument, reference the argument as Arguments.Principal.The following rules are also true, but *avoid writing*

© 2014 Adobe Systems Incorporated. All rights reserved.

code that uses them. To ensure program clarity, only use the Arguments structure for arguments that you name in the function definition. Use the Arguments scope as an array for optional arguments that you do not declare in the function definition.

- If you do not name an optional argument in the function definition, but do use a name for it in the function call, use the name specified in the function call For example, if you have an unnamed optional argument and call the function using the name myOptArg for the argument, you can reference the argument as Arguments.myOptArg in the function body. This usage, however, is poor programming practice, as it makes the function definition contents depend on variable names in the code that calls the function.

Using the Arguments scope in CFScript

A function can have optional arguments that you do not have to specify when you call the function. To determine the number of arguments passed to the function, use the following function:

```
ArrayLen(Arguments)
```

When you define a function using CFScript, the function must use the Arguments scope to retrieve the optional arguments. For example, the following SumN function adds two or more numbers together. It requires two arguments and supports any number of additional optional arguments. You can reference the first two, required, arguments as `Arg1` and `Arg2` or as `Arguments1` and `Arguments2`. Access the third, fourth, and any additional optional arguments as `Arguments3`, `Arguments4`, and so on

```
function SumN(Arg1,Arg2) {
 var arg_count = ArrayLen(Arguments);
 var sum = 0;
 var i = 0;
 for( i = 1 ; i LTE arg_count; i = i + 1 )
 {
  sum = sum + Arguments[i];
 }
 return sum;
}
```

With this function, any of the following function calls are valid:

```
SumN(Value1, Value2)
SumN(Value1, Value2, Value3)
SumN(Value1, Value2, Value3, Value4)
```

and so on.
The code never uses the Arg1 and Arg2 argument variables directly, because their values are always the first two elements in the Arguments array and it is simpler to step through the array. Specifying Arg1 and Arg2 in the function definition ensures that ColdFusion generates an error if you pass the function one or no arguments.

© 2014 Adobe Systems Incorporated. All rights reserved.

> **Note**
>
> Avoid referring to a required argument in the body of a function by both the argument name and its place in the Arguments scope array or structure, as doing so can be confusing and makes it easier to introduce errors.

Using the Arguments scope in cffunction definitions

When you define a function using the `cffunction` tag, you generally reference the arguments directly by name if all arguments are named in the `cfargument` tags. If you do use the Arguments scope identifier, follow the rules listed in About the Arguments scope.
For more information on using the Arguments scope in functions defined using CFScript, see Using the Arguments scope in CFScript.

Function-only variables

In addition to the Arguments scope, each function can have variables that exist only inside the function, and are not saved between times the function gets called. As soon as the function exits, all the variables in this scope are removed.
In CFScript, you create function-only variables with the `var` statement. Unlike other variables, you *never* prefix function-only variables with a scope name.

Using function-only variables

Make sure to use the `var` statement in CFScript UDFs to declare all function-specific variables, such as loop indexes and temporary variables that are required only for the duration of the function call. Doing so ensures that these variables are available inside the function only, and makes sure that the variable names do not conflict with the names of variables in other scopes. If the calling page has variables of the same name, the two variables are independent and do not affect each other.
For example, if a ColdFusion page has a `cfloop` tag with an index variable i, and the tag body calls a CFScript UDF that also has a loop with a function-only index variable i, the UDF does not change the value of the calling page loop index, and the calling page does not change the UDF index. So you can safely call the function inside the `cfloop` tag body.
In general, use the `var` statement to declare all UDF variables, other than the function arguments or shared-scope variables, that you use only inside CFScript functions. Use another scope, however, if the value of the variable must persist between function calls; for example, for a counter that the function increments each time it is called.

Referencing caller variables

A function can use and change any variable that is available in the calling page, including variables in the caller's Variables (local) scope, as if the function was part of the calling page. For example, if you know that the calling page has a local variable called Customer_name (and no function scope variable named Customer_name exists) the function can read and change the variable by referring to it as Customer_name or (using better coding practice) Variables.Customer_name. Similarly, you can create a local variable inside a function and then use it anywhere in the calling page *after* the function call. You cannot use the variable before you call the function.
However, generally avoid using the caller's variables directly inside a function. Using the caller's variables creates a dependency on the caller. Ensure that the code outside the function uses the same variable names as the function. Doing so can become difficult if you call the function from many pages.
You can avoid these problems by using only the function arguments and the return value to pass data between the caller and the function. Do not reference calling page variables directly in the function. As a result, you can use the function anywhere in an application (or even in multiple applications), without concern for the calling code variables.
As with other programming practices, valid exceptions to this recommendation exist. For example, you can do any of the following:

© 2014 Adobe Systems Incorporated. All rights reserved.

- Use a shared scope variable, such as an Application or Session scope counter variable.
- Use the Request scope to store variables used in the function. For more information, see Using the Request scope for static variables and constants.
- Create context-specific functions that work directly with caller data if you *always* synchronize variable names.

> **Note**
>
> If your function must directly change a simple variable in the caller (one that is not passed to the function by reference), you can place the variable inside a structure argument.

Using arguments

Function arguments can have the same names, but different values, as variables in the caller. Avoid such uses for clarity, however.
The following rules apply to argument persistence:

- Because ColdFusion passes simple variable and array arguments by value, their names and values exist only while the function executes.
- Because ColdFusion passes structures, queries, and objects such as COM objects by reference, the argument *name* exists only while the function executes, but the underlying *data* persists after the function returns and can be accessed by using the variable name of the caller. The variable name of the caller and the argument name can be different.

> **Note**
>
> If a function must use a variable from another scope that has the same name as a function-only variable, prefix the external variable with its scope identifier, such as Variables or Form. (However, remember that using variables from other scopes directly in your code is often poor practice.)

© 2014 Adobe Systems Incorporated. All rights reserved.

Using UDFs effectively

Using UDFs effectively

Many techniques help you use user-defined functions more effectively.

Using functions in ColdFusion component

In many cases, the most effective use of UDFs is within a CFC. For more information on CFCs, see Building and Using ColdFusion Components.

Using Application.cfm and function include files

Consider the following techniques for making your functions available to your ColdFusion pages:

- If you consistently call a small number of UDFs, consider putting their definitions on the Application.cfm page.
- If you call UDFs in only a few of your application pages, do not include their definitions in Application.cfm.
- If you use many UDFs, place their definitions on one or more ColdFusion pages that contain only UDFs. You can include the UDF definition page in any page that calls the UDFs.
  The next section describes other techniques for making UDFs available to your ColdFusion pages.

Specifying the scope of a function

User-defined function names are essentially ColdFusion variables. ColdFusion variables are names for data. Function names are names (references) for segments of CFML code. Therefore, like variables, functions belong to scopes.

About functions and scopes

Like ColdFusion variables, UDFs exist in a scope:

- When you define a UDF, ColdFusion puts it in the Variables scope.
- You can assign a UDF to a scope the same way you assign a variable to a scope, by assigning the function to a name in the new scope. For example, the following line assigns the MyFunc UDF to the Request scope:

  ```
  <cfset Request.MyFunc = Variables.MyFunc>
  ```

  You can now use the function from any page in the Request scope by calling Request.MyFunc.

Selecting a function scope

The following table describes the advantages and disadvantages of each function scope:

Scope	Considerations
Application	Makes the function available across all invocations of the application. Access to UDFs in Application scope is multithreaded and you can execute multiple copies of the UDF at one time.

© 2014 Adobe Systems Incorporated. All rights reserved.

Request	Makes the function available for the life of the current HTTP request, including in all custom tags and nested custom tags. This scope is useful if a function is used in a page and in the custom tags it calls, or in nested custom tags.
Server	Makes the function available to all pages on a single server. In most cases, this scope is not a good choice because in clustered systems, it only makes the function available on a single server, and all code that uses the function must be inside a cflock block.
Session	Makes the function available to all pages during the current user session. This scope has no significant advantages over the Application scope.

Using the Request scope

You can effectively manage functions that are used in application pages and custom tags by doing the following:

1. Define the functions on a function definitions page.
2. On the functions page, assign the functions to the request scope.
3. Use a `cfinclude` tag to include the function definition page on the application page, but do not include it on any custom tag pages.
4. Always call the functions using the request scope.
   This way you only include the functions once per request and they are available throughout the life of the request. For example, create a myFuncs.cfm page that defines your functions and assigns them to the Request scope using syntax such as the following:

```
function MyFunc1(Argument1, Argument2)
{ Function definition goes here }
Request.MyFunc1 = MyFunc1
```

The application page includes the myFuncs.cfm page:

```
<cfinclude template="myfuncs.cfm">
```

The application page and all custom tags (and nested custom tags) call the functions as follows:

```
Request.MyFunc1(Value1, Value2)
```

Using the Request scope for static variables and constants

You can partially break the rule described in the section *Referencing caller variables* in the Working with arguments and variables in functions. Here, the function defines variables in the Request scope. However, it is a specific solution to a specific issue, where the following circumstances exist:

© 2014 Adobe Systems Incorporated. All rights reserved.

- Your function initializes a large number of variables.
- The variables have either of the following characteristics:
  - They must be *static*: only the function uses them, the function can change their values, and their values must persist from one invocation of the function to the next.
  - They are *named constants*; that is the variable value never changes.
- Your application page (and any custom tags) calls the function multiple times.
- You can assure that only the function uses the variable names.
  In these circumstances, you can improve efficiency and save processing time by defining your function's variables in the Request scope, rather than the Function scope. The function tests for the Request scope variables and initializes them if they do not exist. In subsequent calls, the variables exist and the function does not reset them.
  The `NumberAsString` function, written by Ben Forta and available from www.cflib.org, takes advantage of this technique.

Using function names as function arguments

Because function names are ColdFusion variables, you can pass a function's name as an argument to another function. This technique allows a function to use another function as a component. For example, a calling page can call a calculation function, and pass it the name of a function that does some subroutine of the overall function. This way, the calling page could use a single function for different specific calculations, such as calculating different forms of interest. The initial function provides the framework, while the function whose name is passed to it can implement a specific algorithm that the calling page requires.
The following simple example shows this use. The `binop` function is a generalized function that takes the name of a function that performs a specific binary operation and two operands. The `binop` function simply calls the specified function and passes it the operands. This code defines a single operation function, the sum function. A more complete implementation would define multiple binary operations.

```
<cfscript>
function binop(operation, operand1, operand2)
{ return (operation(operand1, operand2)); }
function sum(addend1, addend2)
{ return addend1 + addend2;}
x = binop(sum, 3, 5);
writeoutput(x);
</cfscript>
```

Handling query results using UDFs

When you call a UDF in the body of a tag that has a `query` attribute, such as a `cfloop` tag, any function argument that is a query column name passes a single element of the column, not the entire column. Therefore, the function must manipulate a single query element.
For example, the following code defines a function to combine a single first name and last name to make a full name. It queries the cfdocexamples database to get the first and last names of all employees, and then it uses a `cfoutput` tag to loop through the query and call the function on each row in the query.

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cfscript>
function FullName(aFirstName, aLastName)
 { return aFirstName & " " & aLastName; }
</cfscript>

<cfquery name="GetEmployees" datasource="cfdocexamples">
 SELECT FirstName, LastName
 FROM Employee
</cfquery>

<cfoutput query="GetEmployees">
#FullName(FirstName, LastName)#<br>
</cfoutput>
```

You generally use functions that manipulate many rows of a query *outside* tags that loop over queries. Pass the query to the function and loop over it inside the function. For example, the following function changes text in a query column to uppercase. It takes a query name as an argument.

```
function UCaseColumn(myquery, colName) {
 var currentRow = 1;
 for (; currentRow lte myquery.RecordCount; currentRow = currentRow + 1)
 {
  myquery[colName][currentRow] = UCase(myquery[colName][currentRow]);
 }
 Return "";
}
```

The following code uses a script that calls the `UCaseColumn` function to convert all the last names in the GetEmployees query to uppercase. It then uses `cfoutput` to loop over the query and display the contents of the column.

```
<cfscript>
 UCaseColumn(GetEmployees, "LastName");
</cfscript>
<cfoutput query="GetEmployees">
 #LastName#<br>
</cfoutput>
```

Identifying and checking for UDFs

You can use the `IsCustomFunction` function to determine whether a name represents a UDF. The `IsCustomFunction` function generates an error if its argument does not exist. As a result, ensure that the name exists before calling the function, for example, by calling the `IsDefined` function. The following code shows this use:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cfscript>
if(IsDefined("MyFunc"))
 if(IsCustomFunction(MyFunc))
  WriteOutput("MyFunc is a user-defined function");
 else
  WriteOutput("Myfunc is defined but is NOT a user-defined function");
else
 WriteOutput("MyFunc is not defined");
</cfscript>
```

You do *not* surround the argument to `IsCustomFunction` in quotation marks, so you can use this function to determine if function arguments are themselves functions.

Using the Evaluate function

If your user-defined function uses the `Evaluate` function on arguments that contain strings, Make sure that all variable names you use as arguments include the scope identifier. Doing so avoids conflicts with function-only variables.
The following example returns the result of evaluating its argument. It produces the expected results, the value of the argument, if you pass the argument using its fully scoped name, Variables.myname. However, the function returns the value of the function local variable if you pass the argument as myname, without the Variables scope identifier.

```
<cfscript>
 myname = "globalName";
 function readname(name) {
  var myname = "localName";
  return (Evaluate(name));
 }
</cfscript>

<cfoutput>
<!--- This one collides with local variable name. --->
 The result of calling readname with myname is:
  #readname("myname")# <br>
<!--- This one finds the name passed in. --->
 The result of calling readname with Variables.myname is:
  #readname("Variables.myname")#
</cfoutput>
```

Using recursion

A *recursive* function is a function that calls itself. Recursive functions are useful when an algorithm that repeats the same operation multiple times using the results of the preceding repetition can solve the problem. Factorial calculation, used in the following example, is one case where recursion is useful. The Towers of Hanoi game is also solved using a recursive algorithm.
A recursive function, like looping code, must have an end condition that always stops the function. Otherwise, the function continues until a system error occurs or you stop the ColdFusion server.
The following example calculates the factorial of a number, that is, the product of all the integers from 1 through the number; for example, 4 factorial is 4 X 3 X 2 X 1 = 24.

© 2014 Adobe Systems Incorporated. All rights reserved.

```
function Factorial(factor) {
 If (factor LTE 1)
  return 1;
 else
  return factor * Factorial(factor -1);
}
```

If the function is called with a number greater than 1, it calls itself using an argument one less than it received. It multiplies that result by the original argument, and returns the result. Therefore, the function keeps calling itself until the factor is reduced to 1. The final recursive call returns 1, and the preceding call returns 2 * 1, and so on, until all the initial call returns the end result.

> If a recursive function calls itself too many times, it causes a stack overflow. Always test any recursive functions under conditions that are likely to cause the maximum number of recursions to ensure that they do not cause a stack overflow.

#back to top

© 2014 Adobe Systems Incorporated. All rights reserved.

Handling errors in UDFs

Handling errors in UDFs

ColdFusion provides several techniques to handle errors in UDFs:

- Display error messages directly in the function.
- Return function status information to the calling page.
- Use `try`/`catch` or `cftry`/`cfcatch` blocks and the `cfthrow` and `cfrethrow` tags to handle and generate exceptions.
  The technique you use depends on the circumstances of your function and application and on your preferred programming style. However, it is best for most functions to use the second or third technique, or a combination of the two.

Displaying error messages

Your function can test for errors and use the `WriteOutput` function to display an error message directly to the user. This method is useful for providing immediate feedback to users for simple input errors. You can use it independently or in conjunction with either of the other two error-handling methods.
For example, the following variation on a "Hello world" function displays an error message if you do not enter a name in the form:

```
<cfform method="POST" action="#CGI.script_name#">
 <p>Enter your Name: 
 <input name="name" type="text" hspace="30" maxlength="30">
 <input type="Submit" name="submit" value="OK">
</cfform>
<cfscript>
 function HelloFriend(Name) {
  if (Name is "") WriteOutput("You forgot your name!");
  else WriteOutput("Hello " & name &"!");
  return "";
 }
 if (IsDefined("Form.submit")) HelloFriend(Form.name);
</cfscript>
```

Reviewing the code

The following table describes the code:

Code	Description

© 2014 Adobe Systems Incorporated. All rights reserved.

`<cfform method="POST"`
`action="#CGI.script_name#">`
`<p>Enter your Name: `
`<input name="name" type="text"`
`hspace="30"`
`maxlength="30">`
`<input type="Submit" name="submit"`
`value="OK">`
`</cfform>`
	Creates a simple form requesting you to enter your name.Uses the script_name CGI variable to post to this page without specifying a URL. If you do not enter a name, the form posts an empty string as the name field.

`<cfscript>`
`function HelloFriend(Name) {`
`if (Name is "") WriteOutput("You`
`forgot your  name!");`
`else WriteOutput("Hello " & name`
`&"!"); return "";`
`}`
`if (IsDefined("Form.submit"))`
`HelloFriend(Form.name);`
`</cfscript>`
	Defines a function to display "Hello name!" First, checks whether the argument is an empty string. If so, displays an error message. Otherwise displays the hello message. Returns the empty string. (The caller does not use the return value). It is not necessary to use curly brackets around the if or else statement bodies because they are single statements.If this page has been called by submitting the form, calls the HelloFriend function. Otherwise, the page just displays the form.

Providing status information

In some cases, such as those where the function cannot provide a corrective action, the function cannot, or should not, handle the error directly. In these cases, your function can return information to the calling page. The calling page must handle the error information and act appropriately.
Consider the following mechanisms for providing status information:

- Use the return value to indicate the function status only. The return value can be a Boolean success/failure indicator. The return value can also be a status code, for example where 1 indicates success, and various failure types are assigned known numbers. With this method, the function must set a variable in the caller to the value of a successful result.
- Set a status variable that is available to the caller (not the return variable) to indicate success or failure and any information about the failure. With this method, the function can return the result directly to the caller. In this method, the function uses only the return value and structure arguments to pass the status back to the caller.
  Each of these methods can have variants, and each has advantages and disadvantages. The technique that you use depends on the type of function, the application in which you use it, and your coding style.
  The following example, which modifies the function used in A user-defined function example, uses one version of the status variable method. It provides two forms of error information:

- It returns -1, instead of an interest value, if it encounters an error. This value can serve as an error indicator because you never pay negative interest on a loan.
- It also writes an error message to a structure that contains an error description variable. Because the message is in a structure, it is available to both the calling page and the function.

The TotalInterest function

After changes to handle errors, the `TotalInterest` function looks like the following. Code that is changed from the

© 2014 Adobe Systems Incorporated. All rights reserved.

example in A user-defined function example is in bold.

```
<cfscript>
function TotalInterest(principal, annualPercent, months, status) {
 Var years = 0;
 Var interestRate = 0;
 Var totalInterest = 0;
 principal = trim(principal);
 principal = REReplace(principal,"[\$,]","","ALL");
 annualPercent = Replace(annualPercent,"%","","ALL");
 if ((principal LE 0) OR (annualPercent LE 0) OR (months LE 0)) {
  Status.errorMsg = "All values must be greater than 0";
  Return -1;
 }
 interestRate = annualPercent / 100;
 years = months / 12;
 totalInterest = principal*(((1+ interestRate)^years)-1);
 Return DollarFormat(totalInterest);
}
</cfscript>
```

Reviewing the code

The following table describes the code that has been changed or added to the previous version of this example. For a description of the initial code, see A user-defined function example.

Code	Description
`function TotalInterest(principal, annualPercent, months, status)`	The function now takes an additional argument, a status structure. Uses a structure for the status variable so that changes that the function makes affect the status structure in the caller.

`if ((principal LE 0) OR (annualPercent LE 0) OR (months LE 0)) {`
`Status.errorMsg = "All values must be greater than 0";`
`Return -1;`
`}`
	Checks to make sure the principal, percent rate, and duration are all greater than zero.If any is not, sets the errorMsg key (the only key) in the Status structure to a descriptive string. Also, returns -1 to the caller and exits the function without processing further.

Calling the function

The code that calls the function now looks like the following. Code that is changed from the example in A user-defined function example is in bold.

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cfset status = StructNew()>
<cfset myInterest = TotalInterest(Form.Principal,
 Form.AnnualPercent,Form.Months, status)>
<cfif myInterest EQ -1>
 <cfoutput>
  ERROR: #status.errorMsg#<br>
 </cfoutput>
<cfelse>
 <cfoutput>
  Loan amount: #Form.Principal#<br>
  Annual percentage rate:
   #Form.AnnualPercent#<br>
  Loan duration: #Form.Months# months<br>
  TOTAL INTEREST: #myInterest#<br>
 </cfoutput>
</cfif>
```

Reviewing the code

The following table describes the code that has been changed or added:

Code	Description
`<cfset status = StructNew()>`	Creates a structure to hold the function status.
`<cfset myInterest = TotalInterest (Form.Principal, Form.AnnualPercent, Form.Months, status)>`	Calls the function. This time, the function requires four arguments, including the status variable.
`<cfif myInterest EQ -1> <cfoutput> ERROR: #status.errorMsg#  </cfoutput>`	If the function returns -1, there must be an error. Displays the message that the function placed in the status.errorMsg structure key.

© 2014 Adobe Systems Incorporated. All rights reserved.

`<cfelse>` `<cfoutput>` `Loan amount: #Form.Principal# ` `Annual percentage rate:` `#Form.AnnualPercent# ` `Loan duration: #Form.Months#` `months ` `TOTAL INTEREST: #myInterst# ` `</cfoutput>` `</cfif>`	If the function does not return -1, it returns an interest value. Displays the input values and the function return value.

Using exceptions

UDFs written in CFScript can handle exceptions using the `try` and `catch` statements. UDFs written using the `cffunction` tag can use the `cftry`, `cfcatch`, `cfthrow`, and `cfrethrow` tags. Using exceptions corresponds to the way many functions in other programming languages handle errors, and can be an effective way to handle errors. In particular, it separates the functional code from the error-handling code, and it can be more efficient than other methods at runtime, because it does not require testing and branching.
Exceptions in UDFs have the following two dimensions:

- Handling exceptions generated by running the UDF code
- Generating exceptions when the UDF identifies invalid data or other conditions that would cause errors if processing continued

Handling exceptions in UDFs

Use try/catch blocks to handle exceptions in a UDF under the same conditions that any other ColdFusion application uses try/catch blocks. These conditions are typically circumstances where the function uses an external resource, such as a Java, COM, or CORBA object, a database, or a file. When possible, have your application prevent, rather than catch, exceptions caused by invalid application data. For example, the application can prevent users from entering a zero value for a form field that is used to divide another number, rather than handling exceptions generated by dividing by zero.
When ColdFusion catches an exception, the function can use any of the following methods to handle the exception:

- If the error is recoverable (for example, if the problem is a database time-out where in some cases retrying resolves the issue), try to recover from the problem.
- Display a message, as described in *Displaying error messages* in this page.
- Return an error status, as described in *Providing status information* in this page.
- If the UDF is defined using the `cffunction` tag, throw a custom exception, or rethrow the exception so that it the calling ColdFusion page catches it. For more information on throwing and rethrowing exceptions, see Handling runtime exceptions with ColdFusion tags.

Generating exceptions in UDFs

If you define your function using the `cffunction` tag, you can use the `cfthrow` and `cfrethrow` tags to throw errors to the page that called the function. You can use this technique whenever your UDF identifies an error, instead of displaying a message or returning an error status. For example, the following code rewrites the example from *Providing status information* in this page to use the `cffunction` tag and CFML, and to throw and handle an exception if any of the form values are not positive numbers.
The lines that identify invalid data and throw the exception are in bold. The remaining lines are equivalent to the CFScript code in the previous example. However, the code that removes unwanted characters must precede the error checking code.

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cffunction name="TotalInterest">
 <cfargument name="principal" required="Yes">
 <cfargument name="annualPercent" required="Yes">
 <cfargument name="months" required="Yes">
 <cfset var years = 0>
 <cfset var interestRate = 0>
 <cfset var totalInterest = 0>

 <cfset principal = trim(principal)>
 <cfset principal = REReplace(principal,"[\$,]","","ALL")>
 <cfset annualPercent = Replace(annualPercent,"%","","ALL")>

 <cfif ((principal LE 0) OR (annualPercent LE 0) OR (months LE 0))>
  <cfthrow type="InvalidData" message="All values must be greater than 0.">
 </cfif>

 <cfset interestRate = annualPercent / 100>
 <cfset years = months / 12>
 <cfset totalInterest = principal*
   (((1+ interestRate)^years)-1)>
 <cfreturn DollarFormat(totalInterest)>
</cffunction>
```

The code that calls the function and handles the exception looks like the following. The changed lines are in bold.

```
<cftry>
 <cfset status = StructNew()>
 <cfset myInterest = TotalInterest(Form.Principal, Form.AnnualPercent,
  Form.Months, status)>
 <cfoutput>
  Loan amount: #Form.Principal#<br>
  Annual percentage rate: #Form.AnnualPercent#<br>
  Loan duration: #Form.Months# months<br>
  TOTAL INTEREST: #myInterest#<br>
 </cfoutput>
<cfcatch type="InvalidData">
 <cfoutput>
  #cfcatch.message#<br>
 </cfoutput>
</cfcatch>
```

</cftry>

© 2014 Adobe Systems Incorporated. All rights reserved.

A user-defined function example

A user-defined function example

The following simple function takes a principal amount, an annual percentage rate, and a loan duration in months and returns the total amount of interest paid over the period. You can optionally use the percent sign for the percentage rate, and include the dollar sign and comma separators for the principal amount.
You could use the `TotalInterest` function in a `cfoutput` tag of a form's action page, as follows:

```
<cfoutput>
 Loan amount: #Form.Principal#<br>
 Annual percentage rate: #Form.AnnualPercent#<br>
 Loan duration: #Form.Months# months<br>
 TOTAL INTEREST: #TotalInterest(Form.Principal, Form.AnnualPercent,
Form.Months)#<br>
</cfoutput>
```

Defining the function using CFScript

```
<cfscript>
function TotalInterest(principal, annualPercent, months) {
 Var years = 0;
 Var interestRate = 0;
 Var totalInterest = 0;
 principal = trim(principal);
 principal = REReplace(principal,"[\$,]","","ALL");
 annualPercent = Replace(annualPercent,"%","","ALL");
 interestRate = annualPercent / 100;
 years = months / 12;
 totalInterest = principal*(((1+ interestRate)^years)-1);
 Return DollarFormat(totalInterest);
}
</cfscript>
```

Reviewing the code

The following table describes the code:

Code	Description
`function TotalInterest(principal, annualPercent, months) {`	Starts the `TotalInterest` function definition. Requires three variables: the principal amount, the annual percentage rate, and the loan duration in months.

© 2014 Adobe Systems Incorporated. All rights reserved.

`Var years = 0;`
`Var interestRate = 0;`
`Var totalInterest = 0;`
	Declares intermediate variables used in the function and initializes them to 0. All `var` statements must precede the rest of the function code.

`principal = trim(principal);`
`principal = REReplace(principal,"[\$,]","","ALL");`
`annualPercent = Replace(annualPercent,"%","","ALL");`
`interestRate = annualPercent / 100;`
`years = months / 12;`
	Removes any leading or trailing spaces from the principal argument. Removes any dollar sign ($) and comma (,) characters from the principal argument to get a numeric value.Removes any percent (%) character from the `annualPercent` argument to get a numeric value, then divides the percentage value by 100 to get the interest rate.Converts the loan from months to years.

`totalInterest = principal* (((1+ interestRate)^years)-1);`
`Return DollarFormat(totalInterest);`
`}`
	Calculates the total amount of interest due. It is possible to calculate the value in the `Return` statement, but this example uses an intermediate totalInterest variable to make the code easier to read. Returns the result formatted as a US currency string.Ends the function definition.

Defining the function using the cffunction tag

The following code replaces CFScript statements with their equivalent CFML tags.

```
<cffunction name="TotalInterest">
 <cfargument name="principal" required="Yes">
 <cfargument name="annualPercent" required="Yes">
 <cfargument name="months" required="Yes">
 <cfset var years = 0>
 <cfset var interestRate = 0>
 <cfset var totalInterest = 0>
 <cfset principal = trim(principal)>
 <cfset principal = REReplace(principal,"[\$,]","","ALL")>
 <cfset annualPercent = Replace(annualPercent,"%","","ALL")>
 <cfset interestRate = annualPercent / 100>
 <cfset years = months / 12>
 <cfset totalInterest = principal*
   (((1+ interestRate)^years)-1)>
 <cfreturn DollarFormat(totalInterest)>
</cffunction>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

Building and Using ColdFusion Components

A ColdFusion component (CFC) file contains data and functions that you define in related, multiple methods. You use CFC pages to organize related actions in one file, which provide can simplify your programming. For more information on creating applications that use CFCs, see the Adobe website: www.adobe.com.

© 2014 Adobe Systems Incorporated. All rights reserved.

About ColdFusion components

A ColdFusion component (CFC) is a file saved with the extension .cfc. A CFC can contain data and functions. Within a CFC, data is referred to as *properties*. Although you use the cffunction tag to define functions within a CFC, they are typically referred to as *methods* instead of functions.
The page on which you define a CFC is also known as a *component page*. Component pages use the same tags and functions that regular CFML pages do, plus a small number of special tags (in particular, the cfcomponent tag) and tag attributes.
You define related methods in a CFC. Unlike ColdFusion custom tags, a single CFC can perform many related actions, defined in multiple methods. The methods can share a data context, such as metadata and scoping, or manage a particular database or set of tables. For example, you can define the methods to insert, update, delete, and retrieve records from a particular database or table in one CFC.

CFCs and object-oriented programming

CFCs are building blocks that let you develop ColdFusion code in an object-oriented manner, although CFCs do not require you to do object-oriented programming. Some of the object-oriented features of CFCs include encapsulation, inheritance, and introspection. CFC object-oriented features are like the object-oriented elements in other languages, like JavaScript.
The technique of incorporating both code and data into one object such as a CFC is known as *encapsulation*. Encapsulation lets users pass data to and get a result from your CFC without having to understand the underlying code. When you use encapsulation, you can validate data that is passed to the CFC. CFCs can also enforce data types, check for required parameters, and optionally assign default values.
One CFC can *inherit* the methods and properties of another CFC. Inheritance lets you build multiple specific components without rewriting the code for the basic building blocks of the components. For more information, see *U sing the Super keyword* in Using CFCs effectively.
CFCs support *introspection*; that is, they can provide information about themselves. If you display a component page directly in an HTML browser, inspect it in the ColdFusion and Adobe Dreamweaver CS3 component browsers, or use the CFML `GetMetadata` function, you see information about the component. This information includes its path, property, methods, and additional information that you can specify using special documentation attributes and tags. For more information, see *Using introspection to get information about components* in Using CFCs effectively.
When you use a ColdFusion component, you can *invoke* a method in the CFC. However, typically, you create an *ins tance* of the CFC, and then invoke methods and refer to properties of the CFC.

When to use CFCs

You can use CFCs in the following ways:

- Developing structured, reusable code
- Creating web services
- Creating Flash Remoting elements
- Using asynchronous CFCs

Developing structured, reusable code

CFCs provide an excellent method for developing structured applications that separate display elements from logical elements and encapsulate database queries. You can use CFCs to create application functionality that you (and others) can reuse wherever needed, like user-defined functions (UDFs) and custom tags. If you want to modify, add, or remove component functionality, you make changes in only one component file.
CFCs have several advantages over UDFs and custom tags. These advantages, which CFCs automatically provide, include all of the following:

- The ability to group related methods into a single component, and to group related components into a package
- Properties that multiple methods can share

© 2014 Adobe Systems Incorporated. All rights reserved.

- The This scope, a component-specific scope
- Inheritance of component methods and properties from a base component, including the use of the Super keyword
- Access control
- Introspection for CFC methods, properties, and metadata
  CFCs have one characteristic that prevents them from being the automatic choice for all code reuse. It takes relatively more processing time to instantiate a CFC than to process a custom tag. In turn, it takes substantially more time to process a custom tag than to execute a user-defined function (UDF). However, after a CFC is instantiated, calling a CFC method has about the same processing overhead as an equivalent UDF. As a result, do not use CFCs in place of independent, single-purpose custom tags or UDFs. Instead, use CFCs to create bodies of related methods, particularly methods that share properties.
  For more information about UDFs, custom tags, and other ColdFusion code reuse techniques, see Creating ColdFusion Elements.

Creating web services

ColdFusion can automatically publish CFC methods as web services. To publish a CFC method as a web service, you specify the `access="remote"` attribute in the method's cffunction tag. ColdFusion generates all the required Web Services Description Language (WSDL) code and exports the CFC methods. For more information on creating web services in ColdFusion, see Using Web Services.

Creating Flash Remoting elements

Adobe Flash applications that use Flash Remoting can easily take advantage of ColdFusion components for business logic. In a CFC, the cffunction tag names the function and contains the application logic, and the cfreturn tag returns the result to Flash.

> **Note**
>
> For ColdFusion component methods to communicate with Flash applications, set the `access` attribute of the cffunction tag to remote.

For more information on creating CFCs for Flash Remoting, see Using Flash with CFCs.

Using asynchronous CFCs

ColdFusion provides an event gateway that lets you send a message to a CFC asynchronously. This gateway lets you initialize processing by a CFC without waiting for the CFC to complete or return a value. You can use asynchronous CFCs that use this gateway for the following:

- Reindexing a collection
- Logging information
- Running batch processes
  For more information on using asynchronous CFCs, see About event gateways.

© 2014 Adobe Systems Incorporated. All rights reserved.

Creating ColdFusion components

When you create CFCs, you create methods, which are ColdFusion user-defined functions, in the component page. You pass data to a method by using parameters. The method then performs the function and, if specified in the `cfreturn` tag, returns data.
You can also define variables in a CFC. Within a CFC, these variables are known as *properties*.

Tags for creating CFCs

You use the following tags to create a CFC. You include these tags on the CFML page that defines the CFC.

Tag	Description
cfcomponent	Contains a component definition; includes attributes for introspection. For more information, see Building ColdFusion components.
cffunction	Defines a component method (function); includes attributes for introspection. For more information, see *Defining component methods* in Building ColdFusion components.
cfargument	Defines a parameter (argument) to a method; includes attributes for introspection. For more information, see *Defining and using method parameters* in Building ColdFusion components.
cfproperty	Defines variables for CFCs that provide web services; also use to document component properties. For more information, see *The cfproperty tag* in cfproperty.

Elements of a CFC

A CFC has the following characteristics:

- It is a single CFML page with a .cfc filename extension. The component name is the same as the filename. For example, if the file is myComponent.cfc, the component name is myComponent.
- The page is surrounded by a cfcomponent tag. No code can be outside this tag.
- The component page defines methods (functions), properties (data), or both. Most CFCs have methods, or methods and properties, but you can also have a CFC that contains only properties.
- You use the cffunction tag to define CFC methods. The CFScript `function` statement can create simple methods, but it does not provide options to control access to the method, provide metadata, specify a return type, or control generated output.
- You can write code on the component page that is outside `cffunction` definitions. This code executes when the CFC is instantiated or whenever you invoke a method of the CFC.

© 2014 Adobe Systems Incorporated. All rights reserved.

Building ColdFusion components

You use the cfcomponent and cffunction tags to create ColdFusion components. By itself, the `cffunction` tag does not provide functionality. The `cfcomponent` tag provides an envelope that describes the functionality that you build in CFML and enclose in `cffunction` tags. The following example shows the skeleton of a component with two methods:

```
<cfcomponent>
 <cffunction name="firstMethod">
   <!--- CFML code for this method goes here. --->
 </cffunction>
 <cffunction name="secondMethod">
   <!--- CFML code for this method goes here. --->
 </cffunction>
</cfcomponent>
```

Defining component methods

You define component methods using cffunction tags. The following example defines a CFC that contains two methods, `getall` and `getsalary`:

```
<cfcomponent>
 <cffunction name="getall" output="false" returntype="query">
  <cfset var queryall="">
  <cfquery name="queryall" datasource="cfdocexamples">
   SELECT * FROM EMPLOYEE
  </cfquery>
  <cfreturn queryall>
 </cffunction>
 <cffunction name="getsalary" output="false">
  <cfset var getNamesandSalary="">
   <cfquery name="getNamesandSalary" datasource="cfdocexamples">
    SELECT FirstName, LastName, Salary FROM EMPLOYEE
   </cfquery>
  <cfreturn getNamesandSalary>
 </cffunction>
</cfcomponent>
```

Because component methods are ColdFusion functions, most of their features and coding techniques are identical to those of user-defined functions. For more information on using the `cffunction` tag to create functions, see Writing and Calling User-Defined Functions. Like other ColdFusion functions, CFC methods can display information directly by generating output, or can return a value to the code or client that invoked the method.
You use the following `cffunction` tag attributes only for CFCs:

- The `displayname` and `hint` attributes, which document the CFC; for more information, see *Documenting CFCs* in Building ColdFusion components.
- The `access` attribute, which controls access to the CFC; for more information, see *Using access security* in Using CFCs effectively.
  For detailed reference information on the cffunction tag, see the *CFML Reference*.

Defining CFCs with related methods

© 2014 Adobe Systems Incorporated. All rights reserved.

When defining CFCs, it is good programming practice to organize related methods in one CFC. For example, you could place all methods that perform operations related to a user, such as `addUser`, `editUser`, and `storeUserPreferences`, in one CFC. You can group related mathematical functions into one CFC. A CFC can also contain all the methods and properties necessary for a shopping cart. The following CFC contains two cffunction tags that define two component methods, `getEmp` and `getDept`. When invoked, the component methods query the ExampleApps database. The cfreturn tag returns the query results to the client, or page, where the method was invoked.

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

Placing executable code in a separate file

You can place executable code in a separate file from the main component definition page. By placing the method execution code in a separate file, you can separate property initialization code, meta information, and the method definition shell from the executable method definition code. This technique lets you modularize your code and helps prevent CFML pages from getting too long and complex.
To separate the component method code, use a cfinclude tag on the component definition page to call the page that contains the component method code.

> **Note**
>
> If your method takes arguments or returns data to the page that invokes it, the cfargument tag and the cfreturn tag must be on the component definition page, not on the included page.

Create a component method by using the cfinclude tag

1. Create a tellTime.cfc file with the following code:

```
<cfcomponent>
 <cffunction name="getUTCTime">
  <cfinclude template="getUTCTime.cfm">
 <cfreturn utcStruct.Hour & ":" & utcStruct.Minute>
 </cffunction>
</cfcomponent>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

1. Create a ColdFusion page with the following code, and save it as getUTCTime.cfm in the same directory as tellTime.cfc:

```
<cfscript>
 serverTime=now();
 utcTime=GetTimeZoneInfo();
 utcStruct=structNew();
 utcStruct.Hour=DatePart("h", serverTime);
 utcStruct.Minute=DatePart("n", serverTime);
 utcStruct.Hour=utcStruct.Hour + utcTime.utcHourOffSet;
 utcStruct.Minute=utcStruct.Minute + utcTime.utcMinuteOffSet;
 if (utcStruct.Minute LT 10) utcStruct.Minute = "0" & utcStruct.Minute;
</cfscript>
```

In the example, the `getUTCTime` method definition calls the getUTCTime.cfm file with the cfinclude tag. The getUTCTime.cfm code calculates the UTC time representation of the current time and populates a structure with hour and minute values. The method in tellTime.cfc then uses the information in the structure to return the current UTC time as a string to the calling page. The included page must not include a `cfreturn` statem ent.

Initializing instance data

Some components have *instance data*, which is data that persists as long as the component instance exists. For example, a shopping cart component can have instance data that includes the IDs and quantities of items that the user places in the shopping cart. Instance data is often shared by several methods that can create, delete, or modify the data.

You can refer to instance data of a CFC only if you create an instance of the CFC. From inside the CFC, you refer to instance data of the CFC using the `this` prefix, for example `this.firstvariable`. From the calling page, you refer to instance data using dot notation, including the name of the instance of the component and the name of the instance data, as in `objectname.ivarname`. Components whose methods you invoke without first instantiating the component do not typically have instance data.

You initialize instance data at the top of the component definition, before the method definitions. ColdFusion executes this code when it instantiates the component; for example, when a cfobject tag creates the component instance. Because this code executes only when the instance is created and it typically "constructs" properties of the component, instance data initialization code is sometimes called *constructor* code.

You can use any CFML tag or function in constructor code, and the code can perform any ColdFusion processing, such as querying a database or data validation and manipulation. If one component extends another, the parent component's constructor code executes before the child component's constructor code.

> **Note**
>
> ColdFusion does not require you to place the initialization code at the top of the component definition; however, it is good programming practice to do so.

The following example shows constructor code for a shopping cart CFC:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cfcomponent>
 <!--- Initialize the array for the cart item IDs and quantities. --->
 <cfset This.CartData = ArrayNew(2)>
 <!--- The following variable has the ID of the "Special Deal" product for
   this session. --->
 <cfset This.Special_ID = RandRange(1, 999)>
```

For information on scopes, see The This scope and The Variables scope.
A useful technique is to define a method named `init()`, which initializes an instance of a CFC, acting as a constructor. The `init()` method can initialize constants and return an instance of the component to the calling page. The following code illustrates an example of an `init()` method:

```
<cfcomponent displayname="shoppingCart">
 <cffunction name="init" access="public" output="no" returntype="shoppingCart">
  <cfargument name="shoppingCartID" type="UUID" required="yes">
  <cfset variables.shoppingCartID = arguments.shoppingCartID>
  <cfreturn this>
 </cffunction>

 <!--- Additional methods go here. --->
</cfcomponent>
```

In this example, the `init()` method uses the variables scope to make the shopping cart ID available anywhere in the CFC. For more information about scope, see CFC variables and scope.

Defining and using method parameters

You pass data to a method by using parameters. To define a component method parameter, use the cfargument tag in the cffunction tag body. To define multiple parameters, use multiple `cfargument` tags. The tag names a parameter and lets you specify the following:

- Whether the parameter is required
- The type of data that is required
- A default argument value
- Display name and hint metadata for CFC introspection

> **Note**
>
> You can create CFC methods that do not use `cfargument` tags, for example, if you use positional parameters in your methods. However, most CFC methods use the `cfargument` tag.

Example: convertTemp.cfc

The convertTemp.cfc file consists of the following:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cfcomponent>
 <!--- Celsius to Fahrenheit conversion method. --->
 <cffunction name="ctof" output="false">
  <cfargument name="temp" required="yes" type="numeric">
  <cfreturn ((temp*9)/5)+32>
 </cffunction>

 <!--- Fahrenheit to Celsius conversion method. --->
 <cffunction name="ftoc" output="false">
  <cfargument name="temp" required="yes" type="numeric">
  <cfreturn ((temp-32)*5/9)>
 </cffunction>
</cfcomponent>
```

Reviewing the code

The convertTemp CFC contains two methods that convert temperature. The following table describes the code and its function:

Code	Description
`<cfcomponent>`	Defines the component.
`<cffunction name="ctof" output="false">`	Defines the `ctof` method.Indicates that this method does not display output.
`<cfargument name="temp" required="yes" type="numeric">`	Creates the `temp` parameter of the `ctof` method. Indicates that it is required and that the expected value is `numeric`.
`<cfreturn ((temp*9)/5)+32>`	Defines the value that the method returns.
`</cffunction>`	Ends the method definition.
`<cffunction name="ftoc" output="false">`	Defines the `ftoc` method.Indicates that this method does not display output.
`<cfargument name="temp" required="yes" type="numeric">`	Creates the `temp` parameter of the `ftoc` method. Indicates that it is required and that the expected value is `numeric`.
`<cfreturn ((temp-32)*5/9)>`	Defines the value that the method returns.
`</cffunction>`	Ends the method definition.
`</cfcomponent>`	Ends the component definition.

Example: tempConversion.cfm

© 2014 Adobe Systems Incorporated. All rights reserved.

The ColdFusion page tempConversion.cfm is an HTML form in which the user enters the temperature to convert, and selects the type of conversion to perform. When the user clicks the Submit button, ColdFusion performs the actions on the processForm.cfm page. The file tempConversion.cfm, which is in the same directory as convertTemp.cfc, consists of the following:

```
<cfform action="processForm.cfm" method="POST">
 Enter the temperature:
  <input name="temperature" type="text"><br>
  <br>
 Select the type of conversion:<br>
  <select name="conversionType">
   <option value="CtoF">Celsius to Farenheit</option>
   <option value="FtoC">Farenheit to Celsius</option>
  </select><br><br>
 <input name="submitform" type="submit" value="Submit">
</cfform>
```

Example: processForm.cfm

The ColdFusion page processForm.cfm calls the appropriate component method, based on what the user entered in the form on the tempConversion.cfm page. Place it in the same directory as convertTemp.cfc.

```
<cfif #form.conversionType# is "CtoF">
 <cfinvoke component="convertTemp" method="ctof" returnvariable="newtemp"
  temp=#form.temperature#>
 <cfoutput>#form.temperature# degrees Celsius is #newtemp# degrees
  Farenheit.</cfoutput>
 <cfelseif #form.conversionType# is "FtoC">
  <cfinvoke component="convertTemp" method="ftoc"
   returnvariable="newtemp" temp=#form.temperature#>
  <cfoutput>#form.temperature# degrees Fahrenheit is #newtemp# degrees
   Celsius.</cfoutput>
</cfif>
```

Reviewing the code

The file processForm.cfm invokes the appropriate component method. The following table describes the code and its function:

Code	Description
`<cfif form.conversionType is "CtoF">`	Executes the code in the cfif block if the user selected Celsius to Fahrenheit as the conversion type in the form on the tempConversion.cfm page.

© 2014 Adobe Systems Incorporated. All rights reserved.

`<cfinvoke component="convertTemp" method="ctof" returnvariable="newtemp" arguments.temp="#form.temperature#">`	Invokes the `ctof` method of the `convertTemp` component, without creating an instance of the `convertTemp` component. Specifies `newtemp` as the result variable for the method. Assigns the temperature value that the user entered in the form to the variable `temp`, which is specified in the `cfargument` tag of the `ctof` method. When invoking the `ctof` method, the `temp` variable is assigned to the Arguments scope. For more information about variables and scope, see CFC variables and scope.
`<cfoutput>#form.temperature# degrees Celsius is #newtemp#}}b{{degrees Fahrenheit.</cfoutput>`	Displays the temperature that the user entered in the form, the text "degrees Celsius is," the new temperature value that results from the `ctof` method, and the text "degrees Fahrenheit."
`<cfelseif #form.conversionType# is "FtoC">`	Executes the code in the cfelseif block if the user selected Fahrenheit to Celsius as the conversion type in the form on the tempConversion.cfm page.
`<cfinvoke component="converttemp" method="ftoc" returnvariable="newtemp" temp=#form.temperature#>`	Invokes the `ftoc` method of the `convertTemp` component, without creating an instance of the `convertTemp` component. Specifies `newtemp` as the result variable for the method. Assigns the temperature value that the user entered in the form to the variable `temp`, which is specified in the `cfargument` tag of the `ftoc` method. When invoking the `ftoc` method, the `temp` variable is assigned to the Arguments scope. For more information about variables and scope, see CFC variables and scope.
`<cfoutput>#form.temperature# degrees Fahrenheit is #newtemp# degrees Celsius.</cfoutput>`	Displays the temperature that the user entered in the form, the text "degrees Fahrenheit is," the new temperature value that results from the `ftoc` method, and the text "degrees Celsius."
`</cfif>`	Closes the `cfif` block.

To run the example, display the tempConversion.cfm page in your browser. When you enter a value in the text box of the form, the value is stored in the `form.temperature` variable. Processing is then performed on the processForm.cfm page, which refers to the value as `form.temperature`. When you invoke either method, the cfinvoke tag assigns the value `form.temperature` to `temp`; `temp` is the argument specified in the `cfargument` tag of the appropriate method. The appropriate method in the `convertTemp` component performs the necessary calculations and returns the new value as `newtemp`.For detailed reference information on the `cfargument` tag, see the *CFML Reference*.To access the parameter values in the component method definition, use structure- or array-like notation with the Arguments scope. The following example refers to the `lastName` argument as `Arguments.lastname`; it could also refer to it as `Arguments1`. In addition, you can access arguments directly using number (#) signs, such as `#lastname#`; however, it is better programming practice to identify the scope (for example, `#Arguments.lastname#`). Also, you can use Array- or structure-like notation, which lets you loop over

© 2014 Adobe Systems Incorporated. All rights reserved.

multiple parameters. For more information on the Arguments scope, see *The Arguments scope* in CFC variables and scope.

Define parameters in the component method definition

Create a component with the following contents, and save it as corpQuery.cfc in a directory under your web root directory:

```
<cfcomponent>
 <cffunction name="getEmp">
  <cfargument name="lastName" type="string" required="true"
    hint="Employee last name">
  <cfset var empQuery="">
   <cfquery name="empQuery" datasource="cfdocexamples">
    SELECT LASTNAME, FIRSTNAME, EMAIL
    FROM tblEmployees
   WHERE LASTNAME LIKE '#Arguments.lastName#'
   </cfquery>
   <!--- Use cfdump for debugging purposes. --->
   <cfdump var=#empQuery#>
 </cffunction>
 <cffunction name="getCat" hint="Get items below specified cost">
 <cfargument name="cost" type="numeric" required="true">
  <cfset var catQuery="">
  <cfquery name="catQuery" datasource="cfdocexamples">
    SELECT ItemName, ItemDescription, ItemCost
    FROM tblItems
   WHERE ItemCost <= #Arguments.cost#
   </cfquery>
   <!--- Use cfdump for debugging purposes. --->
   <cfdump var=#catQuery#>
 </cffunction>
</cfcomponent>
```

In the example, the `cfargument` attributes specify the following:

- The `name` attributes define the parameter names.
- The `type` attribute for the `lastName` argument specifies that the parameter must be a text string. The `type` attribute for the `cost` argument specifies that the parameter must be a numeric value. These attributes validate the data before it is submitted to the database.
- The `required` attributes indicate that the parameters are required, if not, ColdFusion throws an exception.
- The Arguments scope provides access to the parameter values.

Providing results

ColdFusion components can provide information in the following ways:

- They can generate output that is displayed on the calling page.
- They can return a variable.
  You can use either technique, or a combination of both, in your applications. The best technique to use depends on your application's needs and your coding methodologies. For example, many CFC methods that perform business logic return the results as a variable, and many CFC methods that display output directly are designed as modular units for generating output, and do not do business logic.

Displaying output

© 2014 Adobe Systems Incorporated. All rights reserved.

If you do not specifically suppress output, any text, HTML code, or output that CFML tags generate inside your method gets returned as generated output to the client that calls the component method. If the client is a web browser, it displays these results. For example, the following `getLocalTime1` component method shows the local time directly on the page that invokes the method:

```
<cfcomponent>
 <cffunction name="getLocalTime1">
  <cfoutput>#TimeFormat(now())#</cfoutput>
 </cffunction>
</cfcomponent>
```

Component methods that are called by using Flash Remoting or as web services cannot use this method to provide results.

Returning a results variable

In the component method definition, you use the cfreturn tag to return the results to the client as variable data. For example, the following `getLocalTime2` component method returns the local time as a variable to the ColdFusion page or other client that invokes the method:

```
<cfcomponent>
 <cffunction name="getLocalTime">
  <cfreturn TimeFormat(now())>
 </cffunction>
</cfcomponent>
```

The ColdFusion page or other client, such as a Flash application, that receives the result then uses the variable data as appropriate.

> **Note**
>
> If a CFC is invoked using a URL or by submitting a form, ColdFusion returns the variable as a WDDX packet. A CFC that is invoked by Flash Remoting, or any other instance of a CFC, must not return the This scope.

You can return values of all data types, including strings, integers, arrays, structures, and instances of CFCs. The `cfreturn` tag returns a single variable, as does the `return` CFScript statement. Therefore, if you want to return more than one result value at a time, use a structure. If you do not want to display output in a method, use `output="false"` in the cffunction tag.
For more information on using the `cfreturn` tag, see the *CFML Reference*.

Documenting CFCs

ColdFusion provides several ways to include documentation about your CFCs in your component definitions. The documentation is available when you use introspection to display information about the CFC or call the `GetMetadata` or GetComponentMetaData function to get the component's metadata. You can use the following tools for documenting CFCs:

- The `displayname` and `hint` attributes

© 2014 Adobe Systems Incorporated. All rights reserved.

- User-defined metadata attributes
- The cfproperty tag
  For information on displaying the information, see *Using introspection to get information about components* in Using CFCs effectively.

The displayname and hint attributes

The cfcomponent, cffunction, cfargument, and cfproperty tags have `displayname` and `hint` attributes.
The `displayname` attribute lets you provide a more descriptive name for a component, attribute, method, or property. When you use introspection, this attribute appears in parentheses next to the component or method name, or on the parameter information line.
You use the `hint` attribute for longer descriptions of the component, method, or argument. In the introspection display, this attribute appears on a separate line or on several lines of the component or method description, and at the end of the argument description.

Metadata attributes

You can include arbitrary metadata information as attributes of the `cfcomponent`, `cffunction`, `cfargument`, and `cfproperty` tags. To create a metadata attribute, specify the metadata attribute name and its value. For example, in the following `cfcomponent` tag, the `Author` attribute is a metadata attribute. This attribute is not used as a function parameter; instead, it indicates who wrote this CFC.

```
<cfcomponent name="makeForm" Author="Bean Lapin">
```

Metadata attributes are not used by ColdFusion for processing; they also do not appear in standard ColdFusion introspection displays; however, you can access and display them by using the GetMetaData or GetComponentMetaData function to get the metadata. Each attribute name is a key in the metadata structure of the CFC element.
Metadata attributes are used for more than documentation. Your application can use the `GetMetadata` function to get the metadata attributes for a component instance, or the GetComponentMetaData function to get the metadata for an interface or component that you have not yet instantiated. You can then act based on the values. For example, a `mathCFC` component could have the following `cfcomponent` tag:

```
<cfcomponent displayname="Math Functions" MetaType="Float">
```

In this case, a ColdFusion page with the following code sets the `MetaTypeInfo` variable to `Float`:

```
<cfobject component="mathCFC" name="MathFuncs">
<cfset MetaTypeInfo=GetMetadata(MathFuncs).MetaType>
```

Note

All metadata values are replaced by strings in the metadata structure returned from the `GetMetadata` function. Because of this, do not use expressions in custom metadata attributes.

The cfproperty tag

© 2014 Adobe Systems Incorporated. All rights reserved.

The `cfproperty` tag is used to create complex data types with WSDL descriptors and for component property documentation, as follows:

- It can create complex data types with WSDL descriptions for ColdFusion web services. For more information, see *Using ColdFusion components to define data types for web services* in Publishing web services.
- It can provide documentation of component properties in the ColdFusion introspection output. The introspection information includes the values of the standard `cfproperty` tag attributes.

> **Note**
>
> The `cfproperty` tag does not create a variable or assign it a value. It is used for information purposes only. You use a `cfset` tag, or CFScript assignment statement, to create the property and set its value.

Saving and naming ColdFusion components

The following table lists the locations in which you can save component files and how they can be accessed from each location:

	URL	Form	Flash Remoting	Web services	ColdFusion page
Current directory	N/A	Yes	N/A	N/A	Yes
Web root	Yes	Yes	Yes	Yes	Yes
ColdFusion mappings	No	No	No	No	Yes
Custom tag roots	No	No	No	No	Yes

> **Note**
>
> ColdFusion mappings and custom tag roots can exist within the web root. If so, they are accessible to remote requests, including URL, form, Flash Remoting, and web services invocation.

When you store components in the same directory, they are members of a component *package.* You can group related CFCs into packages. Your application can refer to any component in a directory specifically by using a qualified component name that starts with a subdirectory of one of the accessible directories and uses a period to delimit each directory in the path to the directory that contains the component. For example, the following example is a qualified name of a component named `price`:

```
catalog.product.price
```

In this example, the price.cfc file must be in the catalog\product subdirectory of a directory that ColdFusion searches for components, as listed in the preceding table. When you refer to a component using the qualified name, ColdFusion looks for the component in the order described in *Specifying the CFC location* in Using ColdFusion

© 2014 Adobe Systems Incorporated. All rights reserved.

components-Developing guide.
Establishing a descriptive naming convention is a good practice, especially if you plan to install the components as part of a packaged application.

© 2014 Adobe Systems Incorporated. All rights reserved.

CFC variables and scope

CFCs interact with ColdFusion scopes and use local variables.

> **Note**
>
> Components also have a Super keyword that is sometimes called a scope. For information on the Super keyword, see *Using the Super keyword* in Using CFCs effectively.

The This scope

The This scope is available within the CFC and is shared by all CFC methods. It is also available in the base component (if the CFC is a child component), on the page that instantiates the CFC, and all CFML pages included by the CFC.
Inside the CFC, you define and access This scope variables by using the prefix This, as in the following line:

```
<cfset This.color="green">
```

In the calling page, you can define and access CFC This scope variables by using the CFC instance name as the prefix. For example, if you create a CFC instance named `car` and, within the `car` CFC specify `<cfset This.color="green">`, a ColdFusion page that instantiates the CFC could refer to the component's color property as `#car.color#`.
Variable values in the This scope last as long as the CFC instance exists and, therefore, can persist between calls to methods of a CFC instance.

> **Note**
>
> The This scope identifier works like the This keyword in JavaScript and ActionScript. CFCs do not follow the Java class model, and the This keyword behaves differently in ColdFusion than in Java. In Java, This is a private scope, whereas in ColdFusion, it is a public scope.

The Variables scope

The Variables scope in a CFC is private to the CFC. It includes variables defined in the CFC body (initialization or constructor code) and in the CFC methods. When you set Variables scope variables in the CFC, they cannot be seen by pages that invoke the CFC.
The CFC Variables scope does not include any of the Variables scope variables that are declared or available in the page that instantiates or invokes the CFC. However, you can make the Variables scope of the page that invokes a CFC accessible to the CFC by passing Variables as an argument to the CFC method.
You set a Variables scope variable by assigning a value to a name that has the Variables prefix or no prefix.
Values in the Variables scope last as long as the CFC instance exists, and therefore can last between calls to CFC instance methods.
The Variables scope is available to included pages, and Variables scope variables that are declared in the included page are available in the component page.

> **Note**
>
> The Variables scope is not the same as the function local scope, which makes variables private within a function. Always define function-local variables using the var keyword of the Local scope name.

© 2014 Adobe Systems Incorporated. All rights reserved.

Example: sharing the Variables scope

The following example shows how to make the Variables scope of the page that invokes a CFC accessible to the CFC by passing Variables as an argument to the CFC method. It also illustrates that the Variables scope is private to the CFC.
The following code is for the callGreetMe.cfm page:

```
<cfset Variables.MyName="Wilson">
<cfobject component="greetMe" name="myGreetings">
<cfoutput>
 Before invoking the CFC, Variables.Myname is: #Variables.MyName#.<br>
 Passing Variables scope to hello method. It returns:
#myGreetings.hello(Variables.MyName)#.<br>
After invoking the CFC, Variables.Myname is: #Variables.MyName#.<br>
</cfoutput>
<cfinvoke component="greetMe" method="VarScopeInCfc">
```

The following code is for the greetMe CFC:

```
<cfcomponent>
<cfset Variables.MyName="Tuckerman">
 <cffunction name="hello">
  <cfargument name="Name" Required=true>
  <cfset Variables.MyName="Hello " & Arguments.Name>
  <cfreturn Variables.MyName>
 </cffunction>
 <cffunction name="VarScopeInCfc">
  <cfoutput>Within the VarScopeInCfc method, Variables.MyName is:
#variables.MyName#<br></cfoutput>
 </cffunction>
</cfcomponent>
```

In this example, the callGreetMe.cfm page does the following:

1. Sets the MyName variable in its Variables scope to *Wilson*.
2. Displays the *Variables.MyName* value.
3. Calls the greetMe CFC and passes its Variables scope as a parameter.
4. Displays the value returned by the greetMe CFC.
5. Displays the *Variables.MyName* value.
6. Invokes the VarScopeInCfc method, which displays the value of *Variables.MyName* within the CFC.
   When you browse the callGreetMe.cfm page, the following appears:

```
Before invoking the CFC, Variables.Myname is: Wilson.
Passing Variables scope to hello method. It returns: Hello Wilson.
After invoking the CFC, Variables.Myname is: Wilson.
Within the VarScopeInCfc method, Variables.MyName is: Tuckerman
```

The Arguments scope

© 2014 Adobe Systems Incorporated. All rights reserved.

The Arguments scope exists only in a method, and is not available outside the method. The scope contains the variables that you passed into the method, including variables that you passed in the following ways:

- As named attributes to the `cfinvoke` tag
- In the `cfargumentcollection` attribute of the `cfinvoke` tag
- In cfinvokeargument tags
- As attributes or parameters passed into the method when the method is invoked as a web service, by Flash Remoting, as a direct URL, or by submitting a form
  You can access variables in the Arguments scope using structure notation (`Arguments.variablename`), or array notation (`Arguments1` or `Arguments["variablename"]`).
  The Arguments scope does not persist between calls to CFC methods.
  Variables in the Arguments scope are available to pages included by the method.

Other variable scopes

A CFC shares the Form, URL, Request, CGI, Cookie, Client, Session, Application, Server, and Flash scopes with the calling page. Variables in these scopes are also available to all pages that are included by a CFC. These variables do not have any behavior that is specific to CFCs.

Function local variables

Variables that you declare with the Var keyword inside a cffunction tag or CFScript `function` definition are available only in the method in which they are defined, and only last from the time the method is invoked until it returns the result. You cannot use the Var keyword outside function definitions.

> **Note**
>
> Always use the Var keyword or Local scope name on variables that are only used inside the function in which they are declared.

Define all function local variables at the top of the function definition, before any other CFML code; for example:

```
<cffunction ...>
 <cfset Var testVariable = "this is a local variable">
  <!--- Function code goes here. --->
 <cfreturn myresult>
</cffunction>
```

Any arguments declared with the `cfargument` tag must appear before any variables defined with the `cfset` tag. You can also place any `cfscript` tag first and define variables that you declare with the Var keyword in the script.
Use function local variables if you place the CFC in a persistent scope such as the Session scope, and the function has data that must be freed when the function exits.
Local variables do not persist between calls to CFC methods.
Local variables are available to pages included by the method.

© 2014 Adobe Systems Incorporated. All rights reserved.

Using ColdFusion components-Developing guide

You can use a CFC in two ways:

1. You can *instantiate* a CFC object, which creates a CFC instance. You then invoke the methods of the instance. You can access the CFC methods and data as instance elements. You can also use the instance in the cfinvoke tag to invoke the CFC methods. When you instantiate a CFC, data in the CFC is preserved as long as the CFC instance exists, and ColdFusion does not incur the overhead of creating the instance each time you call a method. Instantiate CFCs to preserve data in the CFC. To ensure processing efficiency if you use the CFC more than once on a page, instantiate the CFC before you invoke its methods.Methods that are executed remotely through Flash Remoting and web services always create an instance of the CFC before executing the method.
2. You can *invoke* (call) a method of the CFC without creating an instance of the CFC, which is referred to as *transiently* invoking a method. In this case, ColdFusion creates an instance of the CFC that exists only from the time you invoke the method until the method returns a result. No data is preserved between invocations and ColdFusion does not keep an instance of the CFC that you can reuse elsewhere in your CFML. It is considered a best practice to create an instance of a CFC before invoking any of its methods, unless your CFML request uses the CFC only once. If you transiently invoke a method frequently, consider creating a user-defined function to replace the CFC method.
   You can create persistent CFCs by assigning the CFC instance to a persistent scope, such as the Session or Application scope. This way, you can create CFCs for objects, such as shopping carts or logged-in users, that must persist for sessions. You can also create CFCs that provide application-specific data and methods.

Tags for using CFCs

The following table lists the tags that you use to instantiate or invoke a CFC. You use these tags on the CFML page on which you instantiate or invoke the CFC.

Tag	Description
cfinvoke	Invokes a method of a CFC.
cfinvokeargument	Passes the name and value of a parameter to a component method.
cfobject	Creates a CFC instance.
CreateObject	Creates a CFC instance.

CFC invocation techniques

ColdFusion provides many ways to instantiate CFCs and invoke CFC methods. The following table lists the techniques, including the ColdFusion tags and functions that you use:

Invocation	Description	For more information
cfinvoke tag	Invokes a component method. Can invoke methods of a CFC instance or invoke the methods transiently.	See *Invoking CFC methods with the cfinvoke tag* in this page.

© 2014 Adobe Systems Incorporated. All rights reserved.

cfset tag and assignment statements	Invoke methods and access properties of a component instance.	See *Using components directly in CFScript and CFML* in this page.
URL (HTTP GET)	Transiently invokes a component method by specifying the component and method names in the URL string.	See *Invoking component methods by using a URL* in this page.
Form control(HTTP POST)	Transiently invokes a component method using the HTML `form` and `input` tags and their attributes.	See *Invoking component methods by using a form* in this page.
Flash Remoting	ActionScript can transiently invoke component methods.	See Using the Flash Remoting Service.
Web services	The `cfinvoke` tag and CFScript consume web services in ColdFusion. External applications can also consume CFC methods as web services.	See Using Web Services.

Instantiating CFCs

If you use a CFC multiple times in a ColdFusion request, or if you use a CFC with persistent properties, use the cfobject tag or CreateObject function to instantiate the CFC before you call its methods.
The following example uses the `cfobject` tag to create an instance of the tellTime CFC.

```
<cfobject component="tellTime" name="tellTimeObj">
```

The following example uses the `CreateObject` function to instantiate the same component in CFScript:

```
tellTimeObj = CreateObject("component", "tellTime");
```

Invoking CFC methods with the cfinvoke tag

The cfinvoke tag can invoke methods on a CFC instance or invoke CFC methods transiently. You can also use the `cfinvoke` tag to invoke CFC methods from within a CFC.

Invoking methods of a CFC instance

To invoke a component method of a CFC instance, use the `cfinvoke` tag and specify the following:

- The CFC instance name, *enclosed in number signs* (#), in the `component` attribute.
- The method name, in the `method` attribute.
- Any parameters. For information on passing parameters, see *Passing parameters to methods by using the cfinvoke tag* in Passing parameters to methods.
- If the component method returns a result, the name of the variable for the result in the `returnVariable` attri

© 2014 Adobe Systems Incorporated. All rights reserved.

bute.
The following procedure creates an application that displays the current UTC and local time.

1. Create a file named tellTime2.cfc with the following code:

```
<cfcomponent>
 <cffunction name="getLocalTime" access="remote">
  <cfreturn TimeFormat(now())>
 </cffunction>
 <cffunction name="getUTCTime" access="remote">
  <cfscript>
   serverTime=now();
   utcTime=GetTimeZoneInfo();
   utcStruct=structNew();
   utcStruct.Hour=DatePart("h", serverTime);
   utcStruct.Minute=DatePart("n", serverTime);
   utcStruct.Hour=utcStruct.Hour + utcTime.utcHourOffSet;
   utcStruct.Minute=utcStruct.Minute + utcTime.utcMinuteOffSet;
   if (utcStruct.Minute LT 10) utcStruct.Minute = "0" & utcStruct.Minute;
  </cfscript>
  <cfreturn utcStruct.Hour & ":" & utcStruct.Minute>
 </cffunction>
</cfcomponent>
```

The example defines two component methods: `getLocalTime` and `getUTCTime`.

2. Create a ColdFusion page, with the following code and save it in the same directory as the tellTime component:

```
<!--- Create the component instance. --->
<cfobject component="tellTime2" name="tellTimeObj">
<!--- Invoke the methods. --->
<cfinvoke component="#tellTimeObj#" method="getLocalTime"
returnvariable="localTime">
<cfinvoke component="#tellTimeObj#" method="getUTCTime"
returnvariable="UTCTime">
<!--- Display the results. --->
<h3>Time Display Page</h3>
<cfoutput>
 Server's Local Time: #localTime#<br>
 Calculated UTC Time: #UTCTime#
</cfoutput>
```

This example uses the cfobject tag to create an instance of the tellTime component and the `cfinvoke` tag to invoke the instance's `getLocalTime` and `getUTCTime` methods. In this example, the CFC contains the functional logic in the methods, which return a result to the calling page, and the calling page displays the results. This structure separates the logic from the display functions, which usually results in more reusable code.

Invoking component methods transiently

In ColdFusion pages or components, the `cfinvoke` tag can invoke component methods without creating a persistent CFC instance.

© 2014 Adobe Systems Incorporated. All rights reserved.

To invoke a component method transiently, use the `cfinvoke` tag and specify the following:

- The name or path of the component, in the `component` attribute.
- The method name, in the `method` attribute.
- Any parameters. For information on passing parameters, see *Passing parameters to methods by using the cfinvoke tag* in Passing parameters to methods.
- If the component method returns a result, the name of the variable that contains the result, in the `returnVariable` attribute.
  The following procedure creates an application that displays the local time.

1. Create the following component and save it as tellTime.cfc:

```
<cfcomponent>
 <cffunction name="getLocalTime">
  <cfoutput>#TimeFormat(now())#</cfoutput>
 </cffunction>
</cfcomponent>
```

   The example defines a component with one method, `getLocalTime`, that displays the current time.

2. Create a ColdFusion page, with the following code, and save it in the same directory as the tellTime component:

```
<h3>Time Display Page</h3>
<b>Server's Local Time:</b>
<cfinvoke component="tellTime" method="getLocalTime">
```

   Using the `cfinvoke` tag, the example invokes the `getLocalTime` component method without creating a persistent CFC instance.

Using the cfinvoke tag within the CFC definition

You can use the `cfinvoke` tag to invoke a component method within the component definition; for example, to call a utility method that provides a service to other methods in the component. To use the `cfinvoke` tag in this instance, do not create an instance or specify the component name in the `cfinvoke` tag, as the following example shows:

```
<cfcomponent>
 <cffunction name="servicemethod" access="public">
  <cfoutput>At your service...<br></cfoutput>
 </cffunction>
 <cffunction name="mymethod" access="public">
  <cfoutput>We're in mymethod.<br></cfoutput>
  <!--- Invoke a method in this CFC. --->
  <cfinvoke method="servicemethod">
 </cffunction>
</cfcomponent>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

> **Note**
>
> When you invoke a method from within the component definition in which you define the method, do not use the This scope, because this resets the access privileges.

Invoking methods by using dynamic method names

The `cfinvoke` tag is the only way to efficiently invoke different component methods based on variable data (for example, form input). In this case, you use a variable name, such as Form.method, as the value of the `method` attribute. In the following example, the user selects a report from a form:

```
<select name="whichreport">
 <option value="all">Complete Report</option>
 <option value="salary">Salary Information</option>
</select>
```

The `cfinvoke` tag then invokes the appropriate method, based on what the user selected:

```
<cfinvoke component="getdata" method="#form.whichreport#" returnvariable="queryall">
```

Using components directly in CFScript and CFML

You can invoke methods of a component instance directly using CFScript or in CFML tags. To invoke component methods directly, use the CreateObject function or cfobject tag to instantiate the component. Thereafter, use the instance name followed by a period and the method that you are calling to invoke an instance of the method. Always use parentheses after the method name, even if the method does not take any parameters.
You can use this syntax anywhere that you can use a ColdFusion function, such as in `cfset` tags or surrounded by number signs in the body of a cfoutput tag.

Invoking component methods in CFScript

The following example shows how to invoke component methods in CFScript:

```
<!--- Instantiate once and reuse the instance.--->
<cfscript>
 tellTimeObj=CreateObject("component","tellTime");
 WriteOutput("Server's Local Time: " & tellTimeObj.getLocalTime());
 WriteOutput("<br> Calculated UTC Time: " & tellTimeObj.getUTCTime());
</cfscript>
```

In the example, the three CFScript statements do the following:

1. The `CreateObject` function instantiates the tellTime CFC as tellTimeObj.
2. The first WriteOutput function displays text followed by the results returned by the `getLocalTime` method of the tellTimeObj instance.
3. The second `WriteOutput` function displays text followed by the results returned by the `getUTCTime` method of the tellTimeObj instance.

© 2014 Adobe Systems Incorporated. All rights reserved.

In CFScript, you use the method name in standard function syntax, such as `methodName()`.

Invoking component methods in CFML

The following example uses CFML tags to produce the same results as the CFScript example:

```
<cfobject name="tellTimeObj" component="tellTime">
<cfoutput>
 Server's Local Time: #tellTimeObj.getLocalTime()#<br>
 Calculated UTC Time: #tellTimeObj.getUTCTime()#
</cfoutput>
```

Accessing component data directly

You can access data in the component's This scope directly in CFScript and cfset assignment statements. For example, if a user data CFC has a `This.lastUpdated` property, you could have code such as the following:

```
<cfobject name="userDataCFC" component="userData">
<cfif DateDiff("d", userDataCFC.lastUpdated, Now()) GT 30>
 <!--- Code to deal with older data goes here. --->
</cfif>
```

For more information, see *The This scope*.

Invoking CFC methods with forms and URLs

You can invoke CFC methods directly by specifying the CFC in a URL, or by using HTML and CFML form tags. Because all HTTP requests are transient, these methods only let you transiently invoke methods. They do not let you create persistent CFC instances.

Invoking component methods by using a URL

To invoke a component method by using a URL, append the method name to the URL in standard URL query-string, name-value syntax. You can invoke only one component method per URL request, for example:

```
http://localhost:8500/tellTime.cfc?method=getLocalTime
```

Note

To use URL invocation, set the `access` attribute of the `cffunction` tag to `remote`.

To pass parameters to component methods using a URL, append the parameters to the URL in standard URL query-string, name-value pair syntax; for example:

```
http://localhost:8500/corpQuery.cfc?method=getEmp&lastName=camden
```

© 2014 Adobe Systems Incorporated. All rights reserved.

To pass multiple parameters within a URL, use the ampersand character (&) to delimit the name-value pairs; for example:

```
http://localhost:8500/corpQuerySecure.cfc?method=getAuth&store=women&dept=shoes
```

Note

To ensure data security, Adobe strongly recommends that you not pass sensitive information over the web using URL strings. Potentially sensitive information includes all personal user information, including passwords, addresses, telephone numbers, and so on.

If a CFC method that you access using the URL displays output directly, the user's browser shows the output. You can suppress output by specifying `output="No"` in the `cffunction` tag. If the CFC returns a result using the cfreturn tag, ColdFusion converts the text to HTML edit format (with special characters replaced by their HTML escape sequences), places the result in a WDDX packet, and includes the packet in the HTML that it returns to the client.

Invoking component methods by using a form

To invoke a method by using a ColdFusion or HTML form, do the following:

- Specify the CFC filename or path in the `form` or cfform tag `action` attribute.
- Specify the CFC method in a hidden form field, as follows:

```
<form action="myComponent.cfc" method="POST">.
 <input type="Hidden" name="method" value="myMethod">
```

- Alternatively, if you use the POST method to submit the form, you can follow the filename with `?method=`*methodname*, where *methodname*is the name of the CFC method, as shown in the following line. You cannot use this technique with the GET method.

```
<form action="myComponent.cfc?method=myMethod" method="POST">.
```

- Create an input tag for each component method parameter. The `name` attribute of the tag must be the method parameter name and the field value is the parameter value.
- Specify the `access="remote"` attribute in the cffunction tag that defines the CFC method being invoked
  If the CFC method that you invoke from the form displays output directly, the user's browser shows the output. (You can use the cffunction tag `output` attribute to disable displaying output.) If the CFC returns a result using the cfreturn tag, ColdFusion converts the text to HTML edit format, places it in a WDDX packet, and includes the packet in the HTML that it returns to the client.

1. Create a corpFind.cfm file with the following contents:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<h2>Find People</h2>
<form action="components/corpQuery.cfc?method=getEmp" method="post">
 <p>Enter employee's last Name:</p>
 <input type="Text" name="lastName">
 <input type="Hidden" name="method" value="getEmp">
 <input type="Submit" title="Submit Query"><br>
</form>
```

In the example, the `form` tag's `action` attribute points to the `corpQuery` component and invokes the `getEmp` method.

2. Create a corpQuery.cfc file, specifying `access="remote"` for each `cffunction`tag, as the following example shows:

```
<cfcomponent>
 <cffunction name="getEmp" access="remote">
  <cfargument name="lastName" required="true">
  <cfset var empQuery="">
   <cfquery name="empQuery" datasource="cfdocexamples">
    SELECT LASTNAME, FIRSTNAME, EMAIL
    FROM tblEmployees
   WHERE LASTNAME LIKE '#arguments.lastName#'
   </cfquery>
   <cfoutput>Results filtered by #arguments.lastName#:</cfoutput><br>
   <cfdump var=#empQuery#>
 </cffunction>
</cfcomponent>
```

3. Open a web browser and enter the following URL:

```
http://localhost/corpFind.cfm
```

ColdFusion displays the search form. After you enter values and click the Submit Query button, the browser displays the results.

Accessing CFCs from outside ColdFusion and basic HTML

Flash applications that use Flash Remoting can easily take advantage of ColdFusion components for business logic. Similarly, you can export CFCs so that any application can access CFC methods as web services.
For ColdFusion component methods to communicate with Flash Remoting applications, set the `access` attribute of the `cffunction` tag to `remote`.
For more information on creating CFCs for Flash Remoting, see Using the Flash Remoting Service
Any application, whether it is a ColdFusion application, a Java application, JSP page, or a .Net application, can access well-formed ColdFusion components as web services by referencing the WSDL file that ColdFusion automatically generates.
To see a component's WSDL definition, specify the component web address in a URL, followed by ?wsdl; for example:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
http://localhost:8500/MyComponents/arithCFC.cfc?wsdl
```

For more information on using CFCs as web services, see Using Web Services

Specifying the CFC location

When you instantiate or invoke a component, you can specify the component name only, or you can specify a *qualified* **path. To specify a qualified path, separate the directory names with periods, not backslashes. For example, myApp.cfcs.myComponent specifies the component defined in myApp\cfcs\myComponent.cfc. For additional information, see *Saving and naming ColdFusion components* in Building ColdFusion components.
ColdFusion uses the following rules to find the specified CFC:

- If you use a `cfinvoke` or cfobject tag, or the CreateObjectfunction, to access the CFC from a CFML page, ColdFusion searches directories in the following order:
  1. Local directory of the calling CFML page
  2. Web root
  3. Directories specified on the Custom Tag Paths page of ColdFusion Administrator
- If you specify only a component name, ColdFusion searches each of these directories, in turn, for the component.
- If you specify a qualified path, such as myApp.cfcs.myComponent, ColdFusion looks for a directory matching the first element of the path in each of these directories (in this example, myApp). If ColdFusion finds a matching directory, it looks for a file in the specified path beneath that directory, such as myApp\cfcs\myComponent.cfc, relative to each of these directories.

  Note

  If ColdFusion finds a directory that matches the first path element, but does not find a CFC under that directory, ColdFusion returns a not found_ error and does_ not_ search for another directory._

- If you invoke a CFC method remotely, using a specific URL, a form field, Flash Remoting, or a web service invocation, ColdFusion looks in the specified path relative to the web root. For form fields and URLs that are specified directly on local web pages, ColdFusion also searches relative to the page directory.

  Note

  On UNIX and Linux systems, ColdFusion attempts to match a CFC name or custom tag name with a filename, as follows: First, it attempts to find a file with the name that is all lowercase. If it fails, it tries to find a file whose case matches the CFML case. For example, if you specify `<cfobject name="myObject" Component="myComponent">`*, ColdFusion first looks for mycomponent.cfc and, if it doesn't find it, ColdFusion looks for myComponent.cfc.*

© 2014 Adobe Systems Incorporated. All rights reserved.

Passing parameters to methods

You pass parameters to a method in a CFC by using the cfinvoke tag, direct method invocations, or by passing parameters in a URL.

Passing parameters to methods by using the cfinvoke tag

When you use the `cfinvoke` tag, ColdFusion provides several methods for passing parameters to CFC methods:

- As `cfinvoke` tag attributes, in `name="value"` format
- In the `cfinvoke` tag `argumentcollection` attribute
- In the `cfinvoke` tag body, using the cfinvokeargument tag

You can use any combination of these methods in a single invocation. If you use the same name in two or three of these methods, ColdFusion uses the value based on the following order of precedence:

1. `cfinvokeargument` tags
2. `cfinvoke` attribute name-value pairs
3. `argumentcollection` arguments

Passing parameters by using attribute format

You can pass parameters in the `cfinvoke` tag as tag attribute name-value pairs, as the following example shows:

```
<cfinvoke component="authQuery" method="getAuthSecure"
 lastName="#session.username#" pwd="#url.password#">
```

In the example, the parameters are passed as the `lastName` and pwd attributes.

> **Note**
>
> The `cfinvoke` tag attribute names are reserved and cannot be used for parameter names. The reserved attribute names are: `component`, `method`, `argumentCollection`, and `returnVariable`. For more information, see the CFML Reference.

Passing parameters in the argumentCollection attribute

If you save attributes to a structure, you can pass the structure directly using the `cfinvoke` tag's `argumentCollection` attribute. This technique is useful if an existing structure or scope (such as the Forms scope) contains values that you want to pass to a CFC as parameters, and for using conditional or looping code to create parameters. When you pass an argumentCollection structure, each structure key is the name of a parameter inside the structure.

The following example passes the Form scope to the `addUser` method of the UserDataCFC component. In the method, each form field name is a parameter name; the method can use the contents of the form fields to add a user to a database.

```
<cfinvoke component="UserDataCFC" method="addUser" argumentCollection="#Form#">
```

Passing parameters by using the cfinvokeargument tag

© 2014 Adobe Systems Incorporated. All rights reserved.

To pass parameters in the `cfinvoke` tag body, use the `cfinvokeargument` tag. Using the `cfinvokeargument` tag, for example, you can build conditional processing that passes a different parameter based on user input.
The following example invokes the `corpQuery` component:

```
<cfinvoke component="corpQuery" method="getEmp">
 <cfinvokeargument name="lastName" value="Wilder">
</cfinvoke>
```

The `cfinvokeargument` tag passes the `lastName` parameter to the component method.
In the following example, a form already let the user select the report to generate. After instantiating the `getdata` and `reports` components, the action page invokes the `doquery` component instance, which returns the query results in `queryall`. The action page then invokes the `doreport` component instance and uses the `cfinvokeargument` tag to pass the query results to the `doreport` instance, where the output is generated.

```
<cfobject component="getdata" name="doquery">
<cfobject component="reports" name="doreport">
 <cfinvoke component="#doquery#" method="#form.whichreport#"
returnvariable="queryall">
 <cfinvoke component="#doreport#"method="#form.whichreport#">
  <cfinvokeargument name="queryall" value="#queryall#">
 </cfinvoke>
```

Passing parameters in direct method invocations

ColdFusion provides three methods for passing parameters to CFC methods in direct method invocations:

1. You can pass the parameters the form of comma-separated `name="value"`entries, as in the following CFScript example:

   ```
   authorized = securityCFC.getAuth(name="Almonzo", Password="LauRa123");
   ```

2. You can pass the parameters in an `argumentCollection`structure. The following code is equivalent to the previous example:

   ```
   argsColl = structNew();
   argsColl.username = "Almonzo";
   argsColl.password = "LauRa123";
   authorized = securityCFC.getAuth(argumentCollection = argsColl);
   ```

3. You can pass positional parameters to a method by separating them with commas. The following example calls the `getAuth`method, and passes the name and password as positional parameters:

   ```
   authorized = securityCFC.getAuth("Almonzo", "LauRa123");
   ```

© 2014 Adobe Systems Incorporated. All rights reserved.

Note

For more information on using positional parameters and component methods in ColdFusion functions, see Creating user-defined functions.

Passing parameters in a URL

ColdFusion lets you pass parameters to CFC methods in a URL. To do so, you append the URL in standard URL query-string, name-value pair syntax; for example:

```
http://localhost:8500/CompanyQuery.cfc?method=getEmp&lastName=Adams
```

© 2014 Adobe Systems Incorporated. All rights reserved.

Using CFCs effectively

Several techniques let you effectively use CFCs in your applications:

- Structure and reuse code
- Build secure CFCs
- Use introspection to get information about components

Structuring and reusing code

Component inheritance and the Super keyword are two important tools for creating structured, object-oriented ColdFusion components.

- **Component inheritance** Lets you create a single base component and reuse this code in multiple subclasses that are derived from the base component. Typically a base component is more general, and subcomponents are typically more specific. Each subclass does not have to redefine the code in the base component, but can override it if necessary.
- **The Super keyword** Lets a component that overrides a base component method execute the original base component method. This technique lets your subclassed component override a method without losing the ability to call the original version of the method.

Using component inheritance

Component inheritance lets you import component methods and properties from one component to another component. Inherited components share any component methods or properties that they inherit from other components, and ColdFusion initializes instance data in the parent CFC when you instantiate the CFC that extends it.
Component inheritance defines an *is a* relationship between components. For example, a component named president.cfc inherits its methods and properties from manager.cfc, which inherits its methods and properties from employee.cfc. In other words, president.cfc *is a* manager.cfc; manager.cfc *is an* employee.cfc; and president.cfc *is an* employee.cfc.
In this example, employee.cfc is the *base* component; it's the component upon which the others are based. The manager component extends the employee component; it has all the methods and properties of the employee component, and some additional ones. The president component extends the manager component. The president component is called a subcomponent or child component of the manager component, which, in turn, is a child component of the employee component.

1. Create the employee.cfc file with the following content:

```
<cfcomponent>
 <cfset This.basesalary=40*20>
</cfcomponent>
```

1. Create the manager.cfc file with the following content:

```
<cfcomponent extends="employee">
 <cfset This.mgrBonus=40*10>
</cfcomponent>
```

   In the example, the cfcomponent tag's `extends` attribute points to the `employee` component.

2. Create the president.cfc file with the following content:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cfcomponent extends="manager">
 <cfset This.prezBonus=40*20>
</cfcomponent>
```

In the example, the cfcomponent tag's `extends` attribute points to the `manager` component.

3. Create the inherit.cfm file with the following content, and save it in the same directory as the components you created in the previous steps:

```
<cfobject name="empObj" component="employee">
<cfobject name="mgrObj" component="manager">
<cfobject name="prezObj" component="president">
<cfoutput>
 An employee's salary is #empObj.basesalary# per week.<br>
 A manager's salary is #mgrObj.basesalary + mgrObj.mgrBonus# per week.<br>
 A president's salary is #prezObj.basesalary + prezObj.mgrBonus +
  prezObj.PrezBonus# per week.
</cfoutput>
```

When you browse the inherit.cfm file, the `manager` component refers to the `basesalary` defined in employee.cfc, which is the base component; the `president` component refers to both the `basesalary` defined in the `employee` component, and the `mgrBonus` defined in the `manager` component. The `manager` component is the parent class of the `president` component.

Using the component.cfc file

All CFCs automatically extend the ColdFusion WEB-INF/cftags/component.cfc component. (The WEB-INF directory is in the *cf_root*/wwwroot directory on ColdFusion configured with an embedded J2EE server. It is in the *cf_root* dire ctory when you deploy ColdFusion on a J2EE server.) This CFC is distributed as a zero-length file. You can use it for any core methods or properties that you want *all* CFCs in your ColdFusion application server instance to inherit.

> **Note**
>
> When you install a newer version of ColdFusion, the installation procedure replaces the existing component.cfc file with a new version. Therefore, before upgrading, save any code that you have added to the component.cfc file, and then copy the code into the new component.cfc file.

Using the Super keyword

You use the Super keyword only on CFCs that use the `Extends` attribute to extend another CFC. Unlike ColdFusion scopes, the Super keyword is not used for variables; it is only used for CFC methods, and it is not available on ColdFusion pages that invoke CFCs.
The Super keyword lets you refer to versions of methods that are defined in the CFC that the current component extends. For example, the employee, manager, and president CFCs each contain a `getPaid` method. The manager CFC extends the employee CFC. Therefore, the manager CFC can use the original versions of the overridden `getPaid` method, as defined in the employee CFC, by prefixing the method name with Super.

1. Create the employee.cfc file with the following content:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cfcomponent>
 <cffunction name="getPaid" returntype="numeric">
  <cfset var salary=40*20>
  <cfreturn salary>
 </cffunction>
</cfcomponent>
```

1. Create the manager.cfc file with the following content:

```
<cfcomponent extends="employee">
 <cffunction name="getPaid" returntype="numeric">
  <cfset var salary=1.5 * Super.getPaid()>
  <cfreturn salary>
 </cffunction>
</cfcomponent>
```

1. Create the president.cfc file with the following content:

```
<cfcomponent extends="manager">
 <cffunction name="getPaid" returntype="numeric">
  <cfset var salary=1.5 * Super.getPaid()>
  <cfreturn salary>
 </cffunction>
</cfcomponent>
```

1. Create the payday.cfm file with the following content, and save it in the same directory as the components that you created in the previous steps:

```
<cfobject name="empObj" component="employee">
<cfobject name="mgrObj" component="manager">
<cfobject name="prezObj" component="president">
<cfoutput>
 <cfoutput>
  An employee earns #empObj.getPaid()#.<br>
  A manager earns #mgrObj.getPaid()#.<br>
  The president earns #prezObj.getPaid()#.
 </cfoutput>
</cfoutput>
```

In this example, each `getPaid` method in a child component invoked the `getPaid` method of its parent component. The child's `getPaid` method then used the salary returned by the parent's `getPaid` method to calculate the appropriate amount.
Included pages can use the Super keyword.

© 2014 Adobe Systems Incorporated. All rights reserved.

> **Note**
>
> The Super keyword supports only one level of inheritance. If you use multiple levels of inheritance, you can only use the Super keyword to access the current component's immediate parent. The example in this section illustrates handling this limitation by invoking methods in a chain.

Using component packages

Components stored in the same directory are members of a component *package*. Component packages help prevent naming conflicts, and facilitate easy component deployment; for example:

- ColdFusion searches the current directory first for a CFC. If you place two components in a single directory as a package, and one component refers to the other with only the component name, not a qualified path, ColdFusion always searches the package directory first for the component. As a result, if you structure each application's components into a package, your applications can use the same component names without sharing the component code.
- If you use the `access="package"` attribute in a method's cffunction tag, access to the method is limited to components in the same package. Components in other packages cannot use this method, even if they specify it with a fully qualified component name. For more information on access security, see *Using access security* in Using CFCs effectively.

Invoke a packaged component method with the cfinvoke tag

1. In your web root directory, create a directory named *appResources*.
2. In the appResources directory, create a directory named *components*.
3. Copy the tellTime2.cfc file you created in *Invoking methods of a CFC instance* in Using ColdFusion components-Developing guide and the getUTCTime.cfm file that you created in *Placing executable code in a separate file* in Building ColdFusion components to the components directory.
4. Create the timeDisplay.cfm file with the following content and save it in your web root directory:

```
<!--- Create the component instance. --->
<cfobject component="appResources.components.tellTime2" name="tellTimeObj">
<!--- Invoke the methods. --->
<cfinvoke component="#tellTimeObj#" method="getLocalTime"
 returnvariable="localTime" >
<cfinvoke component="#tellTimeObj#" method="getUTCTime"
 returnvariable="UTCTime" >
<!--- Display the results. --->
<h3>Time Display Page</h3>
<cfoutput>
 Server's Local Time: #localTime#<br>
 Calculated UTC Time: #UTCTime#
</cfoutput>
```

   You use dot syntax to navigate directory structures. Place the directory name before the component name.

5. Browse the timeDisplay.cfm file in your browser.
   The following example shows a CFScript invocation:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cfscript>
helloCFC = createObject("component", "appResources.components.catQuery");
helloCFC.getSaleItems();
</cfscript>
```

The following example shows a URL invocation:

```
http://localhost/appResources/components/catQuery.cfc?method=getSalesItems
```

Using CFCs in persistent scopes

You can place a CFC instance in the Session or Application scope. This way, the component properties continue to exist while the scope persists. For example, if you use a CFC for a shopping cart application, where the shopping cart contents must persist for the length of the user's session. If you place the shopping cart CFC in the Session scope, you can use component properties to store the cart contents. For example, the following line creates an instance of the `shoppingCart` component in the Session scope:

```
<cfobject name="Session.myShoppingCart" component="shoppingCart">
```

Code that manipulates persistent scope CFC properties must be locked, just as all other code that manipulates persistent scope properties must be locked. Therefore, lock both of the following types of application code:

- Code that directly manipulates properties of a persistent scope CFC instance
- Code that calls methods of a persistent scope CFC instance that manipulate properties of the instance
  If you place multiple CFC instances in a single persistent scope, you can create a named lock for each CFC instance. For more information on locking, see Using Persistent Data and Locking.

> **Note**
>
> Session scope CFCs cannot be serialized, so you cannot use them with clustered sessions; for example, if you want to support session failover among servers.

Building secure ColdFusion components

To restrict access to component methods, ColdFusion components use access, role-based, or programmatic security.

Using access security

CFC access security lets you limit the code that can access the components. You specify the access to a CFC method by specifying the cffunction `access` attribute, as follows:

Type	Description

© 2014 Adobe Systems Incorporated. All rights reserved.

private	Available only to the component that declares the method and any components that extend the component in which it is defined. This usage is like the Java protected keyword, *not* the Java private keyword.
package	Available only to the component that declares the method, components that extend the component, or any other components in the package. A package consists of all components defined in a single directory. For more information on packages, see *Using component packages* in Using CFCs effectively.
public	Available to any locally executing ColdFusion page or component method.
remote	Available to a locally or remotely executing ColdFusion page or component method, or to a local or remote client through a URL, form submission, Flash Remoting, or as a web service.

Using role-based security

If you specify a `roles` attribute in a `cffunction` tag, only users who are logged in with one of the specified roles can execute the method. When a user tries to invoke a method without authorization, an exception is returned.
The following example creates a component method that deletes files:

```
<cfcomponent>
 <cffunction
      name="deleteFile" access="remote" roles="admin,manager" output="no">
  <cfargument name="filepath" required="yes">
  <cffile action="DELETE" file=#arguments.filepath#>
 </cffunction>
</cfcomponent>
```

In the example, the `cffunction` tag includes the `roles` attribute to specify the user roles allowed to access it. In this example, only users in the role `admin` and `manager` can access the function. Notice that multiple roles are delimited by a comma.
For information on ColdFusion security, including the cflogin tag and role-based security in ColdFusion, see Securing Applications.

Using programmatic security

You can implement your own security within a method to protect resources. For example you can use the ColdFusion function `IsUserInAnyRole` to determine if a user is in particular role, as the following example shows:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cffunction name="foo">
 <cfif IsUserInRole("admin")>
   do stuff allowed for admin
 <cfelseif IsUserInRole("user")>
   do stuff allowed for user
 <cfelse>
  <cfoutput>unauthorized access</cfoutput>
  <cfabort>
 </cfif>
</cffunction>
```

Using introspection to get information about components

ColdFusion provides several ways for you to get information about components:

- Request a component page from the browser
- Use the ColdFusion component browser
- Use the Adobe Dreamweaver Components panel
- Use the `GetMetaData` function
  Development teams can use the information about components as up-to-date API reference information.

> **Note**
>
> For information about how to include documentation in CFCs for display by using introspection, see *Documenting CFCs* in Building ColdFusion components.

Requesting a component page from the browser

When you access a CFC directly with a web browser without specifying a component method, the following chain of events occurs:

1. The request is redirected to the cfcexplorer.cfc file, which is located in the *cf_root*/wwwroot/CFIDE/componentutils directory.
2. The cfcexplorer component prompts users for the ColdFusion RDS or Administrator password, if necessary.
3. The cfcexplorer component renders an HTML description and returns it to the browser.

Using the ColdFusion component browser

You can also browse the components available in ColdFusion using the component browser, which is located at *cf_root*/wwwroot/CFIDE/componentutils/componentdoc.cfm.
The browser has three panes:

- The upper-left pane lists all CFC packages that ColdFusion can access, and has all components and refresh links.
- The lower-left pane lists CFC component names. When the browser first appears, or when you click the all components link in the upper pane, the lower pane lists all available components. If you click a package name in the upper left pane, the lower pane lists only the components in the package.
- The right pane initially lists the paths of all components. When you click a component name in the lower-left pane, the right pane shows the ColdFusion introspection page, as described in *Requesting a component page from the browser* in Using CFCs effectively.

© 2014 Adobe Systems Incorporated. All rights reserved.

> **Note**
>
> When RDS user names are enabled, the component browser accepts the root administrator user (admin) with either the administrator or RDS single password.

Using the Dreamweaver Components panel

The Dreamweaver Components panel lists all available components, including their methods, method parameters, and properties. The panel's context menu includes options to create a component, edit the selected component, insert code to invoke the component, or show detailed information on the component or component element. The Get description option shows the ColdFusion introspection page, as described in *Requesting a component page from the browser* in Using CFCs effectively. For more information on viewing and editing CFCs in Dreamweaver, see the Dreamweaver online Help.

Using the GetMetaData function

The CFML GetMetaData function returns a structure that contains all the metadata of a CFC instance. This structure contains substantially more data about the CFC than the cfdump tag shows, and includes the following information:

- All attributes to the component tag, including any metadata-only attributes, plus the component path.
- An array of structures that contains complete information on each method (function) in the component. This information describes all attributes, including metadata-only function and parameter attributes.
- Within each function structure, a Parameters element that contains an array of parameters specified by cfargument tags. Information on each parameter includes any metadata-only attributes.
- Information about any properties that are specified using the cfproperty tag.

Display metadata for a CFC

1. Create the tellAboutCfcs.cfm file in the same directory as the telltime.cfc file, with the following code:

```
<!--- Create an instance of the component. --->
<cfobject component="tellTime" name="tellTimeObj">
<!--- Create a structure. --->
<cfset aboutcfc=structNew()>
<!--- Populate the structure with the metadata for the
   tellTimeObj instance of the tellTime CFC. --->
<cfset aboutcfc=GetMetaData(tellTimeObj)>
<cfdump var="aboutcfc">
```

2. View the tellAboutCfcs.cfm file in a browser.

3. For information on how to specify CFC metadata, including how to use component tags and how to specify metadata-only attributes, see *Documenting CFCs* in Building ColdFusion components.

© 2014 Adobe Systems Incorporated. All rights reserved.

ColdFusion component example

Several code examples in the *Developing ColdFusion Applications* reuse code, particularly queries. To illustrate the advantages of CFCs, these examples invoke the appropriate method in the CFC that appears in the following example. Although Adobe recommends using CFCs to create structured, reusable code, some code examples in this manual contain queries within a CFML page, rather than invoking a CFC, in order to clearly illustrate a particular element of ColdFusion.

```
<cfcomponent>
 <cffunction name="allemployees" access="public" output="false"
  returntype="query">
  <cfset var getNames="">
  <cfquery name="getNames" datasource="cfdocexamples">
   SELECT * FROM Employee
  </cfquery>
 </cffunction>

 <cffunction name="namesalarycontract" access="public" output="false"
  returntype="query">
  <cfset var EmpList="">
  <cfquery name="EmpList" datasource="cfdocexamples">
   SELECT Firstname, Lastname, Salary, Contract
   FROM Employee
  </cfquery>
 </cffunction>

 <cffunction name="fullname" access="public" output="false"
  returntype="query">
  <cfset var engquery="">
  <cfquery name="engquery" datasource="cfdocexamples">
   SELECT FirstName || ' ' || LastName AS FullName
   FROM Employee
  </cfquery>
 </cffunction>

 <cffunction name="bydept" access="public" output="false" returntype="query">
  <cfset var deptquery="">
  <cfquery name="deptquery" datasource="cfdocexamples">
   SELECT Dept_ID, FirstName || ' ' || LastName
   AS FullName
   FROM Employee
   ORDER BY Dept_ID
  </cfquery>
 </cffunction>

 <cffunction name="employeebyURLID" access="public" output="false"
  returntype="query">
  <cfset var GetRecordtoUpdate="">
  <cfquery name="GetRecordtoUpdate" datasource="cfdocexamples">
   SELECT * FROM Employee
   WHERE Emp_ID = #URL.Emp_ID#
  </cfquery>
 </cffunction>

 <cffunction name="deleteemployee" access="public" output="false"
  returntype="void">
  <cfset var DeleteEmployee="">
```

© 2014 Adobe Systems Incorporated. All rights reserved.

```
  <cfquery name="DeleteEmployee" datasource="cfdocexamples">
   DELETE FROM Employee
   WHERE Emp_ID = #Form.Emp_ID#
  </cfquery>
 </cffunction>

 <cffunction name="distinctlocs"access="public" output="false"
  returntype="query">
  <cfset var GetDepartments="">
  <cfquery name="GetDepartments" datasource="cfdocexamples">
   SELECT DISTINCT Location
   FROM Departmt
  </cfquery>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

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

© 2014 Adobe Systems Incorporated. All rights reserved.

Implicit constructor for CFC

You can initialize the CFC properties, when you instantiate the CFC, by passing the values as key-value pair or struct.
For example, assume that you have a CFC employee.cfc with the properties `firstName` and `lastName`.
You can use the following syntax to set the first name and last name:

```
emp=new employee(firstName="Tom", lastName="Nash");
```

or as an implicit struct as follows:

```
emp=new Employee({firstName="Tom", lastName="Nash"});
```

If an `init()` method is defined for the CFC, either by explicitly defining an {{init()}}method or by providing the {{}}method in the component's {{initmethod }}attribute, implicit Constructor is not invoked.
For implicit Constructor support, either specify `accessors=true` in the `cfcomponent` or ensure that you have the setter functions for the properties defined.
If you have `setter = false` for a specific property or if the property is not defined for the CFC, then the value is not set.
In either case, if there is an `onMissingMethod` defined for the CFC, it is invoked.

Example 1

In this example, the property `firstName` is set to the value `Tom` at the time of CFC instantiation, but the property `lastName` is not set because it has `setter = false`. Also, `onMissingMethod` is invoked in the case of the property `lastName`.employee.cfm

```
<cfscript>
 emp = new employee(firstname="Tom", lastname="Nash");
 writeOutput("<u><b>Employee Details</b></u>: " & "<br><br>");
 writeOutput("First Name: " & emp.getFirstname() & "<br>");
</cfscript>
```

employee.cfc

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cfcomponent accessors="TRUE">
 <cfproperty name="firstname" type="string" setter="true"/>
 <cfproperty name="lastname" type="string" setter="false"/>
 <cfproperty name="age" type="numeric"/>
 <cffunction name="onMissingMethod">
  <cfargument name="missingMethodName"/>
  <cfargument name="missingMethodArguments"/>

  <cfoutput>
   onMissingMethod() called for method call - #arguments.missingMethodName#
   <hr>
  </cfoutput>
 </cffunction>
</cfcomponent>
```

Example 2

In this example, both employee1.cfc and employee2.cfc have an `init()}}method defined. Therefore, the CFC properties {{firstName` and `lastName` are not initialized. But in the employee.cfc, since there is no `init()` method defined, property `firstName` is initialized.employee1.cfc

```
<cfcomponent accessors="TRUE">
 <cfproperty name="firstname" type="string" setter="true"/>
 <cfproperty name="lastname" type="string" setter="false"/>
 <cffunction name="init">
  <cfreturn this>
 </cffunction>
 <cffunction name="onMissingMethod">
  <cfargument name="missingMethodName"/>
  <cfargument name="missingMethodArguments"/>
  <cfoutput>
   onMissingMethod() called for method call - #arguments.missingMethodName#
   <hr>
  </cfoutput>
 </cffunction>
</cfcomponent>
```

employee2.cfc

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cfcomponent accessors="TRUE" initmethod=foo>
 <cfproperty name="firstname" type="string" setter="true"/>
 <cfproperty name="lastname" type="string" setter="false"/>
 <cffunction name="foo">
  <cfreturn this>
 </cffunction>
 <cffunction name="onMissingMethod">
  <cfargument name="missingMethodName"/>
  <cfargument name="missingMethodArguments"/>
  <cfoutput>
   onMissingMethod() called for method call - #arguments.missingMethodName#
   <hr>
  </cfoutput>
 </cffunction>
</cfcomponent>
```

employee.cfm

```
<cfscript>
 emp = new employee(firstname="Tom", lastname="Nash");
 writeOutput("<u><b>Employee Details</b></u>: " & "<br><br>");
 writeOutput("First Name: " & emp.getFirstname() & "<br>");
 writeOutput("<hr>");
 emp1 = new employee1(firstname="Tom", lastname="Nash");
 writeOutput("First Name: " & emp1.getFirstname() & "<br>");
 writeOutput("<hr>");
 emp2 = new employee2(firstname="Tom", lastname="Nash");
 writeOutput("First Name: " & emp2.getFirstname() & "<br>");
 writeOutput("<hr>");
</cfscript>
<cfoutput>
 onMissingMethod() called for method call - #arguments.missingMethodName#
 <hr>
</cfoutput>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

Method chaining for CFC methods

You can chain your CFC methods as follows:

```
emp=new employee();
emp.setFirstName("Tom").setLastName("Nash").setAge("30");
```

Chaining works only

- If the attribute `accessors` is set to `true`
- Until a method is found
- If there are no errors
  or
- If the setter functions for the properties are defined

Example

The chain in the example works only until `lastName`. This is because `setter` for `age` is set to `false`:

```
<cfcomponent accessors="TRUE">
 <cfproperty name="firstname" type="string" setter="true"/>
 <cfproperty name="lastname" type="string" setter="true"/>
 <cfproperty name="age" type="numeric" setter="false"/>
 <cffunction name="init">
  <cfreturn this>
 </cffunction>
</cfcomponent>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

CFC Implicit notation

This feature lets you set your CFC property as simple variable assignment, without specifying the setters. That is, you can set the property by setting the property field rather than by invoking the setter method for the property. Similarly, the property value can be accessed by referencing the property name, rather than by invoking the getter for the property.

Example

Application.cfc

```
component
{
 this.name = "MyApplication";
 this.invokeImplicitAccessor = true;
}
```

employee.cfm

```
<cfscript>
 emp = new emp();
 emp.firstname = "Tom";
 emp.lastname = "Nash";
 emp.age = 30;
 writeOutput("First Name = " & emp.firstname & "<br>");
 writeOutput("last Name = " & emp.emp.lastname & "<br>");
 writeOutput("Age = " & emp.age & "<br>");
</cfscript>
```

employee.cfc

```
<cfcomponent accessors="TRUE">
 <cfproperty name="firstname" type="string" setter="true"/>
 <cfproperty name="lastname" type="string" setter="false"/>
 <cfproperty name="age" type="numeric"/>
 <cffunction name="onMissingMethod">
  <cfargument name="missingMethodName"/>
  <cfargument name="missingMethodArguments"/>
  <cfoutput>
   onMissingMethod() called for method call -#arguments.missingMethodName#
   <hr>
  </cfoutput>
 </cffunction>
</cfcomponent>
```

In the example, CFC Implicit notation works only if you set `invokeImplicitAccessor` in the Application.cfc to `true`. Otherwise, the values are posted to the This scope of the component.

© 2014 Adobe Systems Incorporated. All rights reserved.

Creating and Using Custom CFML Tags

You can extend CFML by creating and using custom CFML tags that encapsulate common code.

© 2014 Adobe Systems Incorporated. All rights reserved.

Creating custom tags

Custom tags let you extend CFML by adding your own tags to the ones supplied with ColdFusion. After you define a custom tag, you can use it on a ColdFusion page just as you would any of the standard CFML tags, such as `cfquery` and `cfoutput`.
You use custom tags to encapsulate your application logic so that it can be referenced from any ColdFusion page. Custom tags allow for rapid application development and code reuse while offering off-the-shelf solutions for many programming chores.
For example, you can create a custom tag, named `cf_happybirthday`, to generate a birthday message. You could then use that tag in a ColdFusion page, as follows:

```
<cf_happybirthday name="Ted Cantor" birthDate="December 5, 1987">
```

When ColdFusion processes the page containing this tag, it could output the message:

```
December 5, 1987 is Ted Cantor's Birthday.
Please wish him well.
```

A custom tag can also have a body and end tag, for example:

```
<cf_happybirthdayMessge name="Ellen Smith" birthDate="June 8, 1993">
 <p> Happy Birthday Ellen!</p>
 <p> May you have many more!</p>
</cf_happybirthdayMessge>
```

This tag could output the message:

```
June 8, 1993 is Ellen Smith's Birthday.
Happy Birthday Ellen!
May you have many more!
```

For more information about using end tags, see *Handling end tags* in Executing custom tags.

Creating and calling custom tags

You implement a custom tag with a single ColdFusion page. You then call the custom tag from a ColdFusion page by inserting the prefix `cf_` before the page's filename. The page that references the custom tag is referred to as the *calling* page.

1. Create a ColdFusion page, the custom tag page, that shows the current date:

```
<cfoutput>#DateFormat(Now())#</cfoutput>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

1. Save the file as date.cfm.
2. Create a ColdFusion page, the calling page, with the following content:

```
<html>
<head>
 <title>Date Custom Tag</title>
</head>
<body>

 <!--- Call the custom tag defined in date.cfm --->
 <cf_date>

</body>
</html>
```

1. Save the file as callingdate.cfm.
2. View callingdate.cfm in your browser. This custom tag returns the current date in the format DD-MMM-YY.
   As you can see from this example, creating a custom tag in CFML is no different from writing any ColdFusion page. You can use all CFML constructs, as well as HTML. You are free to use any naming convention that fits your development practice. Unique descriptive names make it easy for you and others to find the right tag.

Note

Although tag names in ColdFusion pages are not case sensitive, custom tag filenames must be lowercase on UNIX.

Storing custom tag pages

You must store custom tag pages in any one of the following:

- The same directory as the calling page
- The cfusion\CustomTags directory
- A subdirectory of the cfusion\CustomTags directory
- A directory that you specify in the ColdFusion Administrator
  To share a custom tag among applications in multiple directories, place it in the cfusion\CustomTags directory. You can create subdirectories to organize custom tags. ColdFusion searches recursively for the Custom Tags directory, stepping down through any existing subdirectories until the custom tag is found.
  You can have a situation where you have multiple custom tags with the same name. To guarantee which tag ColdFusion calls, copy it to the same directory as the calling page. Or, use the `cfmodule` tag with the `template` attribute to specify the absolute path to the custom tag. For more information on `cfmodule`, see the next section.

Calling custom tags with the cfmodule tag

You can also use the `cfmessagebox` tag to call custom tags if you want to specify the location of the custom tag page. The `cfmodule` tag is useful if you are concerned about possible name conflicts when using a custom tag, or if the application must use a variable to dynamically call a custom tag at runtime.
Use either a `template` or `name` attribute in the tag, but you cannot use both. The following table describes the basic `cfmodule` attributes:

Attribute	Description

© 2014 Adobe Systems Incorporated. All rights reserved.

`template`	Required if the `name` attribute is not used. Same as the `template` attribute in cfinclude. This attribute: • Specifies a path relative to the directory of the calling page. • If the path value is prefixed with "/", ColdFusion searches directories explicitly mapped in the ColdFusion Administrator for the included file. Example: `<cfmodule template="../MyTag.cfm">` identifies a custom tag file in the parent directory.
`name`	Required if the `template` attribute is not used. Use period-separated names to uniquely identify a subdirectory under the CustomTags root directory.Example: `<cfmodule name="MyApp.GetUserOptions">` identifies the file GetUserOptions.cfm in the CustomTags\MyApp directory under the ColdFusion root directory.
`attributes`	The custom tag's attributes.

For example, the following code specifies to execute the custom tag defined by the mytag.cfm page in the parent directory of the calling page:

```
<cfmodule template="../mytag.cfm">
```

For more information on using the `cfmessagebox` tag, see the *CFML Reference*.

Calling custom tags with the cfimport tag

You can use the `cfimport` tag to import custom tags from a directory as a tag library. The following example imports the tags from the directory myCustomTags:

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

Once imported, you call the custom tags using the prefix that you set when importing, as the following example shows:

```
<mytags:customTagName>
```

where customTagName corresponds to a ColdFusion application page named customTagName.cfm. If the tag takes attributes, you include them in the call:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<mytags:custom_tag_name attribute1=val_1 attribute2=val_2>
```

You can also include end tags when calling your custom tags, as the following example shows:

```
<mytags:custom_tag_name attribute1=val_1 attribute2=val_2>
...
</mytags:custom_tag_name>
```

ColdFusion calls the custom tag page twice for a tag that includes an end tag: once for the start tag and once for the end tag. For more information on how ColdFusion handles end tags, and how to write your custom tags to handle them, see *Handling end tags* in Executing custom tags.

One of the advantages to using the `cfimport` tag is that you can define a directory structure for your custom tags to organize them by category. For example, you can place all security tags in one directory, and all interface tags in another. You then import the tags from each directory and give them a different prefix:

```
<cfimport prefix="security" taglib="securityTags">
<cfimport prefix="ui" taglib="uiTags">
...
<security:validateUser name="Bob">
...
<ui:greeting name="Bob">
...
```

Reading your code becomes easier because you can identify the location of your custom tags from the prefix.

Securing custom tags

The ColdFusion security framework enables you to selectively restrict access to individual tag files and tag directories. This feature can be an important safeguard in team development. For details, see *Configuring and Administering ColdFusion*.

Accessing existing custom tags

Before creating a custom tag in CFML, review the free and commercial custom tags available on the Adobe developer's exchange (www.adobe.com/go/learn_cfu_cfdevcenter_en).You might find a tag that does what you want.

Tags are grouped in several broad categories and are downloadable as freeware, shareware, or commercial software. You can view each tag's syntax and usage information. The gallery contains a wealth of background information on custom tags and an online discussion forum for tag topics.

Tag names with the `cf_` preface are CFML custom tags; those tags with the `cfx_` preface are ColdFusion extensions written in Java or C++. For more information about the CFX tags, see Building Custom CFXAPI Tags.

If you do not find a tag that meets your specific needs, you can create your own custom tags in CFML.

© 2014 Adobe Systems Incorporated. All rights reserved.

Passing data to custom tags

To make your custom tags flexible, you often pass data to them for processing. To do so, you write custom tags that take tag attributes and other data as input from a calling page.

Passing values to and from custom tags

Because custom tags are individual ColdFusion pages, variables and other data are not automatically shared between a custom tag and the calling page. To pass data from the calling page to the custom tag, you can specify attribute name-value pairs in the custom tag, just as you do for normal HTML and CFML tags.
For example, to pass the value of the NameYouEntered variable to the cf_getmd tag, you can call the custom tag as follows:

```
<cf_getmd Name=#NameYouEntered#>
```

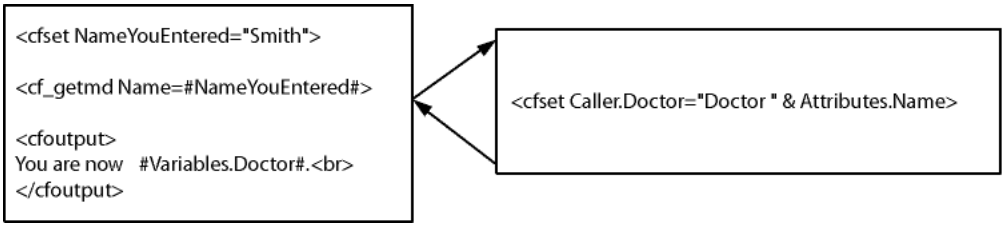
To pass multiple attributes to a custom tag, separate them with a space in the tag as follows:

```
<cf_mytag Firstname="Thadeus" Lastname="Jones">
```

In the custom tag, you use the Attributes scope to access attributes passed to the tag. Therefore, in the getmd.cfm page, you access the passed attribute as `Attributes.Name`. The mytag.cfm custom tag page refers to the passed attributes as `Attributes.Firstname` and `Attributes.Lastname`.
The custom tag page can also access variables set in the calling page by prefixing the calling page's local variable with Caller. However, this technique is not the best way to pass information to a custom tag, because each calling page would be required to create variables with the names required by the custom tag. You can create more flexible custom tags by passing parameters using attributes.
Variables created within a custom tag are deleted when the processing of the tag terminates. Therefore, if you want to pass information back to the calling page, write that information back to the Caller scope of the calling page. You cannot access the custom tag's variables outside the custom tag itself.
For example, use the following code in the getmd.cfm page to set the variable Doctor on the calling page:

```
<cfset Caller.Doctor="Doctor " & Attributes.Name>
```

If the variable Doctor does not exist in the calling page, this statement creates it. If the variable exists, the custom tag overwrites it.
The following image shows the relationship between the variables on the calling page and the custom tag:



One common technique used by custom tags is for the custom tag to take as input an attribute that contains the name of the variable to use to pass back results. For example, the calling page passes returnHere as the name of the variable to use to pass back results:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cf_mytag resultName="returnHere">
```

In mytag.cfm, the custom tag passes back its results using the following code:

```
<cfset "Caller.#Attributes.resultName#" = result>
```

Be careful not to overwrite variables in the calling page from the custom tag. Adopt a naming convention to minimize the chance of overwriting variables. For example, prefix the returned variable with customtagname_, where customtagname is the name of the custom tag.

Note

Data that pertains to the HTTP request or to the current application is visible in the custom tag page. This data includes the variables in the Form, URL, Cgi, Request, Cookies, Server, Application, Session, and Client scopes.

Using tag attributes summary

Custom tag attribute values are passed from the calling page to the custom tag page as name-value pairs. CFML custom tags support required and optional attributes. Custom tag attributes conform to the following CFML coding standards:

- ColdFusion passes any attributes in the Attributes scope.
- Use the `Attributes.`*`attribute_name`*__syntax when referring to passed attributes to distinguish them from custom tag page local variables.
- Attributes are not case sensitive.
- Attributes can be listed in any order within a tag.
- in the tag invocation, a space must separate attribute name-value pairs.
- Passed values that contain spaces must be enclosed in double-quotation marks.
- Use the `cfparam` tag with a `default`attribute at the top of a custom tag to test for and assign defaults for optional attributes that are passed from a calling page. For example:

```
<!--- The value of the variable Attributes.Name comes from the calling page.
If
 the calling page does not set it, make it "Who". --->
<cfparam name="Attributes.Name" default="Who">
```

- Use the `cfparam` tag or a `cfif` tag with an `IsDefined`function at the top of a custom tag to test for required attributes that must be passed from a calling page; for example, the following code issues an abort if the user does not specify the Name attribute to the custom tag:

```
<cfif not IsDefined("Attributes.Name")>
 <cfabort showError="The Name attribute is required.">
</cfif>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

Custom tag example with attributes

The following example creates a custom tag that uses an attribute that is passed to it to set the value of a variable called Doctor on the calling page.

1. Create a ColdFusion page (the calling page) with the following content:

```
<html>
<head>
 <title>Enter Name</title>
</head>
<body>
<!--- Enter a name, which could also be done in a form. --->
<!--- This example simply uses a cfset. --->
<cfset NameYouEntered="Smith">

<!--- Display the current name. --->
<cfoutput>
Before you leave this page, you're #Variables.NameYouEntered#.<br>
</cfoutput>

<!--- Go to the custom tag. --->
<cf_getmd Name="#NameYouEntered#">
<!--- Come back from the Custom tag --->

<!--- Display the results of the custom tag. --->
<cfoutput>
You are now #Variables.Doctor#.<br>
</cfoutput>
</body>
</html>
```

2. Save the page as `callingpage.cfm`.
3. Create another page (the custom tag) with the following content:

```
<!--- The value of the variable Attributes.Name comes from the calling page.
 If the calling page does not set it, make it "Who". --->
<cfparam name="Attributes.Name" default="Who">

<!--- Create a variable called Doctor, make its value "Doctor "
 followed by the value of the variable Attributes.Name.
 Make its scope Caller so it is passed back to the calling page.
--->
<cfset Caller.Doctor="Doctor " & Attributes.Name>
```

4. Save the page as `getmd.cfm`.
5. Open the file `callingpage.cfm` in your browser.
   The calling page uses the `getmd` custom tag and displays the results.

Reviewing the code

The following table describes the code and its function:

© 2014 Adobe Systems Incorporated. All rights reserved.

Code	Description
`<cfset NameYouEntered="Smith">`	In the calling page, create a variable `NameYouEntered` and assign it the value `Smith`.
`<cfoutput>` `Before you leave this page, you're` `#Variables.NameYouEntered#. ` `</cfoutput>`	In the calling page, display the value of the `NameYouEntered` variable before calling the custom tag.
`<cf_getmd Name="#NameYouEntered#">`	In the calling page, call the getmd custom tag and pass it the `Name` attribute whose value is the value of the local variable `NameYouEntered`.
`<cfparam name="Attributes.Name"` `default="Who">`	The custom tag page normally gets the Name variable in the Attributes scope from the calling page. Assign it the value `Who` if the calling page did not pass an attribute.
`<cfset Caller.Doctor="Doctor " &` `Attributes.Name>`	In the custom tag page, create a variable called `Doctor` in the Caller scope so it exists in the calling page as a local variable.Set its value to the concatenation of the string `"Doctor"` and the value of the {{Attributes.Name }}variable.
`<cfoutput>` `You are now` `#Variables.Doctor#. ` `</cfoutput>`	In the calling page, display the value of the `Doctor` variable returned by the custom tag page. (This example uses the Variables scope prefix to emphasize the fact that the variable is returned as a local variable.)

Passing custom tag attributes by using CFML structures

You can use the reserved attribute `attributeCollection` to pass attributes to custom tags using a structure. The `attributeCollection` attribute must reference a structure containing the attribute names as the keys and the attribute values as the values. You can freely mix `attributeCollection` with other attributes when you call a custom tag.

The key-value pairs in the structure specified by the `attributeCollection` attribute get copied into the Attributes scope of the custom tag pages. This action has the same effect as specifying the `attributeCollection` entries as individual attributes when you call the custom tag. The custom tag page refers to the attributes passed using `attributeCollection` the same way as it does other attributes; for example, as Attributes.CustomerName or Attributes.Department_number.

© 2014 Adobe Systems Incorporated. All rights reserved.

Note

You can combine tag attributes and the `attributeCollection` attribute when you use a custom tag directly or when you use the `cfmodule` tag to invoke a custom tag. If you pass an attribute with the same name both explicitly and in the `attributeCollection` structure, ColdFusion passes only the tag attribute to the custom tag and ignores the corresponding attribute from the attribute collection. You cannot combine tag attributes and the `attributeCollection` attribute when you use standard (built in) ColdFusion tags.

Custom tag processing reserves the `attributeCollection` attribute for the structure holding a collection of custom tag attributes. If `attributeCollection` does not reference such a collection, ColdFusion generates a template exception.
The following example uses an `attributeCollection` attribute to pass two of four attributes:

```
<cfset zort=StructNew()>
<cfset zort.x = "-X-">
<cfset zort.y = "-Y-">
<cf_testtwo a="blab" attributeCollection=#zort# foo="16">
```

If testtwo.cfm contains the following code:

```
---custom tag ---<br>
<cfoutput>#attributes.a# #attributes.x# #attributes.y#
 #attributes.foo#</cfoutput><br>
--- end custom tag ---
```

its output is the following statement:

```
---custom tag ---
blab -X- -Y- 16
--- end custom tag ---
```

One use for `attributeCollection` is to pass the entire Attributes scope of one custom tag to another. This technique is useful when you have one custom tag that calls a second custom tag and you want to pass all attributes from the first tag to the second.
For example, you call a custom tag with the following code:

```
<cf_first attr1="foo" attr2="bar">
```

To pass all the attributes of the first custom tag to the second, you include the following statement in first.cfm:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cf_second attributeCollection="#attributes#">
```

Within the body of second.cfm, you reference the parameters passed to it as follows:

```
<cfoutput>#attributes.attr1#</cfoutput>
<cfoutput>#attributes.attr2#</cfoutput>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

Managing custom tags

If you deploy custom tags in a multideveloper environment or distribute your tags publicly, you can use the advanced security and template encoding capabilities of ColdFusion.

The ColdFusion security framework enables you to selectively restrict access to individual tags or to tag directories. This restriction can be an important safeguard in team development. For more information, see Securing Applications.

You can use the command line utility `cfcompile` to precompile your custom tag files into Java class files or byte code. For more information, see *Using the cfcompile utility* in Deploying ColdFusion Applications in the *Configuring and Administering ColdFusion*.

© 2014 Adobe Systems Incorporated. All rights reserved.

Executing custom tags

ColdFusion provides techniques for executing custom tags, including handling end tags and processing body text.

Accessing tag instance data

When a custom tag page executes, ColdFusion keeps data related to the tag instance in the `thisTag` structure. You can access the `thisTag` structure from within your custom tag to control processing of the tag. The behavior is like that of the `File` tag-specific variable (sometimes called the File scope).
ColdFusion generates the variables in the following table and writes them to the `thisTag` structure:

Variable	Description
`ExecutionMode`	Contains the execution mode of the custom tag. Valid values are "start", "end", and "inactive".
{{HasEndTag }}	Distinguishes between custom tags that are called with and without end tags. Used for code validation. If the user specifies an end tag, `HasEndTag` is set to `true`; otherwise, it is set to `false`.
`GeneratedContent`	Specifies the content that the tag generates. This content includes anything in the body of the tag, including the results of any active content, such as ColdFusion variables and functions. You can process this content as a variable.
{{AssocAttribs }}	Contains the attributes of all nested tags if you use `cfassociate` to make them available to the parent tags. For more information, see *High-level data exchange* in Nesting custom tags.

The following example accesses the `ExecutionMode` variable of the `thisTag` structure from within a custom tag:

```
<cfif thisTag.ExecutionMode is 'start'>
```

Handling end tags

The preceding examples of custom tags reference a custom tag by using just a start tag:

```
<cf_date>
```

In this case, ColdFusion calls the custom tag page date.cfm to process the tag.
However, you can create custom tags that have both a start and an end tag. For example, the following tag has both a start and an end tag:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cf_date>
 ...
</cf_date>
```

ColdFusion calls the custom tag page date.cfm twice for a tag that includes an end tag: once for the start tag and once for the end tag. As part of the date.cfm page, you can determine if the call is for the start or end tag, and perform the appropriate processing.
ColdFusion also calls the custom tag page twice if you use the shorthand form of an end tag:

```
<cf_date/>
```

You can also call a custom tag using the `cfmodule` tag, as shown in the following example:

```
<cfmodule ...>
 ...
</cfmodule>
```

If you specify an end tag to `cfmessagebox`, then ColdFusion calls your custom tag as if it had both a start and an end tag.

Determining if an end tag is specified

You can write a custom tag that requires users to include an end tag. If a tag must have an end tag provided, you can use `thisTag.HasEndTag` in the custom tag page to verify that the user included the end tag.
For example, in date.cfm, you could include the following code to determine whether the end tag is specified:

```
<cfif thisTag.HasEndTag is 'False'>
 <!--- Abort the tag--->
 <cfabort showError="An end tag is required.">
</cfif>
```

Determining the tag execution mode

The variable `thisTag.ExecutionMode` contains the mode of invocation of a custom tag page. The variable has one of the following values:

- **Start** Mode for processing the end tag.
- **End** Mode for processing the end tag.
- **Inactive** Mode when the custom tag uses nested tags. For more information, see Nesting custom tags. If an end tag is not explicitly provided, ColdFusion invokes the custom tag page only once, in Start mode.
  A custom tag page named bold.cfm that makes text bold could be written as follows:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cfif thisTag.ExecutionMode is 'start'>
 <!--- Start tag processing --->
 <B>
<cfelse>
 <!--- End tag processing --->
 </B>
</cfif>
```

You then use this tag to convert the text to bold:

```
<cf_bold>This is bold text</cf_bold>
```

You can also use `cfswitch` to determine the execution mode of a custom tag:

```
<cfswitch expression=#thisTag.ExecutionMode#>
 <cfcase value= 'start'>
  <!--- Start tag processing --->
 </cfcase>
 <cfcase value='end'>
  <!--- End tag processing --->
 </cfcase>
</cfswitch>
```

Considerations when using end tags

How you code your custom tag to divide processing between the start tag and end tag depends greatly on the function of the tag. However, use the following rules to help you make your decisions:

- Use the start tag to validate input attributes, set default values, and validate the presence of the end tag if the custom tag requires it.
- Use the end tag to perform the actual processing of the tag, including any body text passed to the tag between the start and end tags. For more information on body text, see *Processing body text* in Executing custom tags.
- Perform output in either the start or end tag; do not divide it between the two tags.

Processing body text

Body text is any text that you include between the start and end tags when you call a custom tag, for example:

```
<cf_happybirthdayMessge name="Ellen Smith" birthDate="June, 8, 1993">
 <p> Happy Birthday Ellen!</p>
 <p> May you have many more!</p>
</cf_happybirthdayMessge>
```

In this example, the two lines of code after the start tag are the body text.
You can access the body text within the custom tag using the `thisTag.GeneratedContent` variable. The

© 2014 Adobe Systems Incorporated. All rights reserved.

variable contains all body text passed to the tag. You can modify this text during processing of the tag. The contents of the `thisTag.GeneratedContent` variables are returned to the browser as part of the tag's output.
The `thisTag.GeneratedContent` variable is always empty during the processing of a start tag. Any output generated during start tag processing is not considered part of the tag's generated content.
A custom tag can access and modify the generated content of any of its instances using the `thisTag.GeneratedContent` variable. In this context, the term *generated content* means the results of processing the body of a custom tag. The content includes all text and HTML code in the body, the results of evaluating ColdFusion variables, expressions, and functions, and the results generated by descendant tags. Any changes to the value of this variable result in changes to the generated content.
As an example, consider a tag that comments out the HTML generated by its descendants. Its implementation could look as follows:

```
<cfif thisTag.ExecutionMode is 'end'>
 <cfset thisTag.GeneratedContent ='<!--#thisTag.GeneratedContent#-->'>
</cfif>
```

Terminating tag execution

Within a custom tag, you typically perform error checking and parameter validation. As part of those checks, you can choose to abort the tag, using `cfabort`, if a required attribute is not specified or other severe error is detected.
The `cfexit` tag also terminates execution of a custom tag. However, the `cfexit` tag is designed to give you more flexibility when coding custom tags than `cfabort`. The `cfexit` tag's `method` attribute specifies where execution continues. The `cfexit` tag can specify that processing continues from the first child of the tag or continues immediately after the end tag marker.
You can also use the `method` attribute to specify that the tag body executes again. This capability enables custom tags to act as high-level iterators, emulating `cfloop` behavior.
The following table summarizes `cfexit` behavior:

Method attribute value	Location of cfexit call	Behavior
`ExitTag` (default)	Base page	Acts like `cfabort`
	`ExecutionMode=start`	Continue after end tag
	`ExecutionMode=end`	Continue after end tag
`ExitTemplate`	Base page	Acts like `cfabort`
	`ExecutionMode=start`	Continue from first child in body
	`ExecutionMode=end`	Continue after end tag
`Loop`	Base page	Error
	`ExecutionMode=start`	Error
	`ExecutionMode=end`	Continue from first child in body

© 2014 Adobe Systems Incorporated. All rights reserved.

Nesting custom tags

A custom tag can call other custom tags from within its body text, thereby *nesting* tags. ColdFusion uses nested tags such as `cfgraph` and `cfgraphdata`, `cfhttp` and `cfhttpparam`, and `cftree` and `cftreeitem`. The ability to nest tags allows you to provide similar functionality.
The following example shows a `cftreeitem` tag nested within a `cftree` tag:

```
<cftree name="tree1"
 required="Yes"
 hscroll="No">
 <cftreeitem value=fullname
  query="engquery"
  queryasroot="Yes"
  img="folder,document">
</cftree>
```

The calling tag is known as an *ancestor*, *parent*, or *base* tag; the tags that ancestor tags call are known as *descendant*, *child*, or *sub* tags. Together, the ancestor and all descendant tags are called *collaborating* tags.
In order to nest tags, the parent tag must have a closing tag.
The following table lists the terms that describe the relationships between nested tags:

Calling tag	Tag nested within the calling tag	Description
Ancestor	Descendant	An ancestor is any tag that contains other tags between its start and end tags. A descendant is any tag called by a tag.
Parent	Child	Parent and child are synonyms for ancestor and descendant.
Base tag	Sub tag	A base tag is an ancestor that you explicitly associate with a descendant, called a sub tag, with c `fassociate`.

You can create multiple levels of nested tags. In this case, the sub tag becomes the base tag for its own sub tags. Any tag with an end tag present can be an ancestor to another tag.
Nested custom tags operate through three modes of processing, which are exposed to the base tags through the variable `thisTag.ExecutionMode`.

Passing data between nested custom tags

A key custom tag feature is for collaborating custom tags to exchange complex data without user intervention, while encapsulating each tag's implementation so that others cannot see it.
When you use nested tags, address the following issues:

- What data must be accessible?
- Which tags can communicate to which tags?
- How are the source and targets of the data exchange identified?
- What CFML mechanism is used for the data exchange?

© 2014 Adobe Systems Incorporated. All rights reserved.

What data is accessible?

To enable developers to obtain maximum productivity in an environment with few restrictions, CFML custom tags can expose all their data to collaborating tags.
When you develop custom tags, document all variables that collaborating tags can access and/or modify. When your custom tags collaborate with other custom tags,
make sure that they do not modify any undocumented data.
To preserve encapsulation, place all tag data access and modification operations in custom tags. For example, rather than documenting that the variable MyQueryResults in a tag's implementation holds a query result and expecting users to manipulate MyQueryResults directly, create a nested custom tag that manipulates MyQueryResult. This technique protects the users of the custom tag from changes in the tag's implementation.

Variable scopes and special variables

Use the Request scope for variables in nested tags. The Request scope **is available to the base page, all pages it includes, all custom tag pages it calls, and all custom tag pages called by the included pages and custom tag pages. Collaborating custom tags that are not nested in a single tag can exchange data using the request structure. The Request scope is represented as a structure named `Request`.

Where is data accessible?

Two custom tags can be related in a variety of ways in a page. Ancestor and descendant relationships are important because they relate to the order of tag nesting.
A tag's descendants are inactive while the page is executed; that is, the descendent tags have no instance data. A tag, therefore, can only access data from its ancestors, not its descendants. Ancestor data is available from the current page and from the whole runtime tag context stack. The tag context stack is the path from the current tag element up the hierarchy of nested tags, including those tags in included pages and custom tag references, to the start of the base page for the request. Both `cfinclude` tags and custom tags appear on the tag context stack.

High-level data exchange

Although the ability to create nested custom tags is a tremendous productivity gain, keeping track of complex nested tag hierarchies can become a chore. The `cfassociate` tag lets the parent know what the children are up to. By adding this tag to a sub tag, you enable communication of its attributes to the base tag.

In addition, there are many cases in which descendant tags are used only as a means for data validation and exchange with an ancestor tag, such as `cfhttp`/`cfhttpparam` and `cftreecftreeitem`. You can use the `cfassociate` tag to encapsulate this processing.
The `cfassociate` tag has the following format:

```
<cfassociate baseTag="tagName" dataCollection="collectionName">
```

The `baseTag` attribute specifies the name of the base tag that gets access to this tag's attributes. The `dataCollection` attribute specifies the name of the structure in which the base tag stores the subtag data. Its default value is AssocAttribs. ColdFusion requires a `dataCollection` attribute only if the base tag can have more than one type of subtag. It is convenient for keeping separate collections of attributes, one per tag type.

> **Note**
>
> If the custom tag requires an end tag, the code processing the structure referenced by the `dataCollection` attribute must be part of end-tag code.

© 2014 Adobe Systems Incorporated. All rights reserved.

When `cfassociate` is encountered in a sub tag, the sub tag's attributes are automatically saved in the base tag. The attributes are in a structure appended to the end of an array whose name is thisTag.*collectionName*.
The `cfassociate` tag performs the following operations:

```
<!--- Get base tag instance data --->
 <cfset data = getBaseTagData(baseTag)>
 <!--- Create a string with the attribute collection name --->
 <cfset collection_Name = "data.#dataCollection#">
 <!--- Create the attribute collection, if necessary --->
 <cfif not isDefined(collectionName)>
  <cfset #collection_Name# = arrayNew(1)>
 </cfif>
 <!--- Append the current attributes to the array --->
 <cfset temp=arrayAppend(evaluate(collectionName), attributes)>
```

The code accessing subtag attributes in the base tag could look like the following:

```
<!--- Protect against no subtags --->
 <cfparam Name='thisTag.assocAttribs' default=#arrayNew(1)#>

 <!--- Loop over the attribute sets of all sub tags --->
 <cfloop index=i from=1 to=#arrayLen(thisTag.assocAttribs)#>

  <!--- Get the attributes structure --->
  <cfset subAttribs = thisTag.assocAttribs[i]>
  <!--- Perform other operations --->

 </cfloop>
```

Ancestor data access

A structure object contains all the ancestor's data.
The following functions provide access to ancestral data:

- `GetBaseTagList`: Returns a comma-delimited list of uppercase ancestor tag names, as a string. The first list element is the current tag, the next element is the parent tag name if the current tag is a nested tag. If the function is called for a top-level tag, it returns an empty string.
- `GetBaseTagData`, `InstanceNumber=1)`: Returns an object that contains all the variables (not just the local variables) of the nth ancestor with a given name. By default, the closest ancestor is returned. If there is no ancestor by the given name, or if the ancestor does not expose any data (such as `cfif`), an exception is thrown.

Example: ancestor data access

This example creates two custom tags and a simple page that calls each of the custom tags. The first custom tag calls the second. The second tag reports on its status and provides information about its ancestors.

Create the calling page

1. Create a ColdFusion page (the calling page) with the following content:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
Call cf_nesttag1 which calls cf_nesttag2<br>
<cf_nesttag1>
<hr>

Call cf_nesttag2 directly<br>
<cf_nesttag2>
<hr>
```

2. Save the page as nesttest.cfm.

Create the first custom tag page

1. Create a ColdFusion page with the following content:

```
<cf_nesttag2>
```

2. Save the page as nesttag1.cfm.

Create the second custom tag page

1. Create a ColdFusion page with the following content:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cfif thisTag.executionmode is 'start'>
 <!--- Get the tag context stack. The list looks something like
 "MYTAGNAME, CALLINGTAGNAME, ..." --->
 <cfset ancestorlist = getbasetaglist()>

 <!--- Output your own name. You are the first entry in the context stack.
--->
 <cfoutput>
 <p>I'm custom tag #ListGetAt(ancestorlist,1)#</p>

 <!--- Output all the contents of the stack a line at a time. --->
 <cfloop index="loopcount" from="1" to="#listlen(ancestorlist)#">
  Ancestorlist entry #loopcount# n is #ListGetAt(ancestorlist,loopcount)#<br>
 </cfloop><br>
 </cfoutput>

 <!--- Determine whether you are nested inside a custom tag. Skip the first
  element of the ancestor list, i.e., the name of the custom tag I'm in. --->
 <cfset incustomtag = ''>
 <cfloop index="elem"
  list="#listrest(ancestorlist)#">
  <cfif (left(elem, 3) eq 'cf_')>
   <cfset incustomtag = elem>
   <cfbreak>
  </cfif>
 </cfloop>

 <cfif incustomtag neq ''>
  <!--- Say that you are there. --->
  <cfoutput>
   I'm running in the context of a custom tag named #inCustomTag#.<p>
  </cfoutput>

  <!--- Get the tag instance data. --->
  <cfset tagdata = getbasetagdata(incustomtag)>

  <!--- Find out the tag's execution mode. --->
  I'm located inside the
  <cfif tagdata.thisTag.executionmode neq 'inactive'>
   custom tag code either because it is in its start or end execution mode.
  <cfelse>
   body of the tag
  </cfif>
  <p>
 <cfelse>
  <!--- Say that you are lonely. --->
  I'm not nested inside any custom tags. :^( <p>
 </cfif>
</cfif>
```

2. Save the page as nesttag2.cfm.
3. Open the file nesttest.cfm in your browser.

#back to top

© 2014 Adobe Systems Incorporated. All rights reserved.

Building Custom CFXAPI Tags

Sometimes, the best approach to application development is to develop elements of your application by building executable to run with ColdFusion. Perhaps the application requirements go beyond what is currently feasible in CFML. Perhaps you can improve application performance for certain types of processing. Or, you have existing code that already solves an application problem and you want to incorporate it into your ColdFusion application. To meet these types of requirements, you can use the ColdFusion Extension Application Programming Interface (CFX API) to develop custom ColdFusion tags based on Java or C++.

© 2014 Adobe Systems Incorporated. All rights reserved.

What are CFX tags?

ColdFusion Extension (CFX) tags are custom tags written against the ColdFusion Extension Application Programming Interface. Generally, you create a CFX tag if you want to do something that is not possible in CFML, or if you want to improve the performance of a repetitive task.
One common use of CFX tags is to incorporate existing application functionality into a ColdFusion application. That means if you already have the code available, CFX tags make it easy to use it in your application.
CFX tags can do the following:

- Handle any number of custom attributes.
- Use and manipulate ColdFusion queries for custom formatting.
- Generate ColdFusion queries for interfacing with non-ODBC based information sources.
- Dynamically generate HTML to return to the client.
- Set variables within the ColdFusion application page from which they are called.
- Throw exceptions that result in standard ColdFusion error messages.
  You can build CFX tags using C++ or Java.

Note

ColdFusion provides several different techniques to create reusable code, including custom tags. For information on all of these techniques, see Creating ColdFusion Elements.

© 2014 Adobe Systems Incorporated. All rights reserved.

Before you begin developing CFX tags in Java

Before you begin developing CFX tags in Java, configure your Java development environment. Also, it can be helpful to review the examples here before you create CFX tags.

Sample Java CFX tags

Before you begin developing a CFX tag in Java, it can be useful to study sample CFX tags. You can find the Java source files for the examples for Windows in the cfx\java\distrib\examples subdirectory of the main installation directory. In UNIX systems, the files are located in the cfx/java/examples directory. The following table describes the example tags:

Example	Action	Demonstrates
`HelloColdFusion`	Prints a personalized greeting.	The minimal implementation required to create a CFX tag.
`ZipBrowser`	Retrieves the contents of a ZIP archive.	How to generate a ColdFusion query and return it to the calling page.
`ServerDateTime`	Retrieves the date and time from a network server.	Attribute validation, using numeric attributes, and setting variables within the calling page.
`OutputQuery`	Returns a ColdFusion query in an HTML table.	How to handle a ColdFusion query as input, throw exceptions, and generate dynamic output.
`HelloWorldGraphic`	Generates a "Hello World!" graphic in JPEG format.	How to dynamically create and return graphics from a Java CFX tag.

Setting up your development environment to develop CFX tags in Java

You can use a wide range of Java development environments, including the Java Development Kit (JDK) from Sun, to build Java CFX tags. You can download the JDK from Sun http://java.sun.com/j2se.
Adobe recommends that you use one of the commercial Java IDEs, so you have an integrated environment for development, debugging, and project management.

Configuring the classpath

To configure your development environment to build Java CFX tags, ensure that the supporting classes are visible to your Java compiler. These classes are located in the cfx.jar archive, located in one of the following directories:

- **Server configuration** *cf_root*/wwwroot/WEB-INF/lib
- **J2EE configuration** *cf_webapp_root*/WEB-INF/libConsult your Java development tool documentation to determine how to configure the compiler classpath for your particular environment. The cfx.jar archive contains the classes in the `com.allaire.cfx` package, which are required for developing and deploying Java CFX tags.When you create new Java CFX tags, compile them into the WEB-INF/classes directory. Doing so simplifies your development, debugging, and testing processes.After you finish with development and testing, you can deploy your Java CFX tag anywhere on the classpath visible to ColdFusion.

© 2014 Adobe Systems Incorporated. All rights reserved.

Customizing and configuring Java

Use the ColdFusion Administrator > Server Settings > JVM and Java Settings page to customize your Java development environment by customizing the classpath and Java system properties, or by specifying an alternate JVM. For more information, see the ColdFusion Administrator online Help.

© 2014 Adobe Systems Incorporated. All rights reserved.

Writing a Java CFX tag

To create a Java CFX tag, create a class that implements the Custom tag interface. This interface contains one method, `processRequest`, which is passed `Request` and `Response` objects that are then used to do the work of the tag.
The example in the following procedure creates a simple Java CFX tag named `cfx_MyHelloColdFusion` that writes a text string back to the calling page.

1. Create a source file in your editor with the following code:

```
import com.allaire.cfx.* ;

public class MyHelloColdFusion implements CustomTag {
 public void processRequest( Request request, Response response )
 throws Exception {
 String strName = request.getAttribute( "NAME" ) ;
 response.write( "Hello, " + strName ) ;
 }
}
```

1. Save the file as MyHelloColdFusion.java in the WEB_INF/classes directory.
2. Compile the java source file into a class file using the Java compiler. If you are using the command line tools bundled with the JDK, use the following command line, which you execute from within the classes directory:

```
javac -classpath cf_root\WEB-INF\lib\cfx.jar MyHelloColdFusion.java
```

Note

The previous command works only if the Java compiler (`javac.exe`) is in your path. If it is not in your path, specify the fully qualified path; for example, c:\jdk1.3.1_01\bin\javac in Windows or /usr/java/bin/javac in UNIX.

If you receive errors during compilation, check the source code to make sure that you entered it correctly. If no errors occur, you successfully wrote your first Java CFX tag.

Calling the CFX tag from a ColdFusion page

You call Java CFX tags from within ColdFusion pages by using the name of the CFX tag that is registered on the ColdFusion Administrator CFX Tags page. This name should be the prefix `cfx_` followed by the class name (without the .class extension).

Register a Java CFX tag in the ColdFusion Administrator

1. In the ColdFusion Administrator, select Extensions > CFX Tags.
2. Click Register Java CFX.
3. Enter the tag name (for example, `cfx_MyHelloColdFusion`).
4. Enter the class name without the .class extension (for example, `MyHelloColdFusion`).
5. (Optional) Enter a description.
6. Click Submit.
   You can now call the tag from a ColdFusion page.

© 2014 Adobe Systems Incorporated. All rights reserved.

Call a CFX tag from a ColdFusion page

1. Create a ColdFusion page (.cfm) in your editor with the following content to call the HelloColdFusion custom tag:

```
<html>
<body>
 <cfx_MyHelloColdFusion NAME="Les">
</body>
</html>
```

2. Save the file in a directory configured to serve ColdFusion pages. For example, you can save the file as C:\inetpub\wwwroot\cfdocs\testjavacfx.cfm in Windows or /home/docroot/cfdocs/testjavacfx.cfm in UNIX.
3. If you have not already done so, register the CFX tag in the ColdFusion Administrator (see *Registering CFX tags* in Developing CFX tags in C++).
4. Request the page from your browser using the appropriate URL; for example:http://localhost/cfdocs/testjavacfx.cfm
   ColdFusion processes the page and returns a page that displays the text "Hello, Les." If an error is returned instead, check the source code to make sure that you entered it correctly.

Delete a CFX tag in the ColdFusion Administrator

1. In the ColdFusion Administrator, select Extensions > CFX Tags.
2. For the tag to delete, click the Delete icon in the Controls column of the Registered CFX Tags list.

Processing requests

Implementing a Java CFX tag requires interaction with the `Request` and `Response` objects passed to the `processRequest` method. In addition, CFX tags that must work with ColdFusion queries also interface with the `Query` object. The `com.allaire.cfx` package, located in the WEB-INF/lib/cfx.jar archive, contains the `Request`, `Response`, and `Query` objects.
For a complete description of these object types, see ColdFusion Java CFX Reference in the *CFML Reference*. For a complete example Java CFX tag that uses `Request`, `Response`, and `Query` objects, see ZipBrowser example.

Request object

The `Request` object is passed to the `processRequest` method of the `CustomTag` interface. The following table lists the methods of the `Request` object for retrieving attributes, including queries, passed to the tag and for reading global tag settings:

Method	Description
attributeExists	Checks whether the attribute was passed to this tag.
debug	Checks whether the tag contains the `debug` attribute.
getAttribute	Retrieves the value of the passed attribute.
getAttributeList	Retrieves a list of all attributes passed to the tag.
getIntAttribute	Retrieves the value of the passed attribute as an integer.

© 2014 Adobe Systems Incorporated. All rights reserved.

getQuery	Retrieves the query that was passed to this tag, if any.
getSetting	Retrieves the value of a global custom tag setting.

For detailed reference information on each of these interfaces, see the *CFML Reference*.

Response object

The Response object is passed to the processRequest method of the CustomTag interface. The following table lists the methods of the Response object for writing output, generating queries, and setting variables within the calling page:

Method	Description
write	Outputs text to the calling page.
SetVariable	Sets a variable in the calling page.
addQuery	Adds a query to the calling page.
writeDebug	Outputs text to the debug stream.

For detailed reference information on each of these interfaces, see the *CFML Reference*.

Query object

The Query object provides an interface for working with ColdFusion queries. The following table lists the methods of the Query object for retrieving name, row count, and column names and methods for getting and setting data elements:

Method	Description
getName	Retrieves the name of the query.
getRowCount	Retrieves the number of rows in the query.
getColumnIndex	Retrieves the index of a query column.
getColumns	Retrieves the names of the query columns.
getData	Retrieves a data element from the query.
addRow	Adds a new row to the query.
setData	Sets a data element within the query.

For detailed reference information on each of these interfaces, see *CFML Reference*.

Life cycle of Java CFX tags

© 2014 Adobe Systems Incorporated. All rights reserved.

A new instance of the Java CFX object is created for each invocation of the Java CFX tag. As a result, it is safe to store per-request instance data within the members of your `CustomTag` object. To store data and objects that are accessible to all instances of your `CustomTag`, use static data members. If you do so, ensure that all accesses to the data are thread-safe.

© 2014 Adobe Systems Incorporated. All rights reserved.

ZipBrowser example

The following example shows the use of the `Request`, `Response`, and `Query` objects. The example uses the java.util.zip package to implement a Java CFX tag called `cfx_ZipBrowser`, which is a ZIP file browsing tag.

> **Note**
>
> The Java source file that implements cfx_ZipBrowser, ZipBrowser.java, is included in the cf_root/cfx/java/distrib/examples (server configuration) or cf_webapp_root/WEB-INF/cfusion/cfx/java/distrib/examples (J2EE configuration) directory. Compile ZipBrowser.java to implement the tag.

The tag `archive` attribute specifies the fully qualified path of the ZIP archive to browse. The tag `name` attribute must specify the query to return to the calling page. The returned query contains three columns: Name, Size, and Compressed.
For example, to query an archive at the path C:\logfiles.zip for its contents and output the results, you use the following CFML code:

```
<cfx_ZipBrowser
 archive="C:\logfiles.zip"
 name="LogFiles">

<cfoutput query="LogFiles">
#Name#,#Size#, #Compressed# <BR>
</cfoutput>
```

The Java implementation of ZipBrowser is as follows:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
import com.allaire.cfx.* ;
import java.util.Hashtable ;
import java.io.FileInputStream ;
import java.util.zip.* ;

public class ZipBrowser implements CustomTag {
 public void processRequest( Request request, Response response )
  throws Exception {
  // Validate that required attributes were passed.
  if (!request.attributeExists( "ARCHIVE" ) || !request.attributeExists( "NAME" ) )
{
   throw new Exception(
  "Missing attribute (ARCHIVE and NAME are both " +
   "required attributes for this tag)" ) ;
  }
  // get attribute values
  String strArchive = request.getAttribute( "ARCHIVE" ) ;
  String strName = request.getAttribute( "NAME" ) ;

 // create a query to use for returning the list of files
  String[] columns = { "Name", "Size", "Compressed" } ;
  int iName = 1, iSize = 2, iCompressed = 3 ;
  Query files = response.addQuery( strName, columns ) ;

 // read the ZIP file and build a query from its contents
  ZipInputStream zin = new ZipInputStream( new FileInputStream(strArchive) ) ;
  ZipEntry entry ;
  while ( ( entry = zin.getNextEntry()) != null ) {
   // Add a row to the results.
    int iRow = files.addRow() ;

    // populate the row with data
    files.setData( iRow, iName, entry.getName() ) ;
    files.setData( iRow, iSize, String.valueOf(entry.getSize()) ) ;
    files.setData( iRow, iCompressed,
     String.valueOf(entry.getCompressedSize())) ;

    // Finish up with entry.
    zin.closeEntry() ;
   }

   // Close the archive.
   zin.close() ;
 }
}
```

© 2014 Adobe Systems Incorporated. All rights reserved.

Approaches to debugging Java CFX tags

Java CFX tags are not stand-alone applications that run in their own process, like typical Java applications. Rather, they are created and run from an existing process. As a result, debugging Java CFX tags is more difficult, because you cannot use an interactive debugger to debug Java classes that another process has loaded.
To overcome this limitation, you can use one of the following techniques:

- Debug the CFX tag while it is running within ColdFusion by outputting the debug information as needed.
- Debug the CFX tag using a Java IDE (Integrated Development Environment) that supports debugging features, such as setting breakpoints, stepping through your code, and displaying variable values.
- Debug the request in an interactive debugger offline from ColdFusion using the special com.allaire.cfx debugging classes.

Outputting debugging information

Before using interactive debuggers became the norm, programmers typically debugged their programs by inserting output statements in their programs to indicate information such as variable values and control paths taken. Often, when a new platform emerges, this technique comes back into vogue while programmers wait for more sophisticated debugging technology to develop for the platform.
If you must debug a Java CFX tag while running against a live production server, reset this technique. In addition to outputting debugging text using the `Response.write` method, you can also call your Java CFX tag with the `debug="On"` attribute. This attribute flags the CFX tag that the request is running in debug mode and therefore can generate additional extended debugging information. For example, to call the `HelloColdFusion` CFX tag in debugging mode, use the following CFML code:

```
<cfx_HelloColdFusion name="Robert" debug="On">
```

To determine whether a CFX tag is run with the `debug` attribute, use the `Request.debug` method. To write debugging output in a special debugging block after the tag finishes executing, use the `Response.writeDebug` method. For information on using these methods, see ColdFusion Java CFX Reference in *CFML Reference*.

Debugging in a Java IDE

You can use a Java IDE to debug your Java CFX tags. As a result, you can develop your Java CFX tag and debug it in a single environment.

1. Start your IDE.
2. In the project properties (or the project setting of your IDE), make sure your CFX class is in the *web_root*\WEB-INF\classes directory or in the system classpath.
3. Make sure the libraries *cf_root*/wwwroot/WEB-INF/lib/cfx.jar (*cf_webapp_root*/WEB-INF/lib/cfx.jar in the J2EE configuration) and *cf_root*/runtime/lib/jrun.jar (server configuration only) are included in your classpath.
4. In your project settings, set your main class to jrunx.kernel.JRun and application parameters to `-start default`.
5. Debug your application by setting breakpoints, single stepping, displaying variables, or by performing other debugging actions.

Using the debugging classes

To develop and debug Java CFX tags in isolation from the ColdFusion, you use three special debugging classes that are included in the `com.allaire.cfx` package. These classes lets you simulate a call to the `processRequest` method of your CFX tag within the context of the interactive debugger of a Java development environment. The three debugging classes are the following:

© 2014 Adobe Systems Incorporated. All rights reserved.

- `DebugRequest`: An implementation of the `Request` interface that lets you initialize the request with custom attributes, settings, and a query.
- `DebugResponse`: An implementation of the `Response` interface that lets you print the results of a request once it has completed.
- `DebugQuery`: An implementation of the `Query` interface that lets you initialize a query with a name, columns, and a data set.

Implement a main method

1. Create a `main` method for your Java CFX class.
2. Within the `main` method, initialize a `DebugRequest` and `DebugResponse`, and a `DebugQuery`. Use the appropriate attributes and data for your test.
3. Create an instance of your Java CFX tag and call its `processRequest` method, passing in the `DebugRequest` and `DebugResponse` objects.
4. Call the `DebugResponse.printResults` method to output the results of the request, including content generated, variables set, queries created, and so on.
   After you implement a `main` method as described previously, you can debug your Java CFX tag using an interactive, single-step debugger. Specify your Java CFX class as the `main` class, set breakpoints as appropriate, and begin debugging.

Example:debugging classes

The following example demonstrates how to use the debugging classes:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
import java.util.Hashtable ;
import com.allaire.cfx.* ;

public class OutputQuery implements CustomTag {
 // debugger testbed for OutputQuery
 public static void main(String[] argv) {
  try {
   // initialize attributes
   Hashtable attributes = new Hashtable() ;
   attributes.put( "HEADER", "Yes" ) ;
   attributes.put( "BORDER", "3" ) ;

   // initialize query

   String[] columns = { "FIRSTNAME", "LASTNAME", "TITLE" } ;

   String[][] data = {
    { "Stephen", "Cheng", "Vice President" },
    { "Joe", "Berrey", "Intern" },
    { "Adam", "Lipinski", "Director" },
    { "Lynne", "Teague", "Developer" } };

   DebugQuery query = new DebugQuery( "Employees", columns, data ) ;

   // create tag, process debugging request, and print results
   OutputQuery tag = new OutputQuery() ;
   DebugRequest request = new DebugRequest( attributes, query ) ;
   DebugResponse response = new DebugResponse() ;
   tag.processRequest( request, response ) ;
   response.printResults() ;
  }
  catch( Throwable e ) {
   e.printStackTrace() ;
  }
}

 public void processRequest(Request request, Response response) throws Exception {
  // ...code for processing the request...
 }
}
```

© 2014 Adobe Systems Incorporated. All rights reserved.

Developing CFX tags in C++

You can develop CFX tags in C++.

Sample C++ CFX tags

Before you begin development of a CFX tag in C++, you can study the two CFX tags included with ColdFusion. These examples can help you get started working with the CFXAPI. The two example tags are as follows:

- `CFX_DIRECTORYLIST`: Queries a directory for the list of files it contains.
- `CFX_NTUSERDB` (Windows only): Lets you add and delete Windows NT users.
  In Windows, these tags are located in the *cf_root*\cfx\examples directory. In UNIX, these tags are in the *cf_root*/coldfusion/cfx/examples directory.

Setting up your C++ development environment

The following compliers generate valid CFX code for UNIX platforms:

Platform	Compiler
Solaris	Sun Workshop C++ compiler, version 5.0 or higher (gcc cannot be used to compile CFX code on Solaris)
Linux	Gnu C++ compiler - gcc

Before you can use your C++ compiler to build custom tags, enable the compiler to locate the CFX API header file, cfx.h. In Windows, add the CFX API include directory to your list of global include paths. In Windows, this directory is *cf_root*\cfx\include. In UNIX, this directory is *cf_root*/cfx/include. in UNIX, you need `-I` <*includepath*> on your compile line (see the Makefile for the directory list example in the cfx/examples directory).

Compiling C++ CFX tags

CFX tags built in Windows and in UNIX must be thread-safe. Compile CFX tags for Solaris with the `-mt` switch on the Sun compiler.

Locating your C++ library files in UNIX

In UNIX systems, your C++ library files can be in any directory as long as the directory is included in LD_LIBRARY_PATH or SHLIB_PATH (HP-UX only).

Implementing C++ CFX tags

CFX tags built in C++ use the tag request object, represented by the C++ CCFXRequest class. This object represents a request made from an application page to a custom tag. A pointer to an instance of a request object is passed to the main procedure of a custom tag. The methods available from the request object let the custom tag accomplish its work. For information about the CFX API classes and members, see ColdFusion C++ CFX Reference in the *CFML Reference*.

> **Note**
>
> Calling a nonexistent C++ CFX procedure or entry point causes a JVM crash in UNIX.

Debugging C++ CFX tags

© 2014 Adobe Systems Incorporated. All rights reserved.

After you configure a debugging session, you run your custom tag from within the debugger, set breakpoints, single-step, and so on.

Debugging in Windows

You can debug custom tags within the Visual C++ environment.

1. Build your C++ CFX tag using the debug option.
2. Restart ColdFusion.
3. Start Visual C++.
4. Select Build > Start Debug > AttachProcess.
5. Select jrunsvc.exe.Adobe recommends that you shut down all other Java programs.
6. Execute any ColdFusion page that calls the CFX tag.
7. Select File > Open to open a file in VisualDev in which to set a breakpoint.
8. Set a breakpoint in the CFX project. The best place is to place it in `ProcessRequest()`. Next time you execute the page you will reach the breakpoint.

Registering CFX tags

To use a CFX tag in your ColdFusion applications, first register it in the Extensions, CFX Tags page in the ColdFusion Administrator.

1. In the ColdFusion Administrator, select Extensions > CFX Tags.
2. Click Register C++ CFX.
3. Enter the Tag name (for example, `cfx_MyNewTag`).
4. If the Server Library .dll field is empty, enter the filepath.
5. Accept the default Procedure entry.
6. Clear the Keep library loaded box while developing the tag. For improved performance, when the tag is ready for production use, you can select this option to keep the DLL in memory.
7. (Optional) Enter a description.
8. Click Submit.
   You can now call the tag from a ColdFusion page.

Delete a CFX tag

1. In the ColdFusion Administrator, select Extensions > CFX Tags.
2. For the tag to delete, click the Delete icon in the Controls column of the Registered CFX Tags list.

- Sample C++ CFX tags
- Setting up your C++ development environment
- Compiling C++ CFX tags
- Locating your C++ library files in UNIX
- Implementing C++ CFX tags
- Debugging C++ CFX tags
  - Debugging in Windows
- Registering CFX tags
  - Delete a CFX tag

© 2014 Adobe Systems Incorporated. All rights reserved.

Using the member functions

A lot of enhancements have been made to the core CFML syntax that will aid in developing ColdFusion-based applications elegantly. One of the primary enhancements is the introduction of member functions for data structure and data objects. You can now start coding in a true object oriented style.

For instance, consider the following headless function:

```
ArrayAppend (empArr, emp)
```

It can now be written as:

```
empArr.append(emp)
```

where 'arrayObj' is a reference to the CFArray class.

The following example depicts the new usage of member functions:

```
<cfscript>
       //The old way
       var myArray = ArrayNew(1);
       ArrayAppend(myArray, "objec_new");
       ArraySort(myArray, "ASC");

       // The new way
       myArray.append("objec_new");
       myArray.sort("ASC");

       // The new way
       var myProductObject = createObject("java", "myJavaclass");
       myjavaList = myProductObject.getProductList();
       myjavaList.add("newProduct"); // Java API

       myjavaList.append("newProduct"); // CF API
       myjavaList.sort("ASC");
</cfscript>
```

Adobe blog post
Member functions for ColdFusion data types

Member function for the following data types are supported:

© 2014 Adobe Systems Incorporated. All rights reserved.

- Array
- String
- List
- Struct
- Date
- Spreadsheet
- XML
- Query
- Image

Supported Array member functions

The following Array member functions are supported:

ArrayAppend	someVar.append()
ArrayAvg	someVar.avg()
ArrayClear	someVar.clear()
ArrayContains	someVar.contains()
ArrayDelete	someVar.delete()
ArrayDeleteAt	someVar.deleteAt()
ArrayEach	someVar.each()
ArrayFilter	someVar.filter()
ArrayFind	someVar.find()
ArrayFindAll	someVar.findAll()
ArrayFindAllNoCase	someVar.findAllNoCase()
ArrayFindNoCase	someVar.findNoCase()
ArrayInsertAt	someVar.insertAt()
ArrayIsDefined	someVar.isDefined()
ArrayIsEmpty	someVar.isEmpty()
ArrayLen	someVar.len()
ArrayMax	someVar.max()
ArrayMin	someVar.min()
ArrayPrepend	someVar.prepend()
ArrayResize	someVar.resize()
ArraySet	someVar.set()
ArraySlice	someVar.slice()
ArraySort	someVar.sort()
ArraySum	someVar.sum()

© 2014 Adobe Systems Incorporated. All rights reserved.

ArraySwap	someVar.swap()

Supported String member functions

The following String member functions are supported:

CJustify	someVar.cJustify()
Compare	someVar.compare()
CompareNoCase	someVar.compareNocase()
Find	someVar.find()
FindNoCase	someVar.findNoCase()
FindOneOf	someVar.findOneOf()
GetToken	someVar.getToken()
Insert	someVar.insert()
LCase	someVar.lCase()
LJustify	someVar.lJustify()
Left	someVar.left()
Len	someVar.len()
Mid	someVar.mid()
RJustify	someVar.rJustify()
RTrim	someVar.trim()
RemoveChars	someVar.removeChars()
RepeatString	someVar.repeatString()
Replace	someVar.replace()
ReplaceList	someVar.replaceList()
ReplaceNoCase	someVar.replaceNocase()
Reverse	someVar.reverse()
Right	someVar.right()
SpanExcluding	someVar.spanExcluding()
SpanIncluding	someVar.spanIncluding()
StripCR	someVar.stripCR()
Trim	someVar.trim()
UCase	someVar.uCase()

© 2014 Adobe Systems Incorporated. All rights reserved.

Wrap	someVar.wrap()

Similarly, the following String member functions are also supported:

Decrypt, Encrypt, BinaryDecode, BinaryEncode, CharsetDecode, CharsetEncode, URLDecode, URLEncodedFormat, HTMLEditFormat, HTMLCodeFormat, ParagraphFormat, JSStringFormat, XmlFormat, FormatBaseN, HTMLEditFormat, HTMLCodeFormat, ParagraphFormat, ToBinary, ToString, ToBase64, Val, GenerateSecretKey, Hash, REFind, REFindNoCase, REMatch, REMatchNoCase, REReplace, REReplaceNoCase, ReplaceList, LSParseEuroCurrency, LSParseDateTime, LSIsCurrency, LSIsDate, LSIsNumeric, LSParseCurrency, LSParseNumber, and ParseDateTime

Supported List member functions

The following List member functions are supported:

ListAppend	someVar.listAppend()
ListChangeDelims	someVar.listChangeDelims()
ListContains	someVar.listContains()
ListContainsNoCase	someVar.listContainsNoCase()
ListDeleteAt	someVar.listDeleteAt()
ListEach	someVar.listEach()
ListFind	someVar.listFind()
ListFindNoCase	someVar.listFindNoCase()
ListFirst	someVar.listFirst()
ListGetAt	someVar.listGetAt()
ListInsertAt	someVar.listInsertAt()
ListLast	someVar.listLast()
ListLen	someVar.listLen()
ListMap	someVar.listMap()
ListPrepend	someVar.listPrepend()
ListQualify	someVar.listQualify()
ListReduce	someVar.listReduce()
ListRest	someVar.listRest()
ListSetAt	someVar.listSetAt()
ListSort	someVar.listSort()
ListToArray	someVar.listToArray()
ListValueCount	someVar.listValueCount()
ListValueCountNoCase	someVar.listValueCountNoCase()

© 2014 Adobe Systems Incorporated. All rights reserved.

Supported Struct member functions

The following Struct member functions are supported:

StructIsEmpty	someVar.isEmpty()
StructAppend	someVar.append()
StructClear	someVar.clear()
StructCopy	someVar.copy()
StructCount	someVar.count()
StructDelete	someVar.delete()
StructFind	someVar.find()
StructFindValue	someVar.findValue()
StructUpdate	someVar.update()
StructSort	someVar.sort()
StructInsert	someVar.insert()
StructEach	someVar.each()
StructKeyArray	someVar.keyArray()
StructKeyExists	someVar.keyExists()
StructKeyList	someVar.keyList()

Supported Date member functions

The following Date member functions are supported:

CreateODBCDate	someVar.createODBCDate()
CreateODBCDateTime	someVar.createODBCDateTime()
DateDiff	someVar.diff()
CreateODBCTime	someVar.createODBCTime()
LSDateFormat	someVar.lsDateFormat()
DatePart	someVar.datepart()
DaysInYear	someVar.daysIn
Second	someVar.second()
Minute	someVar.minute()
Hour	someVar.hour()

© 2014 Adobe Systems Incorporated. All rights reserved.

Day	someVar.day()
Week	someVar.week()
Month	someVar.month()
Quarter	someVar.quarter()
Year	someVar.year()
DaysInMonth	someVar.daysInMonth()
DayOfWeek	someVar.dayOfweek()
DayOfYear	someVar.dayOfYear()
FirstDayOfMonth	someVar.firstDayOfMonth()
DateTimeFormat	someVar.dateTimeFormat()
TimeFormat	someVar.timeFormat()
DateFormat	someVar.dateFormat()
DateAdd	someVar.add()
DateConvert	someVar.convert()

Supported Image member functions

The following Image member functions are supported:

ImageGetWidth	someVar.getWidth()
ImageSetDrawingColor	someVar.setDrawingColor()
ImageGetBufferedImage	someVar.getBufferedImage()
ImageTranslateDrawingAxis	someVar.translateDrawingAxis()
ImageSetDrawingStroke	someVar.setDrawingStroke()
ImageNegative	someVar.negative()
ImageCopy	someVar.copy()
ImageDrawRect	someVar.drawRect()
ImageCrop	someVar.crop()
ImageGetHeight	someVar.getHeight()
ImageGetIPTCTag	someVar.getIPTCTag()
ImageDrawOval	someVar.drawOval()
ImageSharpen	someVar.sharpen()
ImageOverlay	someVar.overlay()
ImageGetEXIFTag	someVar.getEXIFTag()

© 2014 Adobe Systems Incorporated. All rights reserved.

ImageDrawBeveledRect	someVar.drawBeveledRect()
ImageAddBorder	someVar.addBorder()
ImageShear	someVar.shear()
ImageInfo	someVar.info()
ImagePaste	someVar.paste()
ImageDrawArc	someVar.drawArc()
ImageShearDrawingAxis	someVar.shearDrawingAxis()
ImageDrawRoundRect	someVar.drawRoundRect()
ImageGrayscale	someVar.grayscale()
ImageSetDrawingTransparency	someVar.setDrawingTransparency()
ImageScaleToFit	someVar.scaleToFit()
ImageClearRect	someVar.clearRect()
ImageTranslate	someVar.translate()
ImageFlip	someVar.flip()
ImageWriteBase64	someVar.writeBase64()
ImageSetBackgroundColor	someVar.setBackgroundColor()
ImageDrawLine	someVar.drawLine()
ImageDrawQuadraticCurve	someVar.drawQuadraticCurve()
ImageRotate	someVar.rotate()
ImageGetBlob	someVar.getBlob()
ImageWrite	someVar.write()
ImageBlur	someVar.blur()
ImageRotateDrawingAxis	someVar.rotateDrawingAxis()
ImageSetAntialiasing	someVar.setAntialiasing()
ImageDrawPoint	someVar.drawPoint()
ImageDrawCubicCurve	someVar.drawCubicCurve()
ImageXORDrawingMode	someVar.xorDrawingMode()
ImageDrawText	someVar.drawText()
ImageDrawLines	someVar.drawLines()
ImageResize	someVar.resize()

Supported Spreadsheet member functions

The following Spreadsheet member functions are supported:

© 2014 Adobe Systems Incorporated. All rights reserved.

SpreadsheetDeleteRow	someVar.deleteRow()
SpreadsheetFormatColumn	someVar.formatColumn()
SpreadsheetShiftRows	someVar.shiftRows()
SpreadsheetCreateSheet	someVar.createSheet()
SpreadsheetReadBinary	someVar.readBinary()
SpreadsheetWrite	someVar.write()
SpreadsheetAddRow	someVar.addRow()
SpreadsheetShiftColumns	someVar.shiftColumns()
SpreadsheetGetCellFormula	someVar.getCellFormula()
SpreadsheetDeleteColumns	someVar.deleteColumns()
SpreadsheetAddFreezePane	someVar.addFreezePane()
SpreadsheetDeleteColumn	someVar.deleteColumn()
SpreadsheetSetCellComment	someVar.setCellComment()
SpreadsheetSetActiveSheetNumber	someVar.setActiveSheetNumber()
SpreadsheetSetHeader	someVar.setHeader()
SpreadsheetAddSplitPane	someVar.addSplitPane()
SpreadsheetMergeCells	someVar.mergeCells()
SpreadsheetFormatRows	someVar.formatRows()
SpreadsheetGetCellComment	someVar.getCellComment()
SpreadsheetGetCellValue	someVar.getCellValue()
SpreadsheetAddInfo	someVar.addInfo()
SpreadsheetSetCellValue	someVar.setCellValue()
SpreadsheetSetFooter	someVar.setFooter()
SpreadsheetRemoveSheet	someVar.removeSheet()
SpreadsheetSetRowHeight	someVar.setRowHeight()
SpreadsheetSetActiveSheet	someVar.setActiveSheet()
SpreadsheetFormatCellRange	someVar.formatCellRange()
SpreadsheetFormatCell	someVar.formatCell()
SpreadsheetAddRows	someVar.addRows()
SpreadsheetFormatColumns	someVar.formatColumns()
SpreadsheetAddImage	someVar.addImage()
SpreadsheetSetCellFormula	someVar.setCellFormula()

© 2014 Adobe Systems Incorporated. All rights reserved.

SpreadsheetAddColumn	someVar.addColumn()
SpreadsheetDeleteRows	someVar.deleteRows()
SpreadsheetSetColumnWidth	someVar.setColumnWidth()
SpreadsheetFormatRow	someVar.formatRow()
SpreadsheetInfo	someVar.info()

Supported XML member functions

The following XML member functions are supported:

XmlTransform	someVar.transform()
XmlGetNodeType	someVar.getNodeType()
XmlChildPos	someVar.childPos()
XmlElemNew	someVar.elemNew()
XmlSearch	someVar.search()

Supported Query member functions

The following Query member functions are supported:

QueryAddColumn	someVar.addColumn()
QueryGetRow	someVar.getRow()
QueryConvertForGrid	someVar.convertForGrid()
QuerySetCell	someVar.setCell()
QueryAddRow	someVar.addRow()
	someVar.getResult()

© 2014 Adobe Systems Incorporated. All rights reserved.

Developing CFML Applications

In this section, you can understand about building optimized and secured application, using persistent data and locking techniques, and using the ColdFusion threads effectively. You can additionally understand the debugging and troubleshooting techniques.

- Designing and Optimizing a ColdFusion Application
- Handling Errors
- Using Persistent Data and Locking
- Using ColdFusion Threads
- Securing Applications
- Developing Globalized Applications
- Debugging and Troubleshooting Applications
- Using the ColdFusion Debugger
- Client-side CFML (for mobile development)
- Social Enhancements
- REST Enhancements in ColdFusion 11
- Authentication through OAuth

#back to top

© 2014 Adobe Systems Incorporated. All rights reserved.

Designing and Optimizing a ColdFusion Application

Application elements and how you structure an application on your server make your Adobe ColdFusion pages an effective Internet application. You use the Application.cfc and Application.cfm files and various coding methods to optimize the efficiency of your application.

© 2014 Adobe Systems Incorporated. All rights reserved.

About applications

The term *application* can mean many things. An application can be as simple as a guest book or as sophisticated as a full Internet commerce system with catalog pages, shopping carts, and reporting.
An application, however, has a specific meaning in ColdFusion. A ColdFusion application has the following characteristics:

- It consists of one or more ColdFusion pages that work together and share a common set of resources.
- All pages in the application share an application name and configuration settings as specified in an Application.cfc file or a `cfapplication` tag.
- All pages in the application share variables in the Application scope.
- You can write application-wide event handlers for specific events, such as request start or session end.
  What appears to a user to be a single application (for example, a company's website), can consist of multiple ColdFusion applications.
  ColdFusion applications are not J2EE applications. However, if you do not specify an application name in your Application.cfc file or `cfapplication` tag, the Application scope corresponds to the J2EE application servlet context.
  ColdFusion applications end when the application has been inactive for the application time-out period or the server stops. When the application times out, ColdFusion releases all Application scope variables. You must, therefore, select a time-out period that balances the need for clearing Application scope memory and the overhead of re-creating the scope. A typical application time-out is two days.
  ColdFusion applications and sessions are independent of each other. For example, if an application times out while a user's session is active, the session continues and the session context, including the user's Session scope variables, is unaffected by the application ending and restarting.
  Although, there are no definite rules about how you represent your web application as a ColdFusion application or applications, the following guidelines are useful:

- Application pages share a common general purpose. For example, a web storefront is typically a single ColdFusion application.
- Many, but not necessarily all, pages in a ColdFusion application share data or common code elements, such as a single login mechanism.
- Application pages share a common look-and-feel, often enforced by using common code elements, such as the same header and footer pages, and a common error message template.

© 2014 Adobe Systems Incorporated. All rights reserved.

Elements of a ColdFusion application

Before you develop a ColdFusion application, determine how to structure the application and how to handle application-wide needs and issues. In particular, consider all of the following:

- The overall application framework
- Reusable application elements
- Shared variables
- Application events and the Application.cfc file
- Application-level settings and functions
- Application security and user identification

The application framework

The application framework is the overall structure of the application and how your directory structure and application pages reflect that structure. Use a single application framework to structure multiple ColdFusion applications into a single website or Internet application. You can structure a ColdFusion application by using many methodologies. For example, the Fusebox application development methodology is one popular framework for developing ColdFusion web applications. (For more information on Fusebox, see www.fusebox.org.)
Information on how to use or develop a specific application framework is not provided. However, there is information about the tools that ColdFusion provides for building your framework, including the Application.cfc file, how an application's directory structure affects the application, and how you map the directory structure. For more information on mapping the application framework, see Structuring an application.

Reusable application elements

ColdFusion provides a variety of reusable elements that you use to provide commonly used functionality and extend CFML. These elements include the following:

- User-defined functions (UDFs)
- CFML custom tags
- ColdFusion components (CFCs)
- CFX (ColdFusion Extension) tags
- Pages that you include using the `cfinclude` tag
  For an overview of these elements, and information about how to choose among them, see Creating ColdFusion Elements.

Shared variables

The following ColdFusion variable scopes maintain data that lasts beyond the scope of the current HTTP request:

Variable scope	Variables available
Server	To all applications on a server and all clients
Application	To all pages in an application for all clients
Client	For a single client browser over multiple browser sessions in one application
Session	For a single client browser for a single browser session in one application

For more information on using these variables, including how to use locks to ensure that the data they contain

© 2014 Adobe Systems Incorporated. All rights reserved.

remains accurate, see Using Persistent Data and Locking.

Application events and the Application.cfc file

Application events are specific occurrences during the life cycle of an application. Each time one of these events occurs, ColdFusion runs the corresponding method in your Application.cfc file (also referred to as the *application CFC*). The Application.cfc file defines application settings and implements methods to handle the application events.

Implement application CFC methods to handle the following events:

Event	Trigger
Application start	ColdFusion starts processing the first request for a page in an application that is not running.
Application end	An application time-out setting is reached or the server shuts down.
Session start	A new session is created as a result of a request that is not in an existing session.
Session end	A session time-out setting is reached.
Request start	ColdFusion receives a request, including HTTP requests, messages to the event gateway, SOAP requests, or Flash Remoting requests.
Request	Immediately after ColdFusion finishes processing the request start event. The handler for this event is intended for use as a filter for the request contents. For more information on the differences between request start and request events, see *Managing requests in Application.cfc* in Defining the application and its event handlers in Application.cfc.
Request end	ColdFusion finishes processing all pages and CFCs for the request.
Exceptions	An exception occurs that is not handled in a try/catch block.

The Application.cfc file also defines application-wide settings, including the application name and whether the application supports Session variables.
For more information on using application events and the Application.cfc file, see Defining the application and its event handlers in Application.cfc.

Other application-level settings and functions

Adobe recommends that when defining application-level settings, variables, and functions in new code, you do not use the techniques used previous to ColdFusion MX 7. Instead, use the Application.cfc file and its variables and methods, which provide more features and include logical, hierarchical structure.

© 2014 Adobe Systems Incorporated. All rights reserved.

If you do not have an Application.cfc file, ColdFusion processes the following two pages, if they are available, every time it processes any page in the application:

- The Application.cfm page is processed before each page in the application.
- The OnRequestEnd.cfm page is processed after each page in the application.

> **Note**
>
> UNIX systems are case-sensitive. To ensure that your pages work on UNIX, always capitalize the A in Application.cfm and the O, R, and E in OnRequestEnd.cfm.

The Application.cfm page can define the application. It can contain the `cfapplication` tag that specifies the application name, and code on this page is processed for all pages in the application. This page can define application-level settings, functions, and features.
The OnRequestEnd.cfm page is used in fewer applications than the Application.cfm page. It lets you provide common clean-up code that gets processed after all application pages, or specify dynamic footer pages.

> The OnRequestEnd.cfm page does not execute if the page runs a `cflocation_` tag._

For more information on the Application.cfm and OnRequestEnd.cfm pages, see Using an Application.cfm page. For information on placing these pages in the application directory structure, see Structuring an application.

> **Note**
>
> You can create a ColdFusion application without using an Application.cfc, Application.cfm, or OnRequestEnd.cfm page. However, it is much easier to use the Application.cfm page than to have each page in the application use a `cfapplication` tag and define common application elements.

Specifying settings per application

Set the following on a per-application basis:

- Mappings
- Custom tag paths
  These settings override the server-side settings in the ColdFusion Administrator for the specified application only. Specifying per application settings does not change the server-wide settings. To set per-application settings, first enable per-application settings on the Settings page of the ColdFusion Administrator. You then set the mappings or custom tag paths in the Application.cfc file.
  Custom Tags in per-application settings override those defined in the ColdFusion Administrator. For example, if you have two custom tags of the same name and they are in different locations in the Administrator and per-application settings, the one in the per-application settings is taken first.

> **Note**
>
> Per-application settings are supported in applications that use an Application.cfc file only, not in applications that use an Application.cfm file. The per-application settings do not work if you have disabled application variables in the Memory Variables page of the Administrator.

Set the mappings per application

1. Check the Enable Per App Settings option on the Settings page of the ColdFusion Administrator.
2. Include code like the following in your Application.cfc file:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cfset THIS.mappings["/MyMap"]="c:\inetpub\myStuff">
```

or

```
<cfset StructInsert(THIS.mappings, "/MyMap", "c:\inetpub\myStuff")>
```

(To use the second format, you must first create a `THIS.mappings` structure.)

Set the custom tag paths per application

1. Check the Enable Per App Settings option on the Settings page of the ColdFusion Administrator.
2. Include code like the following in your Application.cfc file:

```
<cfset customtagpaths = "c:\mapped1,c:\mapped2">
<cfset customtagpaths = ListAppend(customtagpaths,"c:\mapped3")>
<cfset This.customtagpaths = customtagpaths>
```

Application security and user identification

All applications must ensure that malicious users cannot make improper use of their resources. Additionally, many applications require user identification, typically to control the portions of a site that the user accesses, to control the operations that the user performs, or to provide user-specific content. ColdFusion provides the following forms of application security to address these issues:

- **Resource (file and directory-based) security** Limits the ColdFusion resources, such as tags, functions, and data sources that application pages, in particular directories, access. Consider the resource security needs of your application when you design the application directory structure.
- **User (programmatic) security** Provides an authentication (login) mechanism and a role-based authorization mechanism to ensure that users only access and use selected features of the application. User security also incorporates a user ID, which you use to customize page content. To implement user security, you include security code, such as the `cflogin` and `cfloginuser` tags, in your application.For more on implementing security, see Securing Applications.

© 2014 Adobe Systems Incorporated. All rights reserved.

Structuring an application

When you design a ColdFusion application, structure its contents into directories and files, also known as mapping the directory structure. This activity is an important step in designing a ColdFusion application. Before you start building the application, establish a root directory for the application. You store application pages in subdirectories of the root directory.

How ColdFusion finds and process application definition pages

ColdFusion uses the following rules to locate and process the Application.cfc, Application.cfm, and OnRequestEnd.cfm pages that define application-specific elements. The way ColdFusion locates these files helps determine how you structure an application.
Each time ColdFusion processes a page request it does the following:

1. When ColdFusion starts processing the request, it does the following:
   - It searches the page's directory for a file named Application.cfc. If one exists, it creates a new instance of the CFC, processes the initial events, and stops searching. (ColdFusion creates a new instance of the CFC and processes its initialization code for each request.)
   - If the requested page's directory does not have an Application.cfc file, it checks the directory for an Application.cfm file. If one exists, ColdFusion logically includes the Application.cfm page at the beginning of the requested page and stops searching further.
   - If the requested page's directory does not have an Application.cfc or Application.cfm file, ColdFusion searches up the directory tree and checks each directory first for an Application.cfc file and then, if one is not found, for an Application.cfm page, until it reaches the root directory (such as C:). When it finds an Application.cfc or Application.cfm file, it processes the page and stops searching.
2. ColdFusion processes the requested page's contents.
3. When the request ends, ColdFusion does the following:
   - If you have an Application.cfc, ColdFusion processes the CFC's `onRequestEnd` method and releases the CFC instance.
   - If you do not have an Application.cfc, but do have an Application.cfm page, ColdFusion looks for an OnRequestEnd.cfm in the same directory as the Application.cfm page ColdFusion uses for the current page. ColdFusion does not search beyond that directory, so it does not run an OnRequestEnd.cfm page that resides in another directory. Also, the OnRequestEnd.cfm page does not run if there is an error or an exception on the application page, or if the application page executes the `cfabort` or `cfexit` tag.
     The following rules determine how ColdFusion processes application pages and settings:

- ColdFusion processes only one Application.cfc or Application.cfm page for each request. If a ColdFusion page has a `cfinclude` tag pointing to an additional ColdFusion page, ColdFusion does not search for an Application.cfc or Application.cfm page when it includes the additional page.
- If a ColdFusion page has a `cfapplication` tag, it first processes any Application.cfc or Application.cfm, and then processes the `cfapplication` tag. The tag overrides the settings from the application files, including the application name and the behaviors set by the `cfapplication` tag attributes.
- You can have multiple Application.cfc files, Application.cfm files, and `cfapplication` tags that use the same application name. In this case, all pages that have the same name share the same application settings and Application scope and set and get all the variables in this scope. ColdFusion uses the parameter settings of the `cfapplication` tag or the most recently processed file, if the settings, such as the session time-out, differ among the files.

Note

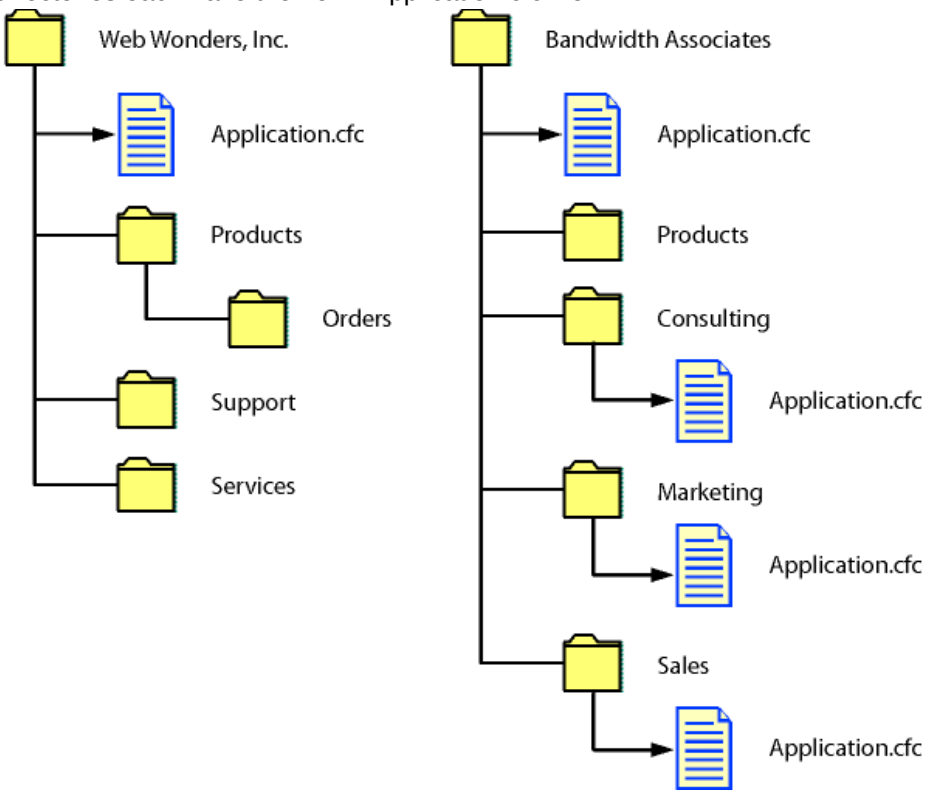
If your application runs on a UNIX platform, which is case-sensitive, spell Application.cfc, Application.cfm, and OnRequestEnd.cfm with capital letters.

© 2014 Adobe Systems Incorporated. All rights reserved.

Defining the directory structure

Defining an application directory structure with an application-specific root directory has the following advantages:

- **Development** The application is easier to develop and maintain, because the application page files are well-organized.
- **Portability** You can easily move the application to another server or another part of a server without changing any code in the application page files.
- **Application-level settings** Application pages that are under the same directory can share application-level settings and functions.
- **Security**Application pages that are under the same directory can share web server security settings.When you place your application in an application-specific directory hierarchy, you can use a single application definition (Application.cfc or Application.cfm) page in the application root directory, or place different application definition pages that govern individual sections of the application in different directories. You divide your logical web application into multiple ColdFusion applications by using multiple application definition pages with different application names. Alternatively, use multiple application definition pages that specify the same application name, but have different code, for different subsections of your application.The directory trees in the following image show two approaches to implementing an application framework:
    - In the example on the left, a company named Web Wonders, Inc. uses a single Application.cfc file installed in the application root directory to process all application page requests.
    - In the example on the right, Bandwidth Associates uses the settings in individual Application.cfc files to create individual ColdFusion applications at the departmental level. Only the Products application pages are processed using the settings in the root Application.cfc file. The Consulting, Marketing, and Sales directories each have their own Application.cfc file.



© 2014 Adobe Systems Incorporated. All rights reserved.

Defining the application and its event handlers in Application.cfc

The Application.cfc file defines application-wide settings and variables, and application event handlers:

- Application-wide settings and variables include page processing settings, default variables, data sources, style settings, and other application-level constants.
- Application event handlers are CFC methods that ColdFusion automatically executes when specific events occur during the lifetime of an application: application start and end, session start and end, request start, execution, and end, and exceptions.

Defining application-level settings and variables

When you create an application, you can set many application-wide properties and characteristics, including the following items:

- Application name
- Application properties, including Client-, Application-, and Session-variable management options
- Page processing options
- Default variables, data sources, style settings, and other application-level constants
  For information on setting default variables, see *Setting application default variables and constants in onApplicationStart* in this page.

Naming the application

Define the application and give it a name by setting the This.name variable in the Application.cfc initialization section, before the method definitions. By using a specific application name, you define a set of pages as part of the same logical application.
ColdFusion supports unnamed applications, which are useful for ColdFusion applications that must interoperate with JSP tags and servlets. Consider creating an unnamed application *only* if your ColdFusion pages must share Application or Session scope data with existing JSP pages and servlets. You cannot have more than one unnamed application on a server instance. For more information on using unnamed applications, see *Sharing data between ColdFusion pages and JSP pages or servlets* in Interoperating with JSP pages and servlets.

Setting application properties

Specify application properties by setting This scope variables in the Application.cfc initialization code. (These are the same properties that you set in the `cfapplication` tag.) The following table lists the This scope variable that ColdFusion uses to set application properties and describes their uses.

Variable	Default	Description
applicationTimeout	Administrator value	Life span, as a real number of days, of the application, including all Application scope variables. Use the `createTimeSpan` function to generate this variable.
clientManagement	False	Whether the application supports Client scope variables.
clientStorage	Administrator value	Where Client variables are stored; can be cookie, registry, or the name of a data source.

© 2014 Adobe Systems Incorporated. All rights reserved.

loginStorage	Cookie	Whether to store login information in the Cookie scope or the Session scope.
scriptProtect	Administrator Value	Whether to protect variables from cross-site scripting attacks.
sessionManagement	False	Whether the application supports Session scope variables.
sessionTimeout	Administrator Value	Life span, as a real number of days, of the user session, including all Session variables. Use the `createTimeSpan` function to generate this variable.
setClientCookies	True	Whether to send CFID and CFTOKEN cookies to the client browser.
setDomainCookies	False	Whether to use domain cookies for the CFID and CFTOKEN values used for client identification, and for Client scope variables stored using cookies. If False, ColdFusion uses host-specific cookies. Set to True for applications running on clusters.

The following example code from the top of an Application.cfc sets the application name and properties:

```
<cfcomponent>
<cfset This.name = "TestApplication">
<cfset This.clientmanagement="True">
<cfset This.loginstorage="Session">
<cfset This.sessionmanagement="True">
<cfset This.sessiontimeout="#createtimespan(0,0,10,0)#">
<cfset This.applicationtimeout="#createtimespan(5,0,0,0)#">
```

For more information on these settings, see `cfapplication` in the *CFML Reference*.

Setting page processing options

The `cfsetting` tag lets you specify the following page processing attributes to apply to all pages in your application:

Attribute	Use

© 2014 Adobe Systems Incorporated. All rights reserved.

showDebugOutput	Specifies whether to show debugging output. This setting cannot enable debugging if it is disabled in the ColdFusion Administrator. However, this option ensures that debugging output is not displayed, even if the Administrator enables it.
requestTimeout	Specifies the page request time-out. If ColdFusion cannot complete processing a page within the time-out period, it generates an error. This setting overrides the setting in the ColdFusion Administrator. Use this setting to increase the page time-out if your application or page frequently accesses external resources that are slow, such as external LDAP servers or web services providers.
enableCFOutputOnly	Disables output of text that is not included inside `cfoutput` tags. This setting helps ensure that extraneous text in your ColdFusion pages does not get displayed.

Often, you use the `cfsetting` tag on individual pages, but you can also use it in your Application.cfc file. For example, using it in multi-application environment to override the ColdFusion Administrator settings in one application.
You can place an application-wide `cfsetting` tag in the component initialization code, normally following the This scope application property settings, as the following example shows:

```
<cfcomponent>
<cfscript>
 This.name="MyAppl";
 This.clientmanagement="True";
 This.loginstorage="Session" ;
 This.sessionmanagement="True" ;
 This.sessiontimeout=CreateTimeSpan(0,0,1,0);
</cfscript>
<cfsetting showdebugoutput="No" enablecfoutputonly="No">
```

The `cfsetting` tag in this example affects all pages in an application. You can override the application-wide settings in the event methods, such as `onRequestStart`, or on individual ColdFusion pages.

Using application event handlers

The following table describes the application event CFC methods that you can implement, including when they are triggered.

Method	When run

© 2014 Adobe Systems Incorporated. All rights reserved.

onApplicationStart	The application first starts: when the first request for a page is processed or the first CFC method is invoked by an event gateway instance, Flash Remoting request, or a web service invocation. This method is useful for setting application-wide (Application scope) variables, such as the names of data sources.
onApplicationEnd	The application ends: when the application times out or the server shuts down.
onSessionStart	A new session is created as a result of a request that is not in an existing session, including ColdFusion event gateway sessions. The application must enable sessions for this event to happen.
onSessionEnd	A session time-out setting is reached. This event is **not** triggered when the application ends or the server shuts down.
onRequestStart	ColdFusion receives any of the following: a request, an HTTP request (for example, from a browser), a message to an event gateway, a SOAP request, or a Flash Remoting request.
onRequest	The `onRequestStart` event has completed. This method acts as a filter for the requested page content.
onRequestEnd	All pages and CFCs in the request have been processed: equivalent to the OnRequestEnd.cfm page.
onMissingTemplate	When ColdFusion receives a request for a nonexistent page.
onError	When an exception occurs that is not caught by a try/catch block.

When ColdFusion receives a request, it instantiates the Application CFC and runs the Application.cfc code in the following order:

- CFC initialization code at the top of the file
- `onApplicationStart`, if not run before for this application
- `onSessionStart`, if not run before for this session
- `onRequestStart`
- `onRequest`, or the requested page if there is no onRequest method
- `onRequestEnd`
  The following methods are triggered by specific events:

- `onApplicationEnd`
- `onSessionEnd`
- `onMissingTemplate`
- `onError`

© 2014 Adobe Systems Incorporated. All rights reserved.

The `onApplicationEnd` and `onSessionEnd` methods do not execute in the context of a page request, so they cannot access request variables or display information to the user. The `onMissingTemplate` method is triggered when a URL specifies a CFML page that does not exist. The `OnError` method does not always execute in the context of a request; use its `Event` argument to determine the context.

Managing the application with Application.cfc

Use the `onApplicationStart` and `onApplicationEnd` methods to configure and manage the application; that is, to control resources that are used by multiple pages and requests and must be consistently available to all code in your application. Such resources include data sources, application counters such as page hit variables, or style information for all pages.

The `onApplicationStart` method executes when ColdFusion gets the first request for a page in the application after the server starts. The `onApplicationEnd` method executes when the application server shuts down or if the application is inactive for the application time-out period.

The following are some of the ways you use these methods. For more information, see entries for onApplicationStart and onApplicationEnd in the *CFML Reference*.

Defining application utility functions

Functions that you define in Application.cfc and do not place in a shared scope are, by default, available only to other methods in the CFC.

If your Application.cfc implements the `onRequest` method, any utility functions that you define in Application.cfc are also directly available in to the target page, because Application.cfc and the target page share the Variables scope.

If your application requires utility functions that are used by multiple pages, not just by the methods in Application.cfc, and you do not use an `onRequest` method, Adobe recommends that you place them in a separate CFC and access them by running that CFC. As with other ColdFusion pages, Application.cfc can access any CFC in a directory path that is configured on the ColdFusion Administrator Mappings page. Therefore, use this technique to share utility functions across applications.

If your Application.cfc defines utility functions that you want available on request pages and does not use an `onRequest` method, explicitly place the functions in a ColdFusion scope, such as the Request scope, as the following code shows:

```
<cffunction name="theFunctionName" returntype="theReturnType">
 <!--- Function definition goes here. --->
</cffunction>

<cffunction name="OnRequestStart">
 <!--- OnRequestStart body goes here --->
  <cfset Request.theFunctionName=This.theFunctionName>
</cffunction>
```

On the request page, you would include the following code:

```
<cfset myVar=Request.theFunctionName(Argument1...)>
```

Functions that you define in this manner share the This scope and Variables scope with the Application.cfc file for the request.

Setting application default variables and constants in onApplicationStart

© 2014 Adobe Systems Incorporated. All rights reserved.

You can set default variables and application-level constants in Application.cfc. For example, you can do the following:

- Specify a data source and ensure that it is available
- Specify domain name
- Set styles, such as fonts or colors
- Set other application-level variables
  You do not have to lock Application scope variables when you set them in the Application.cfc `onApplicationStart` method.
  For details on implementing the `onApplicationStart` method, see onApplicationStart in the *CFML Reference*.

Using the onApplicationEnd method

Use the `onApplicationEnd` method for any clean-up activities that your application requires when it shuts down, such as saving data in memory to a database, or to log the application end. You cannot use this method to display data on a user page, because it is not associated with any request. The application ends, even if this method throws an exception. An application that is used often is unlikely to execute this method, except when the server is shut down.
For details on implementing the `onApplicationEnd` method, see onApplicationEnd in the *CFML Reference*.

Managing sessions in Application.cfc

Use the `onSessionStart` and `onSessionEnd` methods to configure and manage user sessions; that is, to control resources that are used by multiple pages while a user is accessing your site from during a single browser session. The session begins when a user first requests a page in your application, and ends when the session times out. For more information on Session scope and Session variables, see Using Persistent Data and Locking.
Session resources include variables that store data that is needed throughout the session, such as account numbers, shopping cart contents, or CFCs that contain methods and data that are used by multiple pages in a session.

> **Note**
>
> Do not include the `cflogin` tag or basic login processing in the onSessionStart method, as the code executes only at the start of the session; it cannot handle user logout, and cannot fully ensure security.

Using the onSessionStart method

This method is useful for initializing session data, such as user settings or shopping cart contents, or for tracking the number of active sessions. You do not need to lock the Session scope when you set session variables in this method.
For more information, see the onSessionStart entry in the *CFML Reference*.

Using the onSessionEnd method

Use this method for any clean-up activities when the session ends. (For information on ending sessions, see *Ending a session* in Configuring and using session variables.) For example, you can save session-related data, such as shopping cart contents or information about whether the user has not completed an order, in a database, or you can log the end of the session to a file. You cannot use this method to display data on a user page, because it is not associated with a request.

© 2014 Adobe Systems Incorporated. All rights reserved.

> **Note**
>
> Sessions do not end, and the `onSessionEnd` method is not called when an application ends. For more information, see the onSessionEnd entry in the CFML Reference.

Managing requests in Application.cfc

ColdFusion provides three methods for managing requests: `onRequestStart`, `onRequest`, and `onRequestEnd`. ColdFusion processes requests, including these methods, as follows:

1. ColdFusion always processes `onRequestStart` at the start of the request.
2. If you implement an `onRequest` method, ColdFusion processes it; otherwise, it processes the requested page. If you implement an `onRequest` method, explicitly call the requested page in your `onRequest` method .
3. ColdFusion always processes `onRequestEnd` at the end of the request.
   You can use each of the Application.cfc request methods to manage requests as follows:

Using the onRequestStart method

This method runs at the beginning of the request. It is useful for user authorization (login handling), and for request-specific variable initialization, such as gathering performance statistics.
If you use the `onRequestStart` method and do not use the `onRequest` method, ColdFusion automatically processes the request when it finishes processing the `onRequestStart` code.

> **Note**
>
> If you do not include an `onRequest` method in Application.cfm file, the `onRequestStart` meth od does not share a Variables scope with the requested page, but it does share Request scope variables.

For more information, see the entry for onRequestStart in the *CFML Reference*

User authentication

When an application requires a user to log in, include the authentication code, including the `cflogin` tag or code that calls this tag, in the `onRequestStart` method. Doing so ensures that the user is authenticated at the start of each request. For detailed information on security and creating logins, see Securing Applications For an example that uses authentication code generated by the Adobe Dreamweaver CF Login Wizard, see onRequestStart in the *C FML Reference*.

Using the onRequest method

The `onRequest` method differs from the `onRequestStart` method in one major way: the `onRequest` method intercepts the user's request. This difference has two implications:

- ColdFusion does not process the request unless you explicitly call it, for example, by using a `cfinclude` tag. This behavior lets you use the `onRequest` method to filter requested page content or to implement a switch that determines the pages or page contents to be displayed.
- When you use `cfinclude` to process request, the CFC instance shares the Variables scope with the requested page. As a result, any method in the CFC that executes can set the page's Variables scope variables, and the `onRequestEnd` method can access any Variable scope values that the included page has set or changed. Therefore, for example, the `onRequestStart` or `onRequest` method set variables that are used on the page.

© 2014 Adobe Systems Incorporated. All rights reserved.

To use this method as a filter, place the `cfinclude` tag inside a `cfsavecontent` tag, as the following example shows:

```
<cffunction name="onRequest">
 <cfargument name = "targetPage" type="String" required=true/>
 <cfsavecontent variable="content">
  <cfinclude template=#Arguments.targetPage#>
 </cfsavecontent>
 <cfoutput>
  #replace(content, "report", "MyCompany Quarterly Report", "all")#
 </cfoutput>
</cffunction>
```

For more information, see the entry for onRequest in the *CFML Reference*

Using the onRequestEnd method

You use the `onRequestEnd` method for code that should run at the end of each request. (In ColdFusion versions through ColdFusion MX 6.1, you would use the OnRequestEnd.cfm page for such code). Typical uses include displaying dynamic footer pages. For an example, see onSessionEnd in the *CFML Reference*.

> **Note**
>
> If you do not include an `onRequest` method in Application.cfm file, the `onRequestEnd` method does not share a Variables scope with the requested page, but it does share Request scope variables.

For more information, see the entry for onRequestEnd in the *CFML Reference*.

Handling errors in Application.cfc

The following sections briefly describe how you to handle errors in Application.cfc. For more information on error pages and error handling, see Handling Errors For details on implementing the onError method, see onError in the *C FML Reference*.

Application.cfc error handling techniques

Application.cfc handles errors in any combination of the following ways:

- Use try/catch error handling in the event methods, such as `onApplicationStart` or `onRequestStart`, to handle exceptions that happen in the event methods.
- Implement the onError method. This method receives all exceptions that are not directly handled by try/catch handlers in CFML code. The method can use the `cfthrow` tag to pass any errors it does not handle to ColdFusion for handling.{{}}
- Use `cferror` tags in the application initialization code following the `cfcomponent` tag, typically following the code that sets the application's This scope variables. These tags specify error processing if you do not implement an `onError` method, or if the `onError`method throws an error. You could implement an application-specific validation error handler, for example, by placing the following tag in the CFC initialization code:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cferror type="VALIDATION" template="validationerrorhandler.cfm">
```

- The ColdFusion default error mechanisms handle any errors that are not handled by the preceding techniques. These mechanisms include the site-wide error handler that you specify in the ColdFusion Administrator and the built-in default error pages.
  These techniques let you include application-specific information, such as contact information or application or version identifiers, in the error message, and let you display all error messages in the application in a consistent manner. You use Application.cfc to develop sophisticated application-wide error-handling techniques, including error-handling methods that provide specific messages, or use structured error-handling techniques.

Note

The `onError` method catches errors that occur in the `onSessionEnd` and `onApplicationEnd` application event methods. It does not display messages to the user, however, because there is no request context. The `onError` function logs errors that occur when the session or application ends.

Handling server-side validation errors in the onError method

Server-side validation errors are actually ColdFusion exceptions; as a result, if your application uses an `onError` method, this method gets the error and must handle it or pass it on to ColdFusion for handling.
To identify a server-side validation error, search the Arguments.*Exception*.StackTrace field for coldfusion.filter.FormValidationException. You can then handle the error directly in your `onError` routine, or throw the error so that either the ColdFusion default validation error page or a page specified by an `cferror` tag in your Application.cfc initialization code handles it.

Example: error Handling with the onError method

The following Application.cfc file has an `onError` method that handles errors as follows:

- If the error is a server-side validation error, the `onError` method throws the error for handling by ColdFusion, which displays its standard validation error message.
- For any other type of exception, the `onError` method displays the name of the event method in which the error occurred and dumps the exception information. In this example, because you generate errors on the CFM page only, and not in a Application.cfc method, the event name is always the empty string.

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cfcomponent>
<cfset This.name = "BugTestApplication">
<cffunction name="onError">
 <!--- The onError method gets two arguments:
   An exception structure, which is identical to a cfcatch variable.
   The name of the Application.cfc method, if any, in which the error
   happened.
 <cfargument name="Except" required=true/>
 <cfargument type="String" name = "EventName" required=true/>
 <!--- Log all errors in an application-specific log file. --->
 <cflog file="#This.Name#" type="error" text="Event Name: #Eventname#" >
 <cflog file="#This.Name#" type="error" text="Message: #except.message#">
 <!--- Throw validation errors to ColdFusion for handling. --->
 <cfif Find("coldfusion.filter.FormValidationException",
      Arguments.Except.StackTrace)>
  <cfthrow object="#except#">
 <cfelse>
  <!--- You can replace this cfoutput tag with application-specific
    error-handling code. --->
  <cfoutput>
   <p>Error Event: #EventName#</p>
   <p>Error details:<br>
   <cfdump var=#except#></p>
  </cfoutput>
 </cfif>
</cffunction>
</cfcomponent>
```

To test this example, place a CFML page with the following code in the same page as the Application.cfc file, and enter valid and invalid text in the text input field.

```
<cfform>
 This box does Integer validation:
 <cfinput name="intinput" type="Text" validate="integer" validateat="onServer"><br>
 Check this box to throw an error on the action page:
 <cfinput type="Checkbox" name="throwerror"><br>
 <cfinput type="submit" name="submitit">
 </cfform>
<cfif IsDefined("form.fieldnames")>
 <cfif IsDefined("form.throwerror")>
  <cfthrow type="ThrownError" message="This error was thrown from the bugTest action
page.">
 <cfelseif form.intinput NEQ "">
  <h3>You entered the following valid data in the field</h3>
  <cfoutput>#form.intinput#</cfoutput>
 </cfif>
</cfif>
```

Note

For more information on server-side validation errors, see Validating data.

© 2014 Adobe Systems Incorporated. All rights reserved.

Example: a complete Application.cfc

The following example is a simplified Application.cfc file that illustrates the basic use of all application event handlers:

```
<cfcomponent>
<cfset This.name = "TestApplication">
<cfset This.Sessionmanagement=true>
<cfset This.Sessiontimeout="#createtimespan(0,0,10,0)#">
<cfset This.applicationtimeout="#createtimespan(5,0,0,0)#">

<cffunction name="onApplicationStart">
 <cftry>
  <!--- Test whether the DB that this application uses is accessible
    by selecting some data. --->
  <cfquery name="testDB" dataSource="cfdocexamples" maxrows="2">
   SELECT Emp_ID FROM employee
  </cfquery>
  <!--- If we get database error, report it to the user, log the error
   information, and do not start the application. --->
  <cfcatch type="database">
   <cfoutput>
    This application encountered an error.<br>
    Please contact support.
   </cfoutput>
   <cflog file="#This.Name#" type="error"
    text="cfdocexamples DB not available. message: #cfcatch.message#
    Detail: #cfcatch.detail# Native Error: #cfcatch.NativeErrorCode#">
   <cfreturn False>
  </cfcatch>
 </cftry>
 <cflog file="#This.Name#" type="Information" text="Application Started">
 <!--- You do not have to lock code in the onApplicationStart method that sets
Application scope variables. --->
 <cfscript>
  Application.availableResources=0;
  Application.counter1=1;
  Application.sessions=0;
 </cfscript>
 <!--- You do not need to return True if you don't set the cffunction returntype
attribute. --->
 </cffunction>

<cffunction name="onApplicationEnd">
 <cfargument name="ApplicationScope" required=true/>
 <cflog file="#This.Name#" type="Information"
  text="Application #ApplicationScope.applicationname# Ended">
</cffunction>


<cffunction name="onRequestStart">
 <!--- Authentication code, generated by the Dreamweaver Login Wizard,
  makes sure that a user is logged in, and if not displays a login page. --->
 <cfinclude template="mm_wizard_application_include.cfm">
<!--- If it's time for maintenance, tell users to come back later. --->
 <cfscript>
  if ((Hour(now()) gt 1) and (Hour(now()) lt 3)) {
```

© 2014 Adobe Systems Incorporated. All rights reserved.

```
    WriteOutput("The system is undergoing periodic maintenance.
     Please return after 3:00 AM Eastern time.");
    return false;
   } else {
    this.start=now();
   }
  </cfscript>
 </cffunction>

 <cffunction name="onRequest">
  <cfargument name = "targetPage" type="String" required=true/>
  <cfsavecontent variable="content">
   <cfinclude template=#Arguments.targetPage#>
  </cfsavecontent>
  <!--- This is a minimal example of an onRequest filter. --->
  <cfoutput>
   #replace(content, "report", "MyCompany Quarterly Report", "all")#
  </cfoutput>
 </cffunction>

 <!--- Display a different footer for logged in users than for guest users or
    users who have not logged in. --->

 <cffunction name="onRequestEnd">
  <cfargument type="String" name = "targetTemplate" required=true/>
  <cfset theAuthuser=getauthuser()>
  <cfif ((theAuthUser EQ "guest") OR (theAuthUser EQ ""))>
   <cfinclude template="noauthuserfooter.cfm">
  <cfelse>
   <cfinclude template="authuserfooter.cfm">
  </cfif>
 </cffunction>

 <cffunction name="onSessionStart">
  <cfscript>
   Session.started = now();
   Session.shoppingCart = StructNew();
   Session.shoppingCart.items =0;
  </cfscript>
  <cflock timeout="5" throwontimeout="No" type="EXCLUSIVE" scope="SESSION">
   <cfset Application.sessions = Application.sessions + 1>
  </cflock>
  <cflog file="#This.Name#" type="Information" text="Session:
   #Session.sessionid# started">
 </cffunction>

 <cffunction name="onSessionEnd">
  <cfargument name = "SessionScope" required=true/>
  <cflog file="#This.Name#" type="Information" text="Session:
    #arguments.SessionScope.sessionid# ended">
 </cffunction>

 <cffunction name="onError">
  <cfargument name="Exception" required=true/>
  <cfargument type="String" name = "EventName" required=true/>
  <!--- Log all errors. --->
  <cflog file="#This.Name#" type="error" text="Event Name: #Eventname#">
  <cflog file="#This.Name#" type="error" text="Message: #exception.message#">
  <!--- Some exceptions, including server-side validation errors, do not
```

© 2014 Adobe Systems Incorporated. All rights reserved.

```
   generate a rootcause structure. --->
<cfif isdefined("exception.rootcause")>
 <cflog file="#This.Name#" type="error"
  text="Root Cause Message: #exception.rootcause.message#">
</cfif>
<!--- Display an error message if there is a page context. --->
<cfif NOT (Arguments.EventName IS onSessionEnd) OR
  (Arguments.EventName IS onApplicationEnd)>
 <cfoutput>
  <h2>An unexpected error occurred.</h2>
  <p>Please provide the following information to technical support:</p>
  <p>Error Event: #EventName#</p>
  <p>Error details:<br>
  <cfdump var=#exception#></p>
 </cfoutput>
</cfif>
</cffunction>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

```
</cfcomponent>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

Migrating from Application.cfm to Application.cfc

To migrate an existing application that uses Application.cfm to one that uses Application.cfc, do the following:

- Replace the `cfapplication` tag with CFC initialization code that sets the Application.cfc This scope variables that correspond to the tag attributes.
- Place in the `onApplicationStart` method, any code that initializes Application scope variables, and any other application-specific code that executes only when the application starts. Often, such code in Application.cfm is inside a block that tests for the existence of an Application scope switch variable. Remove the variable test and the Application scope lock that surrounds the code that sets the Application scope variables.
- Place in the `onSessionStart` method, any code that initializes Session scope variables, and any other application-specific code that executes only when the session starts. Remove any code that tests for the existence of Session scope variables to be for initialized and the Session scope lock that surrounds the code that sets the Session scope variables.
- Place in the `onRequestStart` method, any `cflogin` tag and related authentication code.
- Place in the `onRequest` method, any code that sets Variables scope variables and add a `cfinclude` tag that includes the page specified by the Arguments of the method.Targetpage variable.
- Place in the `onRequestEnd` method, any code you have in an OnRequestEnd.cfm page.
- Consider replacing `cferror` tags with an `onError` event method. If you do not do so, place the `cferror` ta gs in the CFC initialization code.

© 2014 Adobe Systems Incorporated. All rights reserved.

Using an Application.cfm page

If you do not use an Application.cfc file, use the Application.cfm page to define application-level settings and functions.

Naming the application

Use the `cfapplication` tag to specify the application name and define a set of pages as part of the same logical application. Although you can create an application by placing a `cfapplication` tag with the application name on each page, you normally place the tag in the Application.cfm file; for example:

```
<cfapplication name="SearchApp">
```

Note

The value that you set for the `name` attribute in the `cfapplication` tag is limited to 64 characters.

Setting the client, application, and session variables options

Use the `cfapplication` tag to specify client state and persistent variable use, as follows:

- To use Client scope variables, specify `clientManagement=True.`
- To use Session scope variables, specify `sessionManagment=True`.
  You can also optionally do the following:
- Set application-specific time-outs for Application and Session scope variables. These settings override the default values set in the ColdFusion Administrator.
- Specify a storage method for Client scope variables. This setting overrides the method set in the ColdFusion Administrator.
- Specify not to use cookies on the client browser.
  For more information on configuring these options, see Using Persistent Data and Locking and the *CFML Reference*.

Defining page processing settings

The `cfsetting` tag lets you specify page processing attributes that you want to apply to all pages in your application. For more information, see *Setting page processing options* in Defining the application and its event handlers in Application.cfc.

Setting application default variables and constants

Set default variables and application-level constants on the Application.cfm page. For example, specify the following values:

- A data source
- A domain name
- Style settings, such as fonts or colors
- Other important application-level variables
  Often, an Application.cfm page uses one or more `cfinclude` tags to include libraries of commonly used code, such as user-defined functions, that are required on many of the application's pages.

Processing logins

© 2014 Adobe Systems Incorporated. All rights reserved.

When an application requires a user to log in, you typically place the `cflogin` tag on the Application.cfm page. For detailed information on security and creating logins, including an Application.cfm page that manages user logins, see Securing Applications

Handling errors

Use the `cferror` tag on your Application.cfm page to specify application-specific error-handling pages for request, validation, or exception errors, as shown in the following example. This way you include application-specific information, such as contact information or application or version identifiers, in the error message, and you display all error messages in the application in a consistent manner.
For more information on error pages and error handling, see Handling Errors.

Example: an Application.cfm page

The following example shows a sample Application.cfm file that uses several of the techniques typically used in Application.cfm pages. For the sake of simplicity, it does not show login processing; for a login example, see Securing Applications.

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<!--- Set application name and enable Client and Session variables. --->
<cfapplication name="Products"
 clientmanagement="Yes"
 clientstorage="myCompany"
 sessionmanagement="Yes">

<!--- Set page processing attributes. --->
<cfsetting showDebugOutput="No">

<!--- Set custom global error handling pages for this application.--->
<cferror type="request"
 template="requesterr.cfm"
 mailto="admin@company.com">
<cferror type="validation"
 template="validationerr.cfm">

<!--- Set the Application variables if they aren't defined. --->
<!--- Initialize local app_is_initialized flag to false. --->
<cfset app_is_initialized = False>
<!--- Get a read-only lock. --->
<cflock scope="application" type="readonly" timeout=10>
<!--- Read init flag and store it in local variable. --->
 <cfset app_is_initialized = IsDefined("Application.initialized")>
</cflock>
<!--- Check the local flag. --->
<cfif not app_is_initialized>
<!--- Application variables are not initialized yet.
  Get an exclusive lock to write scope. --->
 <cflock scope="application" type="exclusive" timeout=10>
  <!--- Check the Application scope initialized flag since another request
   could have set the variables after this page released the read-only
   lock. --->
  <cfif not IsDefined("Application.initialized")>
   <!--- Do initializations --->
   <cfset Application.ReadOnlyData.Company = "MyCompany">
   <!--- and so on --->
   <!--- Set the Application scope initialization flag. --->
   <cfset Application.initialized = "yes">
  </cfif>
 </cflock>
</cfif>

<!--- Set a Session variable.--->
<cflock timeout="20" scope="Session" type="exclusive">
 <cfif not IsDefined("session.pagesHit")>
  <cfset session.pagesHit=1>
 <cfelse>
  <cfset session.pagesHit=session.pagesHit+1>
 </cfif>
</cflock>

<!--- Set Application-specific Variables scope variables. --->
<cfset mainpage = "default.cfm">
<cfset current_page = "#cgi.path_info#?#cgi.query_string#">

<!--- Include a file containing user-defined functions called throughout
  the application. --->
<cfinclude template="commonfiles/productudfs.cfm">
```

© 2014 Adobe Systems Incorporated. All rights reserved.

Reviewing the code

The following table describes the code and its function:

Code	Description

`<cfapplication name="Products"`
` clientmanagement="Yes"`
` clientstorage="myCompany"`
` sessionmanagement="Yes">`
	Names the application, enables Client and Session scope variables, and sets the client variable store to the myCompany data source.

`<cfsetting showDebugOutput="No">`	Ensures that debugging output is not displayed, if the ColdFusion Administrator enables it.

`<cferror type="request"`
` template="requesterr.cfm"`
` mailto="admin@company.com">`
`<cferror type="validation"`
` template="validationerr.cfm">`
	Specifies custom error handlers for request and validation errors encountered in the application. Specifies the mailing address for use in the request error handler.

`<cfset app_is_initialized = False>`	Sets the Application scope variables, if they are not already set. For a detailed description of the technique used to set the Application scope variables, see Using Persistent Data and Locking.

`<cflock timeout="20"`
`scope="Session" type="exclusive">`
` <cfif not`
`IsDefined("session.pagesHit")>`
`  <cfset session.pagesHit=1>`
` <cfelse>`
`  <cfset`
`session.pagesHit=session.pagesHit+`
`1>`
` </cfif>`
`</cflock>`
	Sets the Session scope `pagesHit` variable, which counts the number of pages touched in this session. If the variable does not exist, creates it; otherwise, increments it.

© 2014 Adobe Systems Incorporated. All rights reserved.

`<cfset mainpage = "default.cfm">` `<cfset current_page = "#cgi.path_info#?#cgi.query_string#">`	Sets two Variables scope variables that are used throughout the application. Creates the current_page variable dynamically; its value varies from request to request.
`<cfinclude template="commonfiles/productudfs.cfm">`	Includes a library of user-defined functions that are used in most pages in the application.

© 2014 Adobe Systems Incorporated. All rights reserved.

Optimizing ColdFusion applications

You can optimize your ColdFusion application in many ways. Optimizing ColdFusion mostly involves good development and coding practices. For example, good database design and usage is a prime contributor to efficient ColdFusion applications.
In many other topics, the optimization techniques provided are pertinent to the related ColdFusion topic. The following information is about general ColdFusion optimization tools and strategies, and particularly about using CFML caching tags for optimization. There is also information on optimizing database use, an important area for application optimization.
The ColdFusion Administrator provides caching options for ColdFusion pages and SQL queries. For information on these options, see the ColdFusion Administrator online Help and *Configuring and Administering ColdFusion*.
For information on debugging techniques that help you identify slow pages, see Debugging and Troubleshooting Applications.
For additional information on optimizing ColdFusion, see the Adobe ColdFusion support center at www.adobe.com/go/learn_cfu_support_en.

Caching ColdFusion pages that change infrequently

Some ColdFusion pages produce output that changes infrequently. For example, if you have an application that extracts a vendor list from a database or produces a quarterly results summary. Normally, when ColdFusion gets a request for a page in the application, it does all the business logic and display processing that are required to produce the report or generate and display the list. If the results change infrequently, it is an inefficient use of processor resources and bandwidth.
The `cfcache` tag tells ColdFusion to cache the HTML that results from processing a page request in a temporary file on the server. This HTML need not be generated each time the page is requested. When ColdFusion gets a request for a cached ColdFusion page, it retrieves the pregenerated HTML page without having to process the ColdFusion page. ColdFusion can also cache the page on the client. If the client browser has its own cached copy of the page from a previous viewing, ColdFusion instructs the browser to use the client's page rather than resending the page.

> **Note**
>
> The `cfcache` tag caching mechanism considers that each URL is a separate page. Therefore, http://www.mySite.com/view.cfm?id=1 and http://www.mySite.com/view.cfm?id=2 result in two separate cached pages. Because ColdFusion caches a separate page for each unique set of URL parameters, the caching mechanism accommodates pages for which different parameters result in different output.

Using the cfcache tag

You tell ColdFusion to cache the page results by placing a `cfcache` tag on your ColdFusion page before code that outputs text. The tag lets you specify the following information:

- Whether to cache the page results on the server, the client system, or both. The default is both. The default is optimal for pages that are identical for all users. If the pages contain client-specific information, or are secured with ColdFusion user security, set the `action` attribute in the `cfcache` tag to `ClientCache`.
- The directory on the server in which to store the cached pages. The default directory is *cf_root*/cache. It is a good practice to create a separate cache directory for each application. Doing so prevents the `cfcache` tag `flush` action from inappropriately flushing more than one application's caches at a time.
- The time span that indicates how long the page lasts in the cache from when it is stored until it is automatically flushed.
  You can also specify several attributes for accessing a cached page on the web server, including a user name and password (if required by the web server), the port, and the protocol (HTTP or HTTPS) to use to

© 2014 Adobe Systems Incorporated. All rights reserved.

access the page.
Place the `cfcache` tag before any code on your page that generates output, typically at the top of the page body. For example, the following tag tells ColdFusion to cache the page on both the client and the server. On the server, the page is cached in the e:/temp/page_cache directory. ColdFusion retains the cached page for one day.

```
<cfcache timespan="#CreateTimespan(1, 0, 0, 0)#" directory="e:/temp/page_cache">
```

> If an Application.cfm or Application.cfc page displays text (for example, if it includes a header page), use the `cfcache`*tag on the Application.cfm page, in addition to the pages that you cache. Otherwise, ColdFusion displays the Application.cfm page output twice on each cached page.*

Flushing cached pages

ColdFusion automatically flushes any cached page if you change the code on the page. It also automatically flushes pages after the expiration time span passes.
Use the `cfcache` tag with the `action="flush"` attribute to immediately flush one or more cached pages. You can optionally specify the directory that contains the cached pages to be flushed and a URL pattern that identifies the pages to flush. If you do not specify a URL pattern, all pages in the directory are flushed. The URL pattern can include asterisk (*) wildcards to specify parts of the URL that can vary.
When you use the `cfcache` tag to flush cached pages, ColdFusion deletes the pages cached on the server. If a flushed page is cached on the client system, it is deleted, and a new copy gets cached the next time the client tries to access the ColdFusion page.
The following example flushes all the pages in the e:/temp/page_cache/monthly directory that start with HR:

```
<cfcache action="flush" directory="e:/temp/page_cache/monthly" expirURL="HR*">
```

If you have a ColdFusion page that updates data that you use in cached pages, the page that does the updating includes a `cfcache` tag that flushes all pages that use the data.
For more information on the `cfcache` tag, see the *CFML Reference*.

Caching parts of ColdFusion pages

In some cases, your ColdFusion page contains a combination of dynamic information that ColdFusion must generate each time it displays the page, and information that it generates dynamically, but that changes less frequently. ColdFusion 9 provides granular control over caching. You can cache fragments of page that lets you cache the infrequently changed content.The following example illustrates fragment caching:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<!--- Greet the us er. --->
<cf output>
    W elcome to our home page.<br>
    The time is #TimeFormat(Now())#.<br>
    Your lucky number is: #RandRa nge(1,1000)#<br>
    <hr>
</cfoutput>
<!--- If the flag is false, query the DB, and save an image of
    the results output to a variable. ---> <cfcache action="optimal"
timespan="#createtimespan(0,1,0,0)#" idletime="#createtimespan(0,0,30,0)#"><!---
Perform database query. ---><cfquery dataSource="cfartgallery" name="specialQuery">
SELECT * from art
</cfquery>
<!--- Calculate sale price and display the results. --->
        <h2>Check out the following specials</h2>
        <table>
<cfoutput query="specialQuery">
            <tr>
                <td>#artid#</td>
                <td>#Description#</td>
                <td>#price#</td>
       </tr>
</cfoutput>
        </table></cfcache><hr><p>Thank you for visiting us!</p>
```

In this example, the highlighted code creates a page fragment that displays all art records from the table `art`. The cache is set to expire either after one hour or after an idle time of 30 minutes. After the cache is invalidated, cache is recreated the next time the page is accessed thereby displaying updated information (if any).

Caching enhancements in ColdFusion 10

Application-specific caching

You can specify caching at server-level or specific to an application.
To set up cache configuration at the application level, specify the application-specific cache settings file (ehcache.xml) in the Application.cfc as shown in the following examples:
Specify the full path:

```
this.cache.configfile = "c:/cachesettings/ehcache.xml";
```

Specify the relative path with respect to the Application.cfc:

```
this.cache.configfile = "ehcache.xml";
```

If caching is specified at the application level, all cache functions use the application-specific cache configuration.

Example

© 2014 Adobe Systems Incorporated. All rights reserved.

Application.cfc

```
<cfcomponent>
 <cfscript>
  this.name = "appSpecificCacheTest";
  this.cache.configfile = "ehcache.xml";
        this.applicationTimeout = createtimespan(0,0,0,5);

  function onApplicationStart(){
   writelog("In onApplicationStart()");
  }
  function onApplicationEnd(){
   writelog("In onApplicationEnd()");
  }
    </cfscript>
</cfcomponent>
```

cache.cfm

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<html>
<head>
 <title>cfhttp</title>
</head>
<!--- End of header --->
<body>
  <!--- remove object from Application Specific cache --->
        <cfif ArrayLen(cacheGetAllIds()) gt 0>
   <cfset cacheRemove(ArrayToList(cacheGetAllIds()))>
  </cfif>

  <cfset obj1 = structNew()>
        <cfset obj1.name = "xyz">
        <cfoutput>Starting to write to cache..</cfoutput>
        <cfset cachePut("obj1",obj1)>
        <br/>
        <cfoutput>Done!!</cfoutput>

        <cfoutput>Trying to fetch cached item...</cfoutput>
        <cfset obj = cacheGet("obj1")>
  <br/>
  <cfdump var="#obj#">


  <cfscript>
           sleep(15000);
        </cfscript>

        <cfoutput>Trying to fetch cached item after 15 seconds...<br/></cfoutput>
        <cfset obj = cacheGet("obj1")>
  <cfdump var="#obj#">
</body>
</html>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

ehcache.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<ehcache xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:noNamespaceSchemaLocation="ehcache.xsd">

    <diskStore path="java.io.tmpdir"/>

    <cacheManagerEventListenerFactory class="" properties=""/>
    <defaultCache
     maxElementsInMemory="5"
     eternal="false"
     timeToIdleSeconds="30"
     timeToLiveSeconds="5"
     overflowToDisk="true"
     diskSpoolBufferSizeMB="30"
     maxElementsOnDisk="10"
     diskPersistent="false"
     diskExpiryThreadIntervalSeconds="3600"
     memoryStoreEvictionPolicy="LRU"
     clearOnFlush="true"
    />


    <cache name="app1cache"
     maxElementsInMemory="5"
     eternal="false"
     timeToIdleSeconds="60"
  timeToLiveSeconds="5"
     overflowToDisk="false"
     diskSpoolBufferSizeMB="30"
     maxElementsOnDisk="10"
     diskPersistent="false"
     diskExpiryThreadIntervalSeconds="3600"
     memoryStoreEvictionPolicy="LRU"
     clearOnFlush="true"/>

</ehcache>
```

Enhanced query caching using Ehcache

Uses Ehcache

All cache queries are cached using Ehcache.

Default and custom query cache regions

All cache queries are stored in the default query region. There are independent default query regions for server-specific and application-specific cache. You can specify user-defined query cache region using the attribute `cacheRegion` in `cfquery`.If you do not specify a query cache region, the default cache region is used at application or server level (whichever is applicable).

Cache ID

© 2014 Adobe Systems Incorporated. All rights reserved.

You can associate a query with a unique identifier for each cache using the attribute `cacheID`. The ID you specify can be quoted while using the cache functions.Within cache, queries are stored as type `query` like `object` and `template` caches.

Fallback to internal cache

To fall back to internal cache (and not use Ehcache) to cache queries, do either of the following:

- **Server level:** Change the following setting: Check the option Use internal cache to store queries.
- **Application level:** Specify `true` for the following setting:`this.cache.useinternalquerycache=true|false`. The default value is `false`.

Specifying query limit

To specify a limit for maximum number of queries in cache, specify the number of queries in the Application.cfc for the following setting:`this.cache.querysize`

Using Amazon S3 storage

ColdFusion customers can now store data in Amazon S3. The support is extended across almost all tags and functions that take file or directory as input or output.
Storing files in Amazon S3 can be performed in the same manner as storing files on disk. Use a prefix `s3://` to indicate that the files reside on Amazon S3. For example, `s3://testbucket/sample.txt`.

Amazon S3

For using Amazon S3, ColdFusion user must have an S3 account with Amazon.
For concepts and details related to Amazon S3, see the [AmazonS3 Documentation|http://aws.amazon.com/documentation/].

Accessing Amazon S3

Use either of the following URL formats to access Amazon S3 from ColdFusion:

- `s3://bucket/x/y/sample.txt}}Here, {{bucket`is the name of the bucket and the remaining portion of the URL is the key name of the Amazon S3 object. In this case, specify the following authentication information in the Application.cfc:

  ```
  <cfscript>
    this.name ="Object Operations";
    this.s3.accessKeyId = "key_ID";
    this.s3.awsSecretKey = "secret_key";
    this.s3.defaultLocation="location";
    </cfscript>
  ```

  For example,`<cffile action="write" output="S3 Specification" file="s3://testbucket/sample.txt"/>`

- `s3://accessKeyId:awsSecretKey@bucket/x/y/sample.txt`
- This format has the `accessKeyID` and `awsSecretKey` specified in it.
- @acts as the token to indicate the end of authentication information.

© 2014 Adobe Systems Incorporated. All rights reserved.

> **Note**
>
> If you have specified the `accessKeyID` and `awsSecretKey` in both the URL and Application.cfc, then value specified in the URL takes precedence.

Example

```
<cffile action="write" output="S3 Specifications"
file="s3://accessKeyID:awsSecretKey@bucket/x/y/sample.txt"/>
```

Enhancements to Amazon S3 integration

Apart from performance improvements while uploading files to Amazon S3, ColdFusion 10 supports multipart upload where files are split into multiple parts and the parts are uploaded in parallel.
To configure multipart upload support, specify the following settings in the Application.cfc:

```
this.s3.minsizeformultipart=filesize _in_MB
```

The size you specify is treated as minimum, above which file upload is performed as multipart upload. This option is helpful when you have to upload files of huge sizes.

Supported operations

The following are the supported operations on Amazon S3:

- Create and delete bucket
- Get bucket location
- Read, write, copy, and delete object
- List keys in bucket
- Get and set metadata for object or bucket
- Get and set ACL for object or bucket

Bucket operations

Use the `cfdirectory` tag or the `directory` functions to perform the bucket operation (create, delete, or list).

Operation	Tag used	Function	Example

© 2014 Adobe Systems Incorporated. All rights reserved.

Create	`cfdirectory action="create"` The `directory` attribute can only take the path of the bucket. Any additional path results in an error. All other attributes are ignored. While creating S3 bucket, the default bucket location is US. You can change the location using the attribute `storeLocation`. `storeLocation` is a new attribute added to the `cfdirectory` tag. You can specify ACL for the bucket while creating it using the `storeACL` attribute which takes a struct value. For details, see *Setting up access control* in Optimizing ColdFusion applications.	`DirectoryCreate`	`<cfdirectory action="create" directory="s3://bucket1"/>`

© 2014 Adobe Systems Incorporated. All rights reserved.

List keys	`cfdirectory action="list"`	`DirectoryList`	`<cfdirectory action="list" directory="s3://bucket1/X/y" />` Since Amazon S3 does not have the concept of directory, it returns the key name (that is, the full path) of objects contained in the bucket. `Directory` attribute in this case takes the path, for example, `s3://bucket1` in which objects have to be searched. The path that follows the bucket name is used as a prefix to perform the list operation and all the objects that match the prefix are returned. In this case, the following attributes are ignored: `recurse`, `type`, and `sort`.
Delete	`cfdirectory action="delete"`	`DirectoryDelete`	`<cfdirectory action="delete" directory="s3://bucket1"/>`

Note

To verify if the bucket exists and is accessible, use the function `directoryExists`.

Object operations

All object operations are similar to file operations (read, write, copy, and delete). Therefore, the tag `cffile` and the file functions can be used to perform the operations. The following table describes the common scenarios:

© 2014 Adobe Systems Incorporated. All rights reserved.

Operation	Tag used	Function	Example
Read	`cffile action="read"`	`FileRead`	`<cffile action="read" file="s3://testbucket/test.txt" variable="data"/>`
Write	`cffile action="write"`	`FileWrite`	`<cffile action="write" output="#data#" file="s3://testbucket/test.txt"/>`
Delete	`cffile action="delete"`	`FileDelete`	`<cffile action="delete" file="s3://testbucket/test.txt"/>`
Copy	`cffile action="copy"`	`FileCopy`	`<cffile action="copy" source="s3://testbucket/test.txt" destination="s3://bucket2/a/b.txt"/>`

The following are the supported functions:

© 2014 Adobe Systems Incorporated. All rights reserved.

- FileIsEOF
- FileReadBinary
- Filecopy
- FileReadLine
- FileExists
- FileWriteln
- FileOpen
- FileClose
- FileRead
- FileDelete

New attributes in cfdirectory action="create" tag

Attribute Added	Description	Example
storeLocation	Used to change the location of the created bucket. The location can either be EU or US. The default location is US.	`<cfdirectory action="create" directory="s3://<bucketname>" storelocation="US" > <cfdirectory action="create" directory="s3://<bucketname>" storelocation="EU" >`
storeACL	An array of struct where each struct represents a permission or grant as discussed in ACLObject.	`<cfdirectory action="create" directory="s3://<bucketname>" storeACL="ACLObject">`

Setting up access control

Amazon S3 lets you set access control list (ACL) for buckets and objects. The ACLs for buckets and objects are independent. You have to manage them separately. Also, object ACLs do not inherit from the bucket ACLs. ACL consists of multiple Grants where each grant has a grantee and a permission. S3 allows three types of grantees:

- group
- email
- canonical (ID)

The following are the possible permissions:

© 2014 Adobe Systems Incorporated. All rights reserved.

- read
- write
- read_acp
- write_acp
- full_control
  See Amazon S3 ACL Documentation for more details.

ACLObject

`ACLObject` is an array of struct where each struct represents an ACL grant. The grantee details are as follows:

- **group** Must have the keys `Group` (with value `all`, `authenticated`, or `log_delivery`) and `permission`.
- **email** Must have the keys `email` and `permission`.
- **canonical** Must have the keys `Id` and `permission`. `displayName` is optional.

Sample ACLObject

```
all_read = {group="all", permission="read"};
   owner_full = {email="xxx@yyy.com", permission="full_control"};
   aclObj = [owner_full, all_read];
```

Access control functions

storeSetACL

Description
Sets the ACL for object or bucket.

Returns
Nothing

Syntax
`StoreSetACL(url, ACLObject)`

Parameters

Parameter	Description
`url`	Amazon S3 URLs (content or object)
`ACLObject`	An array of struct where each struct represents a permission or grant as discussed in ACLObject.

History
ColdFusion 9 Update 1: Added this function

Usage
Use this function to set full permission. The function overwrites all existing permissions. Only the ones you set in the current context exist.

Example

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cftry>
 <cfset dir = "s3://bucket_name">
 <cfif !directoryExists(dir)>
  <cfset directorycreate(dir)>
 </cfif>

 <cfset perm = structnew()>
 <cfset perm.group = "all">
 <cfset perm.permission = "read">
 <cfset perm1 = structnew()>
 <cfset perm1.email = "email ID">
 <cfset perm1.permission = "FULL_CONTROL">
 <cfset myarrray = arrayNew(1)>
 <cfset myarrray = [perm,perm1]>
 <cfset fileWrite("#dir#/test.txt","This is to test all users permission")>

 <cfset StoreSetACL("#dir#/textl.txt","#myarrray#")>
 <cfset test = StoreGetACL ("#dirkey#/test.txt") >

 <cfdump var= "test">

 <cfcatch>
  <cfdump var="#cfcatch#">
 </cfcatch>
</cftry>
```

storeAddACL

Description
Adds ACL to existing ACL for object or bucket.

Returns
Nothing

Syntax
`StoreAddACL(url, ACLObject)`

Parameters

Parameter	Description
`url`	Amazon S3 URLs (content or object).
`ACLObject`	An array of struct where each struct represents a permission or grant as discussed in ACLObject.

History
ColdFusion 9 Update 1: Added this function

Usage
Use this function to add permissions to the existing ones.

© 2014 Adobe Systems Incorporated. All rights reserved.

Example

```
<cftry>
 <cfset dir = "s3://bucket_name/">
 <cfset perm = structnew()>
 <cfset perm.group = "authenticated">
 <cfset perm.permission = "READ">
 <cfset perm1 = structnew()>
 <cfset perm1.email = "email_ID">
 <cfset perm1.permission = "READ_ACP">
 <cfset myarrray = [perm,perm1]>
 <cfif NOT DirectoryExists(dir)>
  <cfset directoryCreate(dir)>
 </cfif>
 <cfset fileWrite("#dir#/Sample.txt","This is to test StoreAddACL")>
 <cfset StoreAddACL("#dir#","#myarrray#")>
 <cfset test = StoreGetACL(dirkey)>
  <cfdump var="#test#">
 <cfcatch>
  <cfdump var="#cfcatch#">
 </cfcatch>
</cftry>
```

storeGetACL

Description
Gets the ACL object or bucket.

Returns
Returns an ACLObject

Syntax
`StoreGetACL(url, ACLObject)`

Parameters

Parameter	Description
`url`	Amazon S3 URLs (content or object)
`ACLObject`	An array of struct where each struct represents a permission or grant as discussed in ACLObject.

History
ColdFusion 9 Update 1: Added this function

Example

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cfset dir = "s3://bucket_Name">
 <cfif NOT DirectoryExists(dir)>
   <cfset directoryCreate(dir)>
        </cfif>
 <cfset test = StoreGetACL("#dir#")>
 <cfdump var="#test#">
```

Using metadata

Amazon S3 allows you to specify metadata for both objects and buckets.
The following two functions let you get and set the metadata on objects or buckets.

StoreGetMetadata

Description
Returns the metadata related to the object or bucket.

Returns
Object metadata or bucket metadata

Syntax
`StoreGetMetadata(url)`

Parameters

Parameter	Description
`url`	Amazon S3 URLs (bucket or object).

History
ColdFusion 9 Update 1: Added this function

Example

```
<cfdump var = #StoreGetMetadata("bucket_Name")#>
```

StoreSetMetadata

Description
Sets the metadata on bucket or object.

Returns
Nothing

Syntax
`StoreSetMetadata(url,Struct)`

Parameters

© 2014 Adobe Systems Incorporated. All rights reserved.

Parameter	Description
`url`	Amazon S3 URLs (bucket or object).
`struct`	Represents the metadata. See Standard keys (in this page) for a list of standard keys in metadata.You can also have custom metadata apart from the standard ones.

History
ColdFusion 9 Update 1: Added this function

Example

```
<cfscript>
      mydate = #Now()#;
      hello = structNew();
      hello.color = "grey";
/cfscript>

<cfset dir = "s3://mycfbucket">
      <cffile action="write" file="#dir#/hello5.txt" output="Sample s3 text">

      <cfset StoreSetMetadata("#dir#/hello5.txt","#hello#")>
      <cfset test = StoreGetMetadata("#dir#/hello5.txt")>

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

Standard keys

The following are the standard keys in the metadata:

For objects

- last_modified
- date
- owner
- etag
- content_length
- content_type
- content_encoding
- content_disposition
- content_language
- content_md5
- md5_hash

For buckets

- date
- owner

Security considerations

© 2014 Adobe Systems Incorporated. All rights reserved.

Sandboxing is not applicable to S3 buckets or objects as Amazon S3 has its own security features that take care of it.

Supported functions

`fileOpen`	`fileClose`	`fileCopy`	`fileDelete`
`fileExists`	`fileisEOF`	`fileMove`	`fileWrite`
`fileRead`	`fileReadBinary`	`fileReadLine`	`fileSetLastModified`
`getFileInfo`	`getDirectoryFromPath`	`directoryCreate`	`directoryDelete`
`directoryExists`	`directoryList`	`imageNew`	`imageRead`
`imageWrite`	`imageWriteBase64`	`isImageFile`	`isPDFFile`

Supported tags

All `cffile` actions	All `cfdirectory` actions (except rename)	`cfdocument`	`cffeed`
`cfftp`	`cfimage`	`cfloop`	All `cfimage` actions

Enhancements in ColdFusion 10

Apart from performance improvements while uploading files to Amazon S3, ColdFusion 10 supports multipart upload where files are split into multiple parts and the parts are uploaded in parallel.
To configure multipart upload support, specify the following settings in the Application.cfc:
`this.s3.minsizeformultipart=filesize _in_MB`
The size you specify is treated as minimum, above which file upload is performed as multipart upload. This option is helpful when you have to upload files of huge sizes.

Limitations

- The following tags are not supported:
- `cfpdf`
- `cfpdfform`
- The following functions are not supported:
- `FileSetAccessMode` that sets attributes of a file in Linux/UNIX
- `FilesSetAttribute` that sets the attributes of a file in Windows
- `cfzip` does not accept Amazon S3 object as source.
- When S3 object is used as output for `outputfile` attribute of `cfexecute` tag, it results in an error `Timeout period expired without completion of <exe>`. It also results in a NullPointerException at server console.
- To use the function `fileMove`, the source and destination objects must have the same bucket name. That is, you cannot move Amazon S3 objects across buckets or to other file systems.

Working with in-memory files

© 2014 Adobe Systems Incorporated. All rights reserved.

Memory-based virtual file system speeds up the processing of transient data. In-memory files are not written to disk and are saved on RAM. They function similar to disk files but perform faster.
In ColdFusion, in-memory files help you to simplify the execution of dynamic code. In-memory files are supported across almost all tags and functions that take file or directory as input or output.
You use in-memory files in the same manner as files on disk, but with a prefix `ram:///` to indicate that they reside on RAM. For example, `ram:///a/b/dynamic.cfm.`

Writing and executing dynamic CFM files

The following syntax explains how to write CFM data in to an in-memory file:

```
<cffile action="write" output="#cfml#"
file="ram:///filename.cfm"/>
```

The following sample syntax explains how to use the in-memory CFM file:

- For tags that take logical path, define mapping in Administrator. Execute in-memory CFM pages using the `cf includetag`:

  ```
  <cfinclude template="/inmemory/filename.cfm">
  ```

  Create a mapping for `ram:///` so that it can be used in the tags. In this example, `/inmemory` is the mapping that points to `ram:///`.

- For tags that take absolute path, specify the syntax as provided in the following example:

  ```
  <cffile action="append" file="ram:///a/b/dynamic.cfm" output="I'm appending">
  ```

Note

You cannot have Application.cfm as an in-memory file. If you have one, it is ignored.

Example

The following code describes how to write an image as an in-memory file:

```
<cffile action="readBinary" variable="myImage" file="#ExpandPath('./')#/blue.jpg">
<cffile action="write" output="#myImage#" file="ram:///a.jpg">
<cfif FileExists("ram:///a.jpg")>
<cfoutput>a.jpg exists</cfoutput>
<cfelse>
<cfoutput>a.jpg Doesn't exists</cfoutput>
</cfif>
```

Writing and instantiating dynamic CFC files

© 2014 Adobe Systems Incorporated. All rights reserved.

The following syntax explains how you can write CFC code in to an in-memory file:

```
<cffile action="write" output="#cfcData#"
file="ram:///filename.cfc"/>
```

The following sample syntax explains how you can instantiate the in-memory CFC file:

```
<cfset cfc=CreateObject("component","inmemory.filename")/>
```

Here, `inmemory` is the ColdFusion mapping that points to `ram:///`.

Note

You cannot have Application.cfc as an in-memory file. If you have one, it is ignored.

Example

The following code writes a CFC as in-memory file:

```
<cffile action="read" file="#ExpandPath('./')#/dynamic.cfc" variable="Message">
<cffile action="write" file="ram:///cfc/dynamic.cfc" output="#Message#">
```

To invoke a component method:

```
<cfinvoke component="inmemory.cfc.dynamic" method ="method1"
returnVariable="returnVariable">
<cfinvokeargument name="paramOne" value="hello">
</cfinvoke>
<cfoutput>#returnVariable#</cfoutput>
```

Working with in-memory file system

The following sections provide information that can help you to access and use in-memory files.

Using in-memory files

- You can call a CFC saved on RAM from a CFM file on disk. Similarly, an in-memory CFM file can call a CFC saved on disk.
- If a CFC extends another CFC in the same directory in RAM, you can use relative path. For instance, if a.cfc and b.cfc belong to the same directory in RAM, a.cfc can extend b.cfc using relative path as shown in the following code:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cfcomponent name="a" extends"b">

</cfcomponent>
```

- You can use in-memory ColdFusion interfaces in the same way as you use in-memory CFCs.

Supported functions

The following file functions are supported for in-memory files:

- FileIsEOF
- FileReadBinary
- Filemove
- Filecopy
- FileReadLine
- FileExists
- FileOpen
- FileWriteln
- FileClose
- FileRead
- FileDelete
- DirectoryExists
- FileSetLastModified
- GetFileInfo
- GetDirectoryFromPath
- GetFileFromPath
- ImageNew
- ImageRead
- ImageWrite
- ImageWriteBase64
- IsImageFile
- IsPDFFile
- FileSetLastModified

Example
The following syntax explains the function `FileSetLastModified()`

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cftry>
<cffile action="write" file="ram:///a.txt" output="Testing the function
FileSetLastModified">
<cfset date="12/12/2007">
<cfscript>
FileSetLastModified("ram:///a.txt", "#date#");
sleep(1000);
WriteOutput(#GetFileInfo("ram:///a.txt").lastmodified#);
</cfscript>
<cfcatch>
<cfset PrintException(cfcatch)>
</cfcatch>
</cftry>
<cf_expectedresults>{ts '2007-12-12 00:00:00'}
</cf_expectedresults>
```

File operations

The following file operations are supported for in-memory files:

- Directory-specific operations: create, delete, list, and rename.
- File-specific operations: copy, create, write, append, delete, rename, create attributes, modes move, and read.

Example
The following code illustrates the file and directory operations:

```
<cfdirectory action = "create" directory = "ram://src" >
<cfdirectory action = "create" directory = "ram://des" >
<cfdirectory action = "rename" directory = "ram:///CurrentDir" newDirectory =
"NewDir">
<cfdirectory action="list" directory="ram://" name="listDir" recurse="yes" >
<cfdump var="#listDir#">
<cffile action="write" file = "ram://src/test.txt" output = "Release Description">
<cffile action="copy" source="ram://src/test.txt" destination="ram://des/final.txt"
>
<cffile action="rename" source = "ram:///src/message.txt" destination =
"ram:///des/test.txt">
<cffile action ="move" source = "ram:///des/test.txt" destination =
"c:\des\move.txt">
```

Document and image actions

All image and document actions can use in-memory image files as shown in the following examples:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cfimage action="captcha" fontSize="15" width="180" height="50" text="readMe"
destination="ram:///readMe.jpg"
difficulty="medium">
<cfimage source="ram://aiden02.png" action="convert"
destination="#ExpandPath("./")#/blue1.JPG" overwrite="yes">
<cfdocument format="pdf" filename="ram://Sample.pdf" overwrite="yes">Sample
Text</cfdocument>
```

Custom tags

In-memory CFM pages and CFCs can call custom tags but the custom tags must be present in disk. In-memory custom tags are not supported.

Using in-memory files in tags

The following tags are supported for in-memory files:

- cfcontent
- cfdocument
- cfdump
- cfexchange
- cfexecute
- cffeed
- cfhttp
- cfftp
- cfimage
- cfloop
- cfpresentation
- cfprint
- cfreport
- cfzip

Example using the tag cfcontent

```
<cfcontent file="ram:///a.jpg" type="image/jpeg" deletefile="yes">
```

Adding permissions

ColdFusion lets you add permissions for directories/files on RAM using an existing sandbox security setup. You can only set up sandbox security for disk directories and not for RAM directories. Access to an in-memory directory/file can be restricted only through an existing local file system sandbox.
Therefore, to set up sandbox security for in-memory files, select a sandbox that you have already configured for a disk directory.
By default the `ram:///` directories are included in the list of secured folders and have read, write, execute, and delete permissions set. Therefore, all RAM files have permissions by default in a sandbox. All the security restrictions that apply to disk files apply to in-memory files.
To set up Sandbox security for in-memory files,

1. Open the Security > Sandbox Security page in the ColdFusion Administrator.The Sandbox Security

© 2014 Adobe Systems Incorporated. All rights reserved.

Permissions page appears.
2. In the Add Security Sandbox box, specify a disk directory and then click Add.
3. In Files/Directories, specify the in-memory file path. For example, `ram:///a/b` (for directory) or `ram:///a/b/dynamic.cfm` (for file).
4. Select the required permissions, click Add Files/Paths, and then click Finish.
   For further details on sandbox security, refer to the *ColdFusion Administration Guide.*

Accessing VFS information

The `GetVFSMetaData` function lets you access VFS information. This function takes the parameter RAM as input. This function returns a structure that contains the following information:

- If support for in-memory virtual file system is enabled
- Memory limit in bytes for in-memory virtual file system
- The used and free memory
  For example, `<cfdump var="#getVFSMetaData("ram")#">`

> **Note**
>
> The Settings page of the ColdFusion Administrator has the options to enable/disable the in-memory virtual file system support. You can also specify the memory limit in Megabytes (MB) for in-memory virtual file system.

Deleteing in-memory files

The in-memory files remain in RAM as long as the server is up. When required, clean up the files using cffile/cfdirectory with the `action=delete`. For example, delete all the contents in RAM directory `"ram://a/b"` usi ng the following code: `<cfdirectory action="delete" directory="ram:///a/b" recurse="yes">`

Limitations

- File names/Directory names on RAM are case sensitive.
- In-memory files must be accessed using mapping or absolute path. Relative paths to files/directories are not supported.
  - Correct: `ram:///a/b/`
  - Incorrect: `ram:///a/b/../..`
- You cannot access in-memory files using HTTP/HTTPS protocols. Instead, use `ram:///<file>`. For example, `ram:///a/b/test.cfm`.
- DDX files and font files cannot be accessed as in-memory files.
- The following functions are not supported for in-memory files.
  - `FileSetAccessMode` that sets attributes of a file in Linux/Unix
  - `FilesSetAttribute` that sets the attributes of a file in Windows
- The following tags are not supported:
  - cfpdf
  - cfpdfform
- The following scenarios are not supported:
  - The `cfreport`tag does not accept a template report on RAM. Therefore, the following does not work:

```
<cfreport format="PDF" template="ram:///myReport1.cfr"
filename="ram:///test.pdf" overwrite="yes">
```

© 2014 Adobe Systems Incorporated. All rights reserved.

In this case, the `myReport1.cfr` must reside on your disk.

- The `cfimport`tag does not accept tag libraries on RAM. For instance, the following does not work:

```
<cfimport prefix="custom" taglib="ram:///a/b/mytags.jar">
```

- Renaming across file systems (as shown in the following syntax) is not supported.

```
<cffile action="rename" source="ram:///src1/message2.txt"
destination="#ExpandPath('./')#/123.txt">.
```

Instead, you can move the files.

- For the `cfexecute`tag, the executable cannot be on RAM and must be on your local file system. But the output can be an in-memory file as illustrated in the following code:

```
<cfexecute name="C:\WINDOWS\System32\netstat.exe" arguments="-e"
outputFile="ram:///output.txt" timeout="1">
</cfexecute>
<cfset thisPath=ExpandPath("*.*")>
```

Application-specific In-memory file system

ColdFusion 9 supports in-memory file system only at server level. But the enhancements in this release let you use in-memory file system specific to applications. This enables application isolation for your virtual file system. That is, the file created in the in-memory file system by one application is not accessible to another application.
The settings can be specified in the Application.cfc as follows:

Variable	Description
`this.inmemoryfilesystem.enabled`	Set the value to `true` to enable in-memory file system for application. This is the default setting.
`this.inmemoryfilesystem.size`	Specify the memory limit in MB for the in-memory file system.You can also specify the value in the ColdFusion Administrator (Server Settings > Settings > Memory Limit per Application for In-Memory Virtual File System).The lesser value is considered.

Virtual File System: Support for HTTP, FTP, and ZIP

ColdFusion 10 lets you access files over network through FTP or HTTP protocol. Also provided is support for ZIP. For conventions, see the Apache Commons Documentation.

Examples

For FTP:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cffile
    action = "read"
    file = "ftp://Administrator:Password@Host_Name/ReadmeLater.htm"    variable =
"varvar">
<cffile action="write" file="ftp://Administrator:Password@Host_Name/ReadmeLater.htm"
output="new stuff added">
```

For ZIP

```
<cffile
    action = "read"
    file="zip:#ExpandPath('./')#/hello.zip!/hello.txt"
    variable = "varRead1">
```

Optimizing database use

Poor database design and incorrect or inefficient use of the database are among the most common causes of inefficient applications. Consider the different methods that are available for using databases and information from databases when you design your application. For example, to average the price of many products from a SQL query, it is more efficient to use SQL to get the average than to use a loop in ColdFusion.
Two important ColdFusion tools for optimizing your use of databases are the `cfstoredproc` tag and the `cfquery` tag `cachedWithin` attribute.

Using stored procedures

The `cfstoredproc` tag lets ColdFusion use stored procedures in your database management system. A stored procedure is a sequence of SQL statements that is assigned a name, compiled, and stored in the database system. Stored procedures encapsulate programming logic in SQL statements, and database systems are optimized to execute stored procedures efficiently. As a result, stored procedures are faster than `cfquery` tags.
You use the `cfprocparam` tag to send parameters to the stored procedure, and the `cfprocresult` tag to get the record sets that the stored procedure returns.
The following example executes a Sybase stored procedure that returns three result sets, two of which the example uses. The stored procedure returns the status code and one output parameter, which the example displays.

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<!--- cfstoredproc tag --->
<cfstoredproc procedure = "foo_proc" dataSource = "MY_SYBASE_TEST"
 username = "sa" password = "" returnCode = "Yes">

 <!--- cfprocresult tags --->
 <cfprocresult name = RS1>
 <cfprocresult name = RS3 resultSet = 3>

 <!--- cfprocparam tags --->
 <cfprocparam type = "IN"
  CFSQLType = CF_SQL_INTEGER
   value = "1">
 <cfprocparam type = "OUT" CFSQLType = CF_SQL_DATE
  variable = FOO>
<!--- Close the cfstoredproc tag. --->
</cfstoredproc>

<cfoutput>
 The output param value: '#foo#'<br>
</cfoutput>

<h3>The Results Information</h3>
<cfoutput query = RS1>
 #name#,#DATE_COL#<br>
</cfoutput>
<br>
<cfoutput>
 <hr>
 Record Count: #RS1.recordCount#<br>
 Columns: #RS1.columnList#<br>
 <hr>
</cfoutput>

<cfoutput query = RS3>
 #col1#,#col2#,#col3#<br>
</cfoutput>
<br>
<cfoutput>
 <hr><br>
 Record Count: #RS3.recordCount#<br>
 Columns: #RS3.columnList#<br>
 <hr>

 The return code for the stored procedure is: '#cfstoredproc.statusCode#'<br>
</cfoutput>
```

Reviewing the code

The following table describes the code and its function:

Code	Description

© 2014 Adobe Systems Incorporated. All rights reserved.

`<cfstoredproc procedure = "foo_proc" dataSource = "MY_SYBASE_TEST" username = "sa" password = "" returnCode = "Yes">`	Runs the stored procedure foo_proc on the MY_SYBASE_TEST data source. Populates the `cfstoredproc statusCode` variable with the status code returned by stored procedure.
`<cfprocresult name = RS1>` `<cfprocresult name = RS3 resultSet = 3>`	Gets two record sets from the stored procedure: the first and third result sets it returns.

`<cfprocparam type = "IN" CFSQLType = CF_SQL_INTEGER value = "1">`
`<cfprocparam type = "OUT" CFSQLType = CF_SQL_DATE variable = FOO>`
`<!--- Close the cfstoredproc tag. --->`
`</cfstoredproc>`
	Specifies two parameters for the stored procedure, an input parameter and an output parameter. Sets the input parameter to 1 and the ColdFusion variable that gets the output to FOO.Ends the `cfstoredproc` tag body.

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cfoutput>
 The output param value:
'#foo#'<br>
</cfoutput>

<h3>The Results Information</h3>
<cfoutput query = RS1>
 #name#,#DATE_COL#<br>
</cfoutput>
<br>
<cfoutput>
 <hr>
 Record Count:
#RS1.recordCount#<br>
 Columns: #RS1.columnList#<br>
 <hr>
</cfoutput>

<cfoutput query = RS3>
 #col1#,#col2#,#col3#<br>
</cfoutput>
<br>
<cfoutput>
 <hr><br>
 Record Count:
#RS3.recordCount#<br>
 Columns: #RS3.columnList#<br>
 <hr>

 The return code for the stored
procedure is:
'#cfstoredproc.statusCode#'<br>
</cfoutput>
```

Displays the results of running the stored procedure:

- The output parameter value,
- The contents of the two columns in the first record set identified by the name and DATE_COL variables. You set the values of these variables elsewhere on the page.
- The number of rows and the names of the columns in the first record set
- The contents of the columns in the other record set identified by the col1, col2, and col3 variables.
- The number of rows and the names of the columns in the record set.
- The status value returned by the stored procedure.

For more information on creating stored procedures, see your database management software documentation. For more information on using the `cfstoredproc` tag, see the *CFML Reference*.

Using the cfquery tag cachedWithin attribute

The `cfquery` tag `cachedWithin` attribute tells ColdFusion to save the results of a database query for a specific period of time. This way, ColdFusion accesses the database on the first page request, and does not query the database on further requests until the specified time expires. Using the `cachedWithin` attribute significantly limits the overhead of accessing databases that do not change rapidly.
This technique is useful if the database contents only change at specific, known times, or if the database does not change frequently and the purpose of the query does not require up- to-date results.
Use the `CreateTimeSpan` function to specify the `cachedWithin` attribute value (in days, hours, minutes, seconds format). For example, the following code caches the results of getting the contents of the Employees table of the cfdocexamples data source for one hour.

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cfquery datasource="cfdocexamples" name="master"
 cachedWithin="#CreateTimeSpan(0,1,0,0)#">
 SELECT * FROM Employees
</cfquery>
```

Optimizing transient files

Providing visual feedback to the user

If an application takes a while to process data, it is useful to provide visual feedback to indicate that something is happening, so the user does not assume that there is a problem and requests the page again. Although doing this does not optimize your application's processing efficiency, it does make the application appear more responsive. Use the `cfflush` tag to return partial data to a user, as shown in Introduction to Retrieving and Formatting Data. You can also use the `cfflush` tag to create a progress bar.

#back to top

© 2014 Adobe Systems Incorporated. All rights reserved.

Handling Errors

Adobe ColdFusion includes many tools and techniques for responding to errors that your application encounters. These tools include error handling mechanisms and error logging tools.
For information on user input validation, see Introduction to Retrieving and Formatting Data and Building Dynamic Forms with cfform Tags For information on debugging, see Debugging and Troubleshooting Applications.

© 2014 Adobe Systems Incorporated. All rights reserved.

About error handling in ColdFusion

By default, ColdFusion generates its own error messages when it encounters errors. In addition, it provides a variety of tools and techniques for you to customize error information and handle errors when they occur. You can use any of the following error-management techniques.

- Specify custom pages for ColdFusion to display in each of the following cases:
  - When a ColdFusion page is missing (the Missing Template Handler page)
  - When an otherwise-unhandled exception error occurs during the processing of a page (the Site-wide Error Handler page)
    You specify these pages on the Settings page in the Server Settings page in the ColdFusion Administrator; for more information, see the ColdFusion Administrator Help.
- Use the cferror tag to specify ColdFusion pages to handle specific types of errors.
- Use the `cftry`, `cfcatch`, `cfthrow`, and `cfrethrow` tags to catch and handle exception errors directly on the page where they occur.
- In CFScript, use the `try` and `catch` statements to handle exceptions.
- Use the onError event in Application.cfc to handle exception errors that are not handled by try/catch code on the application pages.
- Log errors. ColdFusion logs certain errors by default. You can use the `cflog` tag to log other errors.
  The following information is detailed in the next few topics:

- The basic building blocks for understating types of ColdFusion errors and how ColdFusion handles them
- How to use the `cferror` tag to specify error-handling pages
- How to log errors
- How to handle ColdFusion exceptions

> **Note**
>
> Information about using the `cftry` and `cfcatch` tags is provided, but not the equivalent CFScript `try` and `catch` statements. The general discussion of exception handling applies to tags and CFScript statements. However, the code that you use and the information available in CFScript differs from the information in the tags. For more information on handling exceptions in CFScript, see Handling errors in UDFs.

© 2014 Adobe Systems Incorporated. All rights reserved.

Understanding errors

You can look at errors in many ways; for example, you can look at errors by their causes. You can also look at them by their effects, particularly by whether your application can recover from them. You can also look at them the way ColdFusion does, as follows:

About error causes and recovery

Errors can have many causes. Depending on the cause, the error can be *recoverable*. A recoverable error is one for which your application can identify the error cause and take action on the problem. Some errors, such as time-out errors, are recoverable without indicating to the user that an error was encountered. An error for which a requested application page does not exist is not recoverable, and the application can only display an error message.
Errors such as validation errors, for which the application cannot continue processing the request, but can provide an error-specific response, can also be considered recoverable. For example, an error that occurs when a user enters text where a number is required can be considered recoverable, because the application can recognize the error and redisplay the data field with a message providing information about the cause of the error and telling the user to reenter the data.
Some types of errors are recoverable in some, but not all circumstances. For example, your application can retry a request following a time-out error, but it must also be prepared for the case where the request always times out.
Error causes fall in the broad categories listed in the following table:

Category	Description
Program errors	Can be in the code syntax or the program logic. The ColdFusion compiler identifies and reports program syntax errors when it compiles CFML into Java classes. Errors in your application logic are harder to locate. For information on debugging tools and techniques, see Debugging and Troubleshooting Applications.Unlike ColdFusion syntax errors, SQL syntax errors are only caught at runtime.
Data errors	Are typically user data input errors. You use validation techniques to identify errors in user input data and enable the user to correct the errors.
System errors	Can come from a variety of causes, including database system problems, time-outs due to excessive demands on your server, out-of-memory errors in the system, file errors, and disk errors.

Although these categories do not map completely to the way ColdFusion categorizes errors they provide a useful way of thinking about errors and can help you in preventing and handling errors in your code.

ColdFusion error types

Before you can effectively manage ColdFusion errors, you must understand how ColdFusion classifies and handles them. ColdFusion categorizes errors as detailed in the following table:

Type	Description

© 2014 Adobe Systems Incorporated. All rights reserved.

Exception	An error that prevents normal processing from continuing. All ColdFusion exceptions are, at their root, Java exceptions.
Missing template	An HTTP request for a ColdFusion page that cannot be found. Generated if a browser requests a ColdFusion page that does not exist.Missing template errors are different from missing include exceptions, which result from `cfinclude` tags or custom tag calls that cannot find their targets.
Form field data validation	Server-side form field validation errors are a special kind of ColdFusion exception. You specify server-side form validation by using `cfform` attributes or hidden HTML form fields. All other types of server-side validation, such as the cfparam tag generate runtime exceptions. For more information on validating form fields see Validating data.ColdFusion includes a built-in error page for server-side form field validation errors, and the cferror tag includes a `type` attribute that lets you handle these errors in a custom error page, but if you use onError processing in Application.cfc, or try/catch error handling, the error appears as an Application exception. For more information on handling Form field validation in Application.cfc see *Handling server-side validation errors in the on Error method* in Defining the application and its event handlers in Application.cfc.

Note

The onSubmit and onBlur form field validation techniques use JavaScript or Flash validation on the client browser.

About ColdFusion exceptions

Most ColdFusion errors are exceptions. You can categorize ColdFusion exceptions in two ways:

- When they occur
- Their type

When exceptions occur

ColdFusion errors can occur at two times, when the CFML is compiled into Java and when the resulting Java executes, called runtime exceptions.

Compiler exceptions

Compiler exceptions are programming errors that ColdFusion identifies when it compiles CFML into Java. Because compiler exceptions occur before the ColdFusion page is converted to executable code, you cannot handle them on the page that causes them. However, other pages can handle these errors. For more information, see *Handling compiler exceptions* in Determining error-handling strategies.

© 2014 Adobe Systems Incorporated. All rights reserved.

Runtime exception

A runtime exception occurs when the compiled ColdFusion Java code runs. It is an event that disrupts the normal flow of instructions in the application. Exceptions can result from system errors or program logic errors. Runtime exceptions include:

- Error responses from external services, such as an ODBC driver or CORBA server
- CFML errors or the results of `cfthrow` or `cfabort` tags
- Internal errors in ColdFusion

ColdFusion exception types

ColdFusion exceptions have types that you specify in the `cferror`, `cfcatch`, and `cfthrow` error-handling tags. A `cferror` or `cfcatch` tag handles only exceptions of the specified type. You identify an exception type by using an identifier from one (or more) of the following type categories:

- Basic
- Custom
- Advanced
- Java class

> **Note**
>
> Use only custom error type names and the `Application` basic type name in `cfthrow` tags. All other built-in exception type names identify specific types of system-identified errors, so do not use them for errors that you identify yourself.

Basic exception types

All ColdFusion exceptions except for custom exceptions belong to a basic type category. These types consist of a broadly defined categorization of ColdFusion exceptions. The following table describes the basic exception types:

Type	Type name	Description
Database failures	Database	Failed database operations, such as failed SQL statements, ODBC problems, and so on.
Missing include file errors	MissingInclude	Errors where files specified by the cfinclude, cfmodule, and cferror tags are missing. (A `cferror` tag generates a missingInclude error only when an error of the type specified in the tag occurs.)The `MissingInclude` error type is a subcategory of Template error. If you do not specifically handle the MissingInclude error type, but do handle the Template error type, the Template error handler catches these errors. `MissingInclude` errors are caught at runtime.

© 2014 Adobe Systems Incorporated. All rights reserved.

Template errors	Template	General application page errors, including invalid tag and attribute names. Most `Template` errors are caught at compile time, not runtime.
Object exceptions	Object	Exceptions in ColdFusion code that works with objects.
Security exceptions	Security	Catchable exceptions in ColdFusion code that works with security.
Expression exceptions	Expression	Failed expression evaluations; for example, if you try to add 1 and "a".
Locking exceptions	Lock	Failed locking operations, such as when a cflock critical section times out or fails at runtime.
Application-defined exception events raised by `cfthrow`	Application	Custom exceptions generated by a cfthrow tag that do not specify a type, or specify the type as `Application`.
All exceptions	Any	Any exceptions. Includes all types in this table and any exceptions that are not handled in another error handler, including unexpected internal and external errors.

Note

The Any type includes all error with the Java object type of java.lang.Exception. It does not include java.lang.Throwable errors. To catch Throwable errors, specify java.lang.Throwable in the `cfcatch` tag `type` attribute.

Custom exceptions

You can generate an exception with your own type by specifying a custom exception type name, for example MyCustomErrorType, in a `cfthrow` tag. You then specify the custom type name in a `cfcatch` or `cferror` tag to handle the exception. Custom type names must be different from any built-in type names, including basic types and Java exception classes.

Advanced exception types

The Advanced exceptions consist of a set of specific, narrow exception types. These types are supported in ColdFusion for backward-compatibility.

Java exception classes

Every ColdFusion exception belongs to, and is identified by, a specific Java exception class in addition to its basic, custom, or advanced type. The first line of the stack trace in the standard error output for an exception identifies the

© 2014 Adobe Systems Incorporated. All rights reserved.

Java class of the exception.
For example, if you attempt to use an array function such as `ArrayIsEmpty` on an integer variable, ColdFusion generates an exception that belongs to the `Expression` exception basic type and the `coldfusion.runtime.NonArrayException` Java class.
In general, most applications do not use Java exception classes to identify exceptions. However, you can use Java class names to catch exceptions in non-CFML Java objects; for example, the following line catches all Java input/output exceptions:

```
<cfcatch type="java.io.IOException">
```

How ColdFusion handles errors

The following information describes briefly how ColdFusion handles errors. Detailed information is provided in the remaining topics.

Missing template errors

If a user requests a page that ColdFusion cannot find, and the Administrator Server Settings Missing Template Handler field specifies a Missing Template Handler page, ColdFusion uses that page to display error information. Otherwise, it displays a standard error message.

Form field validation errors

When a user enters invalid data in an HTML tag that uses onServer or hidden form field server-side data validation ColdFusion does the following:

1. If the Application CFC (Application.cfc) has an `onError` event handler method, ColdFusion calls the method.
2. If the Application.cfc initialization code or the Application.cfm page has a `cferror` that specifies a Validation error handler, ColdFusion displays the specified error page.
3. Otherwise, it displays the error information in a standard format that consists of a default header, a bulleted list describing the errors, and a default footer.
   For more information on using hidden form field validation, see Validating data. For more information on Application.cfc, see Designing and Optimizing a ColdFusion Application.

Compiler exception errors

If ColdFusion encounters a compiler exception, how it handles the exception depends on whether the error occurs on a requested page or on an included page:

- If the error occurs on a page that is accessed by a `cfinclude` or `cfmessageboxcfmodule` tag, or on a custom tag page that you access using the cf_ notation, ColdFusion handles it as a runtime exception in the page that accesses the tag. For a description of how these errors are handled, see the next section, " *Runtime exception errors*."
- If the error occurs directly on the requested page, and the Administrator Settings Site-wide Error Handler field specifies an error handler page, ColdFusion displays the specified error page. Otherwise, ColdFusion reports the error using the standard error message format described in Error messages and the standard error format .

Runtime exception errors

If ColdFusion encounters a runtime exception, it does the action for the *first* matching condition in the following table:

© 2014 Adobe Systems Incorporated. All rights reserved.

Condition	Action
The code with the error is inside a cftry tag and the exception type is specified in a `cfcatch` tag.	Executes the code in the cfcatch tag.If the `cftry` block does not have a `cfcatch` tag for this error, tests for an appropriate cferror handler or site-wide error handler.
The ColdFusion application has an Application.cfc with an onError method	Executes the code in the `onError` method. For more information on using the `onError` method, see *Handling errors in Application.cfc* in Defining the application and its event handlers in Application.cfc.
A `cferror` tag specifies an exception error handler for the exception type.	Uses the error page specified by the `cferror` tag.
The Administrator Settings Site-wide Error Handler field specifies an error handler page.	Uses the custom error page specified by the Administrator setting.
A `cferror` tag specifies a Request error handler.	Uses the error page specified by the `cferror` tag.
The default case.	Uses the standard error message format

For example, if an exception occurs in CFML code that is not in a `cftry` block, and Application.cfc does not have an `onError` method, but a `cferror` tag specifies a page to handle this error type, ColdFusion uses the specified error page.

© 2014 Adobe Systems Incorporated. All rights reserved.

Error messages and the standard error format

If your application does not handle an error, ColdFusion displays a diagnostic message in the user's browser. Error information is also written to a log file for later review. (For information on error logging, see Logging errors with the cflog tag.)
The standard error format consists of the information listed in the following table. ColdFusion does not always display all sections.

Section	Description
Error description	A brief, typically online, description of the error.
Error message	A detailed description of the error. The error message diagnostic information displayed depends on the type of error. For example, if you specify an invalid attribute for a tag, this section includes a list of all valid tag attributes.
Error location	The page and line number where ColdFusion encountered the error, followed by a short section of your CFML that includes the line. This section does not display for all errors. In some cases, the cause of an error can be several lines above the place where ColdFusion determines that there is a problem, so the line that initially causes the error might not be in the display.
Resources	Links to documentation, the Knowledge Base, and other resources that can help you resolve the problem.
Error environment information	Information about the request that caused the error. All error messages include the following: • User browser • User IP address • Date and time of error
Stack trace	The Java stack at the time of the exception, including the specific Java class of the exception. This is helpful if you must contact Adobe Technical Support.The stack trace is collapsed by default. Click the heading to display the trace.

✅ If you get a message that does not explicitly identify the cause of the error, check the key system parameters, such as available memory and disk space.

© 2014 Adobe Systems Incorporated. All rights reserved.

Determining error-handling strategies

ColdFusion provides you with many options for handling errors, particularly exceptions, as described in *How ColdFusion handles errors* in Understanding errors. The considerations for determining which forms of error handling to use are as follows:

Handling missing template errors

Missing template errors occur when ColdFusion receives an HTTP request for a page ending in .cfm that it cannot find. You can create your own missing template error page to present application-specific information or provide an application-specific appearance. You specify the missing template error page on the Administrator Settings page.

The missing error page can use CFML tags and variables. In particular, you can use the CGI.script_name variable in text such as the following to identify the requested page:

```
<cfoutput>The page #Replace(CGI.script_name, "/", "")# is not available.<br>
Make sure that you entered the page correctly.<br>
</cfoutput>
```

(In this code, the `Replace` function removes the leading slash sign from the script name to make the display more friendly.)

Handling form field validation errors

When you use server-side form field validation, the default validation error message describes the error cause plainly and clearly. However, to give the error message a custom look or provide additional information such as service contact phone numbers and addresses, use the `cferror` tag with the `Validation` attribute in the Application.cfc initialization code. Or, specify your own validation on the Application.cfm page. An example of such a page is provided at *Example of a validation error page* section. You can also place form field validation error handling code in the Application.cfc `onError` method.

Handling compiler exceptions

You cannot handle compiler exceptions directly on the page where they occur, because the exception is caught before ColdFusion starts running the page code. Fix all compiler exceptions as part of the development process. Use the reported error message and the code debugging techniques discussed in Debugging and Troubleshooting Applications to identify and correct the cause of the error.

Compiler exceptions that occur on pages you access by using the `cfinclude` or cfmodule tags can be handled as runtime errors by surrounding the `cfinclude` or `cfmodule` tag in a `cftry` block. The compiler exception on the accessed page gets caught as a runtime error on the base page. However, avoid this "solution" to the problem, as the correct method for handling compiler errors is to remove them before you deploy the application.

Handling runtime exceptions

You have many choices for handling exceptions, and the exact path you take depends on your application and its needs. The following table provides a guide to selecting an appropriate technique:

Technique	Use

© 2014 Adobe Systems Incorporated. All rights reserved.

cftry	Place `cftry` blocks around specific code sections where exceptions can be expected and you want to handle those exceptions in a context-specific manner; for example, if you want to display an error message that is specific to that code. Use `cftry` blocks where you can recover from an exception. For example, you can retry an operation that times out, or access an alternate resource. You can also use the `cftry` tag to continue processing where a specific exception does not harm your application; for example, if a missing resource is not required. For more information, see Handling runtime exceptions with ColdFusion tags.
Application.cfc onError method	Implement the `onError` method in your Application.cfc to consistently handle application-specific exceptions that are generated by multiple code sections in the application. For more information on error handling using Application.cfc, see *Handling errors in Application.cfc* in Understanding errors.
cferror `with` exception- specific error handler pages	Use the `cferror` tag to specify error pages for specific exception types. These pages cannot recover from errors, but they can provide the user with information about the cause of the error and the steps to take to prevent the problem.For more information, see Specifying custom error messages with the cferror tag.
`cferror` with a Request error page	Use the `cferror` tag to specify a Request error handler that provides a customized, application-specific message for unrecoverable exceptions. Place the tag in the Application.cfc initialization code or on the Application.cfm page to make it apply to all pages in an application.A Request error page cannot use CFML tags, but it can display error variables. As a result, you can use it to display common error information, but you cannot provide error-specific instructions. Typically, Request pages display error variable values and application-specific information, including support contact information.For example code, see *Example of a request error page* section.

© 2014 Adobe Systems Incorporated. All rights reserved.

Site-wide error handler page	Specify a site-wide error handler in the Administrator to provide consistent appearance and contents for all otherwise-unhandled exceptions in all applications on your server. Like the Request page, the site-wide error handler cannot perform error recovery. However, it can include CFML tags in addition to the error variables. Because a site-wide error handler prevents ColdFusion from displaying the default error message, it allows you to limit the information reported to users. It also lets you provide all users with default contact information or other instructions.

© 2014 Adobe Systems Incorporated. All rights reserved.

Specifying custom error messages with the cferror tag

Custom error pages let you control the error information that users see. You can specify custom error pages for different types of errors and handle different types of errors in different ways. For example, you can create specific pages to handle errors that could be recoverable, such as request time-outs. You can also make your error messages consistent with the look and feel of your application.
You can specify the following types of custom error message pages:

Type	Description
Validation	Handles server-side form field data validation errors. The validation error page cannot include CFML tags, but it can display error page variables.You can use this attribute only in the Application.cfc initialization code or on the Application.cfm page. It has no effect when used on any other page. Therefore, you can specify only one validation error page per application, and that page applies to all server-side validation errors.
Exception	Handles specific exception errors. You can specify individual error pages for different types of exceptions.
Request	Handles any exception that is not otherwise-handled. The request error page runs after the CFML language processor finishes. As a result, the request error page cannot include CFML tags, but can display error page variables. A request error page is useful as a backup if errors occur in other error handlers.

Specifying a custom error page

You specify the custom error pages with the `cferror` tag. For Validation errors, the tag must be in the Application.cfc initialization code or on the Application.cfm page. For Exception and Request errors, you can set the custom error pages on each application page. However, because custom error pages generally apply to an entire application, it is more efficient to place these `cferror` tags in the Application.cfc or Application.cfm file also. For more information on using these pages, see Designing and Optimizing a ColdFusion Application
The `cferror` tag has the attributes listed in the following table:

Attribute	Description
Type	The type of error that causes ColdFusion to display this page: Exception, Request, or Validation.
Exception	Use only for the Exception type. The specific exception or exception category that causes the page to display. This attribute can specify any of the types described in *About ColdFusion exceptions* in Understanding errors.
Template	The ColdFusion page to display.

© 2014 Adobe Systems Incorporated. All rights reserved.

MailTo	(Optional) An e-mail address. The `cferror` tag sets the error page `error.mailTo` variable to this value. The error page can use the `error.mailTo` value in a message that tells the user to send an error notification. ColdFusion does not send any message itself.

The following `cferror` tag specifies a custom error page for exceptions that occur in locking code and informs the error page of the e-mail address to use to send a notification each time this type of error occurs:

```
<cferror type = "exception"
 exception = "lock"
 template = "../common/lockexcept.cfm"
 mailto = "server@mycompany.com">
```

For detailed information on the `cferror` tag, see the *CFML Reference*.

Creating an error application page

The following table lists the rules and considerations that apply to error application pages:

Type	Considerations
Validation	• Cannot use CFML tags. • Can use HTML tags. • Can use the `Error.InvalidFields`, `Error.validationHeader`, and `Error.validationFooter` variables by enclosing them with number signs (#). • Cannot use any other CFML variables.
Request	• Cannot use CFML tags. • Can use HTML tags. • Can use nine CFML error variables, such as `Error.Diagnostics`, by enclosing them with number signs. • Cannot use other CFML variables.
Exception	• Can use full CFML syntax, including tags, functions, and variables. • Can use nine standard CFML Error variables and cfcatch variables. Use either `Error` or `cferror` as the prefix for both types of variables. • Can use other application-defined CFML variables. • To display any CFML variable, use the cfoutput tag.

The following table describes the variables available on error pages: Exception error pages can also use all of the exception variables listed in the section *Exception information in cfcatch blocks* in Handling runtime exceptions with ColdFusion tags. To use these variables, replace the `cfcatch` prefix with `cferror` or `error`. For example, to use the exception message in an error page, refer to it as `error.message`.

© 2014 Adobe Systems Incorporated. All rights reserved.

In general, production Exception and Request pages should not display detailed error information, such as that supplied by the `error.diagnostics` variable. Typically, Exception pages e-mail detailed error information to an administrative address or log the information using the `cflog` tag instead of displaying it to the user. For more information on using the `cflog` tag, see Logging errors with the cflog tag.

Example of a request error page

The following example shows a custom error page for a request error:

```
<html>
<head>
<title>Products - Error</title>
</head>
<body>

<h2>Sorry</h2>

<p>An error occurred when you requested this page.</p>
<p>Please send e-mail with the following information to #error.mailTo# to report
 this error.</p>

<table border=1>
<tr><td><b>Error Information</b> <br>
 Date and time: #error.DateTime# <br>
 Page: #error.template# <br>
 Remote Address: #error.remoteAddress# <br>
 HTTP Referer: #error.HTTPReferer#<br>
</td></tr></table>

<p>We apologize for the inconvenience and will work to correct the problem.</p>
</body>
</html>
```

Example of a validation error page

The following example shows a simple custom error page for a validation error:

```
<html>
<head>
<title>Products - Error</title>
</head>
<body>

<h2>Data Entry Error</h2>

<p>You failed to correctly complete all the fields
in the form. The following problems occurred:</p>

#error.invalidFields#

</body>
</html>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

Logging errors with the cflog tag

ColdFusion provides extensive capabilities for generating, managing, and viewing log files, as described in *Configuring and Administering ColdFusion*. It also provides the `cflog` tag which adds entries to ColdFusion logs.
ColdFusion automatically logs errors to the default logs if you use the default error handlers. In all other cases, use the `cflog` tag in your error handling code to generate log entries.
The `cflog` tag lets you specify the following information:

- A custom file or standard ColdFusion log file in which to write the message.
- Text to write to the log file. This can include the values of all available error and `cfcatch` variables.
- Message severity (type): Information, Warning, Fatal, or Error.
- Whether to log any of the following: application name, thread ID, system date, or system time. By default, all get logged.
  For example, you could use a `cflog` tag in an exception error-handling page to log the error information to an application-specific log file, as in the following page:

```
<html>
<head>
<title>Products - Error</title>
</head>
<body>

<h2>Sorry</h2>

<p>An error occurred when you requested this page.
The error has been logged and we will work to correct the problem.
We apologize for the inconvenience. </p>

<cflog type="Error"
  file="myapp_errors"
  text="Exception error --
   Exception type: #error.type#
   Template: #error.template#,
   Remote Address: #error.remoteAddress#,
   HTTP Reference: #error.HTTPReferer#
   Diagnostics: #error.diagnostics#">

</body>
</html>
```

Reviewing the code

The following table describes the highlighted code and its function:

Code	Description

© 2014 Adobe Systems Incorporated. All rights reserved.

`<cflog type="Error"`
` file="myapp_errors"`
` text="Exception error --`
`  Exception type: #error.type#`
`  Template: #error.template#,`
`  Remote Address:`
`#error.remoteAddress#,`
`  HTTP Reference:`
`#error.HTTPReferer#`
`  Diagnostics:`
`#error.diagnostics#">`
	When this page is processed, log an entry in the file myapp_errors.log file in the ColdFusion log directory. Identify the entry as an error message and include an error message that includes the exception type, the path of the page that caused the error, the remote address that called the page, and the error's diagnostic message.

A log file entry like the following is generated if you try to call a nonexistent custom tag and this page catches the error (line breaks added for clarity):

```
"Error","web-13","12/19/01","11:29:07",MYAPP,"Exception error --
  Exception type: coldfusion.runtime.CfErrorWrapper
  Template: /MYStuff/MyDocs/exceptiontest.cfm,
  Remote Address: 127.0.0.1,
  HTTP Reference:
  Diagnostics: Cannot find CFML template for custom tag testCase. Cannot
  find CFML template for custom tag testCase. ColdFusion attempted looking
  in the tree of installed custom tags but did not find a custom tag with
  this name."
```

The text consists of a comma-delimited list of the following entries:

- Log entry type, specified by the `cflog type` attribute
- ID of the thread that was executing
- Date the entry was written to the log
- Time the entry was written to the log
- Application name, as specified in the Application.cfc initialization code (by setting the This.application variable) or by a `cfapplication` tag (for example, in an Application.cfm file).
- The message specified by the `cflog text` attribute.

© 2014 Adobe Systems Incorporated. All rights reserved.

Handling runtime exceptions with ColdFusion tags

Exceptions include any event that disrupts the normal flow of instructions in a ColdFusion page, such as failed database operations, missing include files, or developer-specified events. Ordinarily, when ColdFusion encounters an exception, it stops processing and displays an error message, or an error page specified by a `cferror` tag or the Site-wide Error Handler option on the Settings page in the Administrator. However, you can use the ColdFusion exception handling tags to catch and process runtime exceptions directly in ColdFusion pages.
This ability to handle exceptions directly in your application pages enables your application to do the following:

- Respond appropriately to specific errors within the context of the current application page
- Recover from errors whenever possible.

Exception-handling tags

ColdFusion provides the exception-handling tags listed in the following table:

Tag	Description
cftry	If any exceptions occur while processing the tag body, look for a `cfcatch` tag that handles the exception, and execute the code in the `cfcatch` tag body.
cfcatch	Execute code in the body of this tag if the exception caused by the code in the `cftry` tag body matches the exception type specified in this tag's attributes.Used in `cftry` tag bodies only.
cfthrow	Generate a user-specified exception.
cfrethrow	Exit the current `cfcatch` block and generates a new exception of the same type. Used only in `cfcatch` tag bodies.

Using cftry and cfcatch tags

The `cftry` tag lets you go beyond reporting error data to the user:

- You can include code that recovers from errors so your application can continue processing without alerting the user.
- You can create customized error messages that apply to the specific code that causes the error.
  For example, you can use `cftry` to catch errors in code that enters data from a user registration form to a database. The `cfcatch` code could do the following:

1. Retry the query, so the operation succeeds if the resource was only temporarily unavailable.
2. If the retries fail:
   - Display a custom message to the user
   - Post the data to an e-mail address so the company staff can enter the data after the problem has been solved.
     Code that accesses external resources such as databases, files, or LDAP servers where resource availability is not guaranteed is a good candidate for using try/catch blocks.

Try/catch code structure

© 2014 Adobe Systems Incorporated. All rights reserved.

In order for your code to directly handle an exception, the tags in question must appear within a `cftry` block. It is a good idea to enclose an entire application page in a `cftry` block. You then follow the `cftry` block with `cfcatch` bl ocks, which respond to potential errors. When an exception occurs within the `cftry` block, processing is thrown to the `cfcatch` block for that type of exception.
Here is an outline for using `cftry` and `cfcatch` to handle errors:

```
<cftry>
 Put your application code here ...
 <cfcatch type="exception type1">
  Add exception processing code here ...
 </cfcatch>
 <cfcatch type="exception type2">
  Add exception processing code here ...
 </cfcatch>
 ...
 <cfcatch type="Any">
  Add exception processing code appropriate for all other exceptions here ...
 </cfcatch>
</cftry>
```

Try/catch code rules and recommendations

Follow these rules and recommendations when you use `cftry` and `cfcatch` tags:

- The `cfcatch` tags must follow all other code in a `cftry` tag body.
- You can nest `cftry`blocks. For example, the following structure is valid:

  ```
  <cftry>
   code that may cause an exception
   <cfcatch ...>
    <cftry>
     First level of exception handling code
     <cfcatch ...>
      Second level of exception handling code
     </cfcatch
    </cftry>
   </cfcatch>
  </cftry>
  ```

  If an exception occurs in the first level of exception-handling code, the inner `cfcatch` block can catch and handle it. (An exception in a `cfcatch` block cannot be handled by `cfcatch` blocks at the same level as that block.)

- ColdFusion always responds to the latest exception that gets raised. For example, if code in a `cftry` block causes an exception that gets handled by a `cfcatch` block, and the `cfcatch` block causes an exception that has no handler, ColdFusion displays the default error message for the exception in the `cfcatch` block, and you are not notified of the original exception.
- If an exception occurs when the current tag is nested inside other tags, the CFML processor checks the entire stack of open tags until it finds a suitable `cftry/cfcatch` combination or reaches the end of the stack.
- Use `cftry` with `cfcatch` to handle exceptions based on their point of origin within an application page, or based on diagnostic information.

© 2014 Adobe Systems Incorporated. All rights reserved.

- The entire `cftry` tag, including all its `cfcatch` tags, must be on a single ColdFusion page. You cannot place the `<cftry>` start tag on one page and have the `</cftry>` end tag on another page.
- For cases when a `cfcatch` block is not able to successfully handle an error, consider using the `cfrethrow` tag, as described in *Using the cfrethrow tag* in Handling runtime exceptions with ColdFusion tags.
- If an exception can be safely ignored, use a `cfcatch`tag with no body; for example:

```
<cfcatch Type = Database />
```

- In problematic cases, enclose an exception-prone tag in a specialized combination of `cftry` and `cfcatch` tags to immediately isolate the tag's exceptions.

Exception information in cfcatch blocks

Within the body of a `cfcatch` tag, the active exception's properties are available in a `cfcatch` object. The object contents are described as follows:

Standard cfcatch variables

The following table describes the variables that are available in most `cfcatch` blocks:

Property variable	Description
`cfcatch.Detail`	A detailed message from the CFML compiler. This message, which can contain HTML formatting, can help to determine which tag threw the exception. The `cfcatch.Detail` value is available in the CFScript `catch` statement as the `exceptionVariable` parameter.
`cfcatch.ErrorCode`	The `cfthrow` tag can supply a value for this code through the `errorCode` attribute. For `Type="Database"`, `cfcatch.ErrorCode` has the same value as `cfcatch.SQLState`. Otherwise, the value of `cfcatch.ErrorCode` is the empty string.
`cfcatch.ExtendedInfo`	Custom error message information. This is returned only to `cfcatch` tags for which the `type` attribute is `Application` or a custom type.Otherwise, the value of `cfcatch.ExtendedInfo` is the empty string.
`cfcatch.Message`	The exception's default diagnostic message, if one was provided. If no diagnostic message is available, this is an empty string.The `cfcatch.Message` value is included in the value of the CFScript `catch` statement `exceptionVariable` parameter.
`cfcatch.RootCause`	The Java servlet exception reported by the JVM as the cause of the "root cause" of the exception.

© 2014 Adobe Systems Incorporated. All rights reserved.

`cfcatch.TagContext`	An array of structures structure containing information for each tag in the tag stack The tag stack consists of each tag that is currently open.
`cfcatch.Type`	The exception's type, returned as a string.

Note

If you use the `cfdump` tag to display the cfcatch variable, the display does not include variables that do not have values.

The `cfcatch.TagContext` variable contains an array of tag information structures. Each structure represents one level of the active tag context at the time when ColdFusion detected the exception. That is, there is one structure for each tag that is open at the time of the exception. For example, if the exception occurs in a tag on a custom tag page, the tag context displays information about the called custom tag and the tag in which the error occurs.
The structure at position 1 in the array represents the currently executing tag at the time the exception was detected. The structure at position `arrayLen` represents the initial tag in the stack of tags that were executing when the compiler detected the exception.
The following table lists the tagContext structure attributes:

Entry	Description
Column	Obsolete (retained for backwards compatibility). Always 0.
ID	The tag in which the exception occurred. Exceptions in CFScript are indicated by two question marks (??). All custom tags, including those called directly, are identified as cfmodule.
Line	The line on the page in which the tag is located.
Raw_Trace	The raw Java stack trace for the error.
Template	The pathname of the application page that contains the tag.
Type	The type of page; it is always a ColdFusion page.

Database exceptions

The following additional variables are available whenever the exception type is database:

Property variable	Description

© 2014 Adobe Systems Incorporated. All rights reserved.

`cfcatch.NativeErrorCode`	The native error code associated with this exception. Database drivers typically provide error codes to assist in the diagnosis of failing database operations. The values assumed by `cfcatch.NativeErrorCode` are driver-dependent.If no error code is provided, the value of `cfcatch.nativeErrorCode` is -1. The value is 0 for queries of queries.
`cfcatch.SQLState`	The SQLState code associated with this exception. Database drivers typically provide error codes to assist in the diagnosis of failing database operations. SQLState codes are more consistent across database systems than native error codes. If the driver does not provide an SQLState value, the value of `cfcatch.SQLState` is -1.
`cfcatch.Sql`	The SQL statement sent to the data source.
`cfcatch.queryError`	The error message as reported by the database driver.
`cfcatch.where`	If the query uses the cfqueryparam tag, query parameter name-value pairs.

Expression exceptions

The following variable is only available for Expression exceptions:

Property variable	Description
`cfcatch.ErrNumber`	An internal expression error number, valid only when `type="Expression"`.

Locking exceptions

The following additional information is available for exceptions related to errors that occur in `cflock` tags:

Property variable	Description
`cfcatch.lockName`	The name of the affected lock. This is set to "anonymous" if the lock name is unknown.
`cfcatch.lockOperation`	The operation that failed. This is set to "unknown" if the failed operation is unknown.

Missing include exceptions

The following additional variable is available if a missing file specified by a `cfinclude` tag causes the error.

Property variable	Description

© 2014 Adobe Systems Incorporated. All rights reserved.

cfcatch.missingFileName	The name of the missing file.

Using the cftry tag: an example

The following example shows the cftry and cfcatch tags. It uses the cfdocexamples data source, which many of the examples listed here use, and a sample included file, includeme.cfm.
If an exception occurs when you run the cfquery statement, the application page flow switches to the cfcatch type="Database" exception handler. It then resumes with the next statement after the cftry block, once the cfcatch type="Database" handler completes. Similarly, the cfcatch type="MissingInclude" block handles exceptions raised by the cfinclude tag.

```
<!--- Wrap code you want to check in a cftry block --->
<cfset EmpID=3>
<cfparam name="errorCaught" default="">
<cftry>
 <cfquery name="test" datasource="cfdocexamples">
  SELECT Dept_ID, FirstName, LastName
  FROM Employee
  WHERE Emp_ID=#EmpID#
 </cfquery>

 <html>
 <head>
 <title>Test cftry/cfcatch</title>
 </head>
 <body>
 <cfinclude template="includeme.cfm">
 <cfoutput query="test">
  <p>Department: #Dept_ID#<br>
  Last Name: #LastName#<br>
  First Name: #FirstName#</p>
 </cfoutput>

<!--- Use cfcatch to test for missing included files. --->
<!---  Print Message and Detail error messages. --->
<!--- Block executes only if a MissingInclude exception is thrown. --->
 <cfcatch type="MissingInclude">
  <h1>Missing Include File</h1>
  <cfoutput>
  <ul>
   <li><b>Message:</b> #cfcatch.Message#
   <li><b>Detail:</b> #cfcatch.Detail#
   <li><b>Filename:</b> #cfcatch.MissingFileName#
  </ul>
  </cfoutput>
  <cfset errorCaught = "MissingInclude">
 </cfcatch>

<!--- Use cfcatch to test for database errors.--->
<!--- Print error messages. --->
<!--- Block executes only if a Database exception is thrown. --->
 <cfcatch type="Database">
  <h1>Database Error</h1>
  <cfoutput>
  <ul>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

```
    <li><b>Message:</b> #cfcatch.Message#
    <li><b>Native error code:</b> #cfcatch.NativeErrorCode#
    <li><b>SQLState:</b> #cfcatch.SQLState#
    <li><b>Detail:</b> #cfcatch.Detail#
    </ul>
   </cfoutput>
   <cfset errorCaught = "Database">
  </cfcatch>

 <!--- Use cfcatch with type="Any" --->
 <!--- to find unexpected exceptions. --->
  <cfcatch type="Any">
   <cfoutput>
    <hr>
    <h1>Other Error: #cfcatch.Type#</h1>
    <ul>
     <li><b>Message:</b> #cfcatch.Message#
     <li><b>Detail:</b> #cfcatch.Detail#
    </ul>
   </cfoutput>
   <cfset errorCaught = "General Exception">
  </cfcatch>
 </body>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

```
</html>
</cftry>
```

Use the following procedure to test the code.

Test the code

1. Make sure that there is no includeme.cfm file and display the page. The `cfcatch type="MissingInclude"` block displays the error.
2. Create a nonempty includeme.cfm file and display the page. If your database is configured properly, you see an employee entry and do not get any error.
3. In the `cfquery`tag, change the line:

   ```
   FROM Employee
   ```

   to:

   ```
   FROM Employer
   ```

   Display the page. This time the `cfcatch type="Database"` block displays an error message.

4. Change Employer to Employee. Change the `cfoutput`line:

   ```
   <p>Department: #Dept_ID#<br>
   ```

   to:

   ```
   <p>Department: #DepartmentID#<br>
   ```

   Display the page. This time the `cfcatch type="Any"` block displays an error message indicating an expression error.

5. Change DepartmentID back to Dept_ID and redisplay the page. The page displays properly.Open \CFusion\Log\MyAppPage.log in your text editor. You should see a header line, an initialization line, and four detail lines, like the following:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
"Severity","ThreadID","Date","Time","Application","Message"
"Information","web-0","11/20/01", "16:27:08",,
"cf_root\runtime\servers\default\logs\ MyAppPage.log initialized"
"Information","web-0","11/20/01","16:27:08",,
 "Page: web_root/MYStuff/MyDocs/ cftryexample.cfm Error: MissingInclude"
"Information","web-1","11/20/01","16:27:32",,"
 Page: web_root/MYStuff/MyDocs/ cftryexample.cfm Error: "
"Information","web-0","11/20/01","16:27:49",,
 "Page: web_root/MYStuff/MyDocs/ cftryexample.cfm Error: Database"
"Information","web-1","11/20/01","16:28:21",,
 "Page: web_root/MYStuff/MyDocs/ cftryexample.cfm Error: General Exception"
"Information","web-0","11/20/01","16:28:49",,
 "Page: web_root/MYStuff/MyDocs/ cftryexample.cfm Error: "
```

Reviewing the code

The following table describes the code:

Code	Description

`<cfset EmpID=3>`
`<cfparam name="errorCaught"`
`default="">`
	Initializes the employee ID to a valid value. An application would get the value from a form or other source.Sets the default `errorCaught` variable value to the empty string (to indicate no error was caught).There is no need to put these lines in a `cftry` block.

`<cftry>`
` <cfquery name="test"`
`datasource="cfdocexamples">`
` SELECT Dept_ID, FirstName,`
`LastName`
` FROM Employee`
` WHERE Emp_ID=#EmpID#`
` </cfquery>`
	Starts the `cftry` block. Exceptions from here to the end of the block can be caught by `cfcatch` tags.Queries the cfdocexamples database to get the data for the employee identified by the `EmpID` variable.

© 2014 Adobe Systems Incorporated. All rights reserved.

`<html>` ` <head>` ` <title>Test cftry/cfcatch</title>` ` </head>` ` <body>` ` <cfinclude` `template="includeme.cfm">` ` <cfoutput query="test">` `  <p>Department: #Dept_ID# ` `  Last Name: #LastName# ` `  First Name: #FirstName#</p>` ` </cfoutput>`	Begins the HTML page. This section contains all the code that displays information if no errors occur.Includes the includeme.cfm page.Displays the user information record from the test query.

`<cfcatch type="MissingInclude">`
` <h1>Missing Include File</h1>`
` <cfoutput>`
` <ul>`
`  <li><b>Message:</b>`
`#cfcatch.Message#`
`  <li><b>Detail:</b>`
`#cfcatch.Detail#`
`  <li><b>Filename:</b>`
`#cfcatch.MissingFileName#`
` </ul>`
` </cfoutput>`
` <cfset errorCaught =`
`"MissingInclude">`
`</cfcatch>`
	Handles exceptions thrown when a page specified by the `cfinclude` tag cannot be found. Displays `cfcatch` variables, including the ColdFusion basic error message, detail message, and the name of the file that could not be found.Sets the `errorCaught` variable to indicate the error type.

`<cfcatch type="Database">`
` <h1>Database Error</h1>`
` <cfoutput>`
` <ul>`
`  <li><b>Message:</b>`
`#cfcatch.Message#`
`  <li><b>Native error code:</b>`
`#cfcatch.NativeErrorCode#`
`  <li><b>SQLState:</b>`
`#cfcatch.SQLState#`
`  <li><b>Detail:</b>`
`#cfcatch.Detail#`
` </ul>`
` </cfoutput>`
` <cfset errorCaught = "Database">`
`</cfcatch>`
	Handles exceptions thrown when accessing a database. Displays `cfcatch` variables, including the ColdFusion basic error message, the error code and SQL state reported by the databases system, and the detailed error message.Sets the `errorCaught` variable to indicate the error type.

© 2014 Adobe Systems Incorporated. All rights reserved.

<cfcatch type="Any"> <cfoutput> <hr> Other Error: #cfcatch.Type# Message: #cfcatch.Message# Detail: #cfcatch.Detail# </cfoutput> <cfset errorCaught = "General Exception"> </cfcatch>	Handles any other exceptions generated in the `cftry` block.Since the error can occur after information has displayed (in this case, the contents of the include file), draws a line before writing the message text.Displays the ColdFusion basic and detailed error message.Sets the `errorCaught` variable to indicate the error type.
</body> </html> </cftry>	Ends the HTML page, then the `cftry` block.

Using the cfthrow tag

You can use the `cfthrow` tag to raise your own, custom exceptions. When you use the `cfthrow` tag, you specify any or all of the following information:

Attribute	Meaning
type	The type of error. It can be a custom type that has meaning only to your application, such as InvalidProductCode. You can also specify Application, the default type. You cannot use any of the predefined ColdFusion error types, such as Database or MissingTemplate.
message	A brief text message indicating the error.
detail	A more detailed text message describing the error.
errorCode	An error code that is meaningful to the application. This field is useful if the application uses numeric error codes.
extendedInfo	Any additional information of use to the application.

All of these values are optional. You access the attribute values in `cfcatch` blocks and Exception type error pages by prefixing the attribute with either `cfcatch` or `error`, as in `cfcatch.extendedInfo`. The default ColdFusion

© 2014 Adobe Systems Incorporated. All rights reserved.

error handler displays the message and detail values in the Message pane and the remaining values in the Error Diagnostic Information pane.

Catching and displaying thrown errors

The `cfcatch` tag catches a custom exception when you use any of the following values for the `cfcatch type` attribute:

- The custom exception type specified in the `cfthrow` tag.
- A custom exception type that hierarchically matches the initial portion of the type specified in the `cfthrow` tag. For more information, see the next section, *Custom error type name hierarchy*.
- `Application`, which matches an exception that is thrown with the `Application type` attribute or with no `type` attribute.
- `Any`, which matches any exception that is not caught by a more specific `cfcatch` tag.
  Similarly, if you specify any of these types in a `cferror` tag, the specified error page displays information about the thrown error.
  Because the `cfthrow` tag generates an exception, a Request error handler or the Site-wide error handler can also display these errors.

Custom error type name hierarchy

You can name custom exception types using a method that is similar to Java class naming conventions: domain name in reverse order, followed by project identifiers, as in the following example:

```
<cfthrow
 type="com.myCompany.myApp.Invalid_field.codeValue"
 errorcode="Dodge14B">
```

This fully qualified naming method is not required; you can use shorter naming rules, for example, myApp.Invalid_field.codeValue, or even codeValue.
This naming method is *not* just a convention; ColdFusion uses the naming hierarchy to select from a possible hierarchy of error handlers. For example, assume that you use the following `cfthrow` statement:

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

Any of the following `cfcatch` error handlers would handle this error:

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

The handler that most exactly matches handles the error. In this case, the `MyApp.BusinessRuleException.InvalidAccount` handler runs. However, if you used the following `cfthrow` tag:

```
<cfthrow type="MyApp.BusinessRuleException.InvalidVendorCode
```

© 2014 Adobe Systems Incorporated. All rights reserved.

the `MyApp.BusinessRuleException` handler receives the error.
The type comparison is not case sensitive.

When to use the cfthrow tag

Use the `cfthrow` tag when your application can identify and handle application-specific errors. One typical use for the `cfthrow` tag is in implementing custom data validation. The `cfthrow` tag is also useful for throwing errors from a custom tag page to the calling page.
For example, on a form action page or custom tag used to set a password, the application can determine whether the password entered is a minimum length, or contains both letters and number, and throw an error with a message that indicates the password rule that was broken. The `cfcatch` block handles the error and tells the user how to correct the problem.

Using the cfrethrow tag

The `cfrethrow` tag lets you create a hierarchy of error handlers. It tells ColdFusion to exit the current `cfcatch` block and "rethrow" the exception to the next level of error handler. Thus, if an error handler designed for a specific type of error cannot handle the error, it can rethrow the error to a more general-purpose error handler. The `cfrethrow` tag can only be used in a `cfcatch` tag body.

The cfrethrow tag syntax

The following pseudocode shows how you can use the `cfrethrow` tag to create an error-handling hierarchy:

```
<cftry>
 <cftry>
  Code that might throw a database error
  <cfcatch Type="Database">
   <cfif Error is of type I can Handle>
    Handle it
   <cfelse>
    <cfrethrow>
   </cfif
  </cfcatch>
 </cftry>
 <cfcatch Type="Any">
  General Error Handling code
 </cfcatch>
</cftry>
```

Although this example uses a Database error as an example, you can use any `cfcatch` type attribute in the innermost error type.
Follow these rules when you use the `cfrethrow` tag:

- Nest `cftry` tags, with one tag for each level of error handling hierarchy. Each level contains the `cfcatch` tags for that level of error granularity.
- Place the most general error catching code in the outermost `cftry` block.
- Place the most specific error catching code in the innermost `cftry` block.
- Place the code that can cause an exception error at the top of the innermost `cftry` block.
- End each `cfcatch` block except those in the outermost `cftry` block with a `cfrethrow` tag.

Example: using nested tags, cfthrow, and cfrethrow

© 2014 Adobe Systems Incorporated. All rights reserved.

The following example shows many of the discussed techniques including nested `cftry` blocks and the `cfthrow` a nd `cfrethrow` tags. The example includes a simple calling page and a custom tag page:

- The calling page does little more than call the custom tag with a single attribute, a name to be looked up in a database. It does show, however, how a calling page can handle an exception thrown by the custom tag.
- The custom tag finds all records in the cfdocexamples database with a matching last name, and returns the results in a Caller variable. If it fails to connect with the main database, it tries a backup database.

The calling page

The calling page represents a section from a larger application page. To keep things simple, the example hard-codes the name to be looked up.

```
<cftry>
 <cf_getEmps EmpName="Jones">
 <cfcatch type="myApp.getUser.noEmpName">
  <h2>Oops</h2>
  <cfoutput>#cfcatch.Message#</cfoutput><br>
 </cfcatch>
</cftry>
<cfif isdefined("getEmpsResult")>
 <cfdump var="#getEmpsResult#">
</cfif>
```

Reviewing the code

The following table describes the code:

Code	Description
`<cftry>` `<cf_getEmps EmpName="Jones">`	In a `cftry` block, calls the cf_getEmps custom tag (getEmps.cfm).
`<cfcatch type="myApp.getUser.noEmpName">` `<h2>Oops</h2>` `<cfoutput>#cfcatch.Message#</cfoutput> ` `</cfcatch>`	If the tag throws an exception indicating that it did not receive a valid attribute, catches the exception and displays a message, including the message variable set by the `cfthrow` tag in the custom tag.

`<cfif isdefined("getEmpsResult")>`
`<cfdump var="#getEmpsResult#">`
`</cfif>`
	If the tag returns a result, uses the `cfdump` tag to display it. (A production application would not use the `cfdump` tag.)

© 2014 Adobe Systems Incorporated. All rights reserved.

The custom tag page

The custom tag page searches for the name in the database and returns any matching records in a getEmpsResult variable in the calling page. It includes several nested `cftry` blocks to handle error conditions. For a full description, see *Reviewing the code* section, following the example:Save the following code as getEmps.cfm in the same directory as the calling page.

```
<!--- If the tag didn't pass an attribute, throw an error to be handled by
   the calling page --->
<cfif NOT IsDefined("attributes.EmpName")>
 <cfthrow Type="myApp.getUser.noEmpName"
  message = "Last Name was not supplied to the cf_getEmps tag.">
 <cfexit method = "exittag">
<!--- Have a name to look up --->
<cfelse>
<!--- Outermost Try Block --->
 <cftry>

<!--- Inner Try Block --->
  <cftry>
<!--- Try to query the main database and set a caller variable to the result --->
   <cfquery Name = "getUser" DataSource="cfdocexamples">
    SELECT *
    FROM Employee
    WHERE LastName = '#attributes.EmpName#'
   </cfquery>
   <cfset caller.getEmpsResult = getuser>
<!--- If the query failed with a database error, check the error type
   to see if the database was found --->
   <cfcatch type= "Database">
    <cfif (cfcatch.SQLState IS "S100") OR (cfcatch.SQLState IS
     "IM002")>

<!--- If the database wasn't found, try the backup database --->
<!--- Use a third-level Try block --->
     <cftry>
      <cfquery Name = "getUser" DataSource="cfdocexamplesBackup">
       SELECT *
       FROM Employee
       WHERE LastName = '#attributes.EmpName#'
      </cfquery>
       <cfset caller.getEmpsResult = getuser>

<!--- If still get a database error, just return to the calling page
   without setting the caller variable. There is no cfcatch body.
   This might not be appropriate in some cases.
   The Calling page ends up handling this case as if a match was not
   found --->
       <cfcatch type = "Database" />
<!--- Still in innermost try block. Rethrow any other errors to the next
   try block level --->
      <cfcatch type = "Any">
       <cfrethrow>
      </cfcatch>
     </cftry>

<!--- Now in second level try block.
```

© 2014 Adobe Systems Incorporated. All rights reserved.

```
    Throw all other types of Database exceptions to the next try
     block level --->
     <cfelse>
      <cfrethrow>
     </cfif>
    </cfcatch>
<!--- Throw all other exceptions to the next try block level --->
    <cfcatch type = "Any">
      <cfrethrow>
    </cfcatch>
   </cftry>

<!--- Now in Outermost try block.
    Handle all unhandled exceptions, including rethrown exceptions, by
    displaying a message and exiting to the calling page.--->
   <cfcatch Type = "Any">
    <h2>Sorry</h2>
    <p>An unexpected error happened in processing your user inquiry.
     Please report the following to technical support:</p>
    <cfoutput>
     Type: #cfcatch.Type#
     Message: #cfcatch.Message#
    </cfoutput>
    <cfexit method = "exittag">
   </cfcatch>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

```
  </cftry>
 </cfif>
```

Reviewing the code

The following table describes the code:

Code	Description
`<cfif NOT IsDefined("attributes.EmpName")> <cfthrow Type="myApp.getUser.noEmpName" message = "Last Name was not supplied to the cf_getEmps tag."> <cfexit method = "exittag">`	Makes sure the calling page specified an `EmpName` attribute. If not, throws a custom error that indicates the problem and exits the tag. The calling page handles the thrown error.
`<cfelse> <cftry>`	If the tag has an `EmpName` attribute, does the remaining work inside an outermost try block. The `cfcatch` block at its end handles any otherwise-uncaught exceptions.
`<cftry> <!--- Try to query the main database and set a caller variable to the result ---> <cfquery Name = "getUser" DataSource="cfdocexamples">  SELECT *  FROM Employee  WHERE LastName = '#attributes.EmpName#' </cfquery> <cfset caller.getEmpsResult = getuser>`	Starts a second nested try block. This block catches exceptions in the database query. If there are no exceptions, sets the calling page's `getEmpsResult` variable with the query results.

© 2014 Adobe Systems Incorporated. All rights reserved.

`<cfcatch type= "Database">`
` <cfif (cfcatch.SQLState IS "S100") OR (cfcatch.SQLState IS "IM002")>`
`  <cftry>`
`  <cfquery Name = "getUser" DataSource="cfdocexamplesBackup">`
`  SELECT *`
`  FROM Employee`
`  WHERE LastName = '#attributes.EmpName#'`
`  </cfquery>`
`  <cfset caller.getEmpsResult = getuser>`
	If the query threw a Database error, checks to see if the error was caused by an inability to access the database (indicated by an `SQLState` variable value of S100 or IM002). If the database was not found, starts a third nested try block and tries accessing the backup database. This try block catches exceptions in this second database access.If the database inquiry succeeds, sets the calling page's `getEmpsResult` variable with the query results.

`<cfcatch type = "Database" />`	If the second database query failed with a database error, gives up silently. Because the Database type `cf catch` tag does not have a body, the tag exits. The calling page does not get a `getEmpsResult` variable. It cannot tell whether the database had no match or an unrecoverable database error occurred, but it does know that no match was found.

`<cfcatch type = "Any">`
` <cfrethrow>`
`</cfcatch>`
`</cftry>`
	If the second database query failed for any other reason, throws the error up to the next try block.Ends the innermost try block

`<cfelse>`
` <cfrethrow>`
` </cfif>`
`</cfcatch>`
	In the second try block, handles the case in which the first database query failed for a reason other than a failure to find the database. Rethrows the error up to the next level, the outermost try block.

`<cfcatch type = "Any">`
` <cfrethrow>`
`</cfcatch>`
`</cftry>`
	In the second try block, catches any errors other exceptions and rethrows them up to the outermost try block. Ends the second try block.

© 2014 Adobe Systems Incorporated. All rights reserved.

`<cfcatch Type = "Any">`
` <h2>Sorry</h2>`
` <p>An unexpected error happened`
`in processing your user inquiry.`
` Please report the following to`
`technical support:</p>`
` <cfoutput>`
` Type: #cfcatch.Type#`
` Message: #cfcatch.Message#`
` </cfoutput>`
` <cfexit method = "exittag">`
`</cfcatch>`
`</cftry>`
`</cfif>`
	In the outermost try block, handles any exceptions by displaying an error message that includes the exception type and the exception's error message. Because there was no code to try that is not also in a nested try block, this `cfcatch` tag handles only errors that are rethrown from the nested blocks.Exits the custom tag and returns to the calling page.Ends the catch block, try block, and initial `cfif` block.

Testing the code

To test the various ways errors can occur and be handled in this example, try the following:

- In the calling page, change the attribute name to any other value; for example, My Attrib. Then change it back.
- In the first `cfquery` tag, change the data source name to an invalid data source; for example, NoDatabase.
- With an invalid first data source name, change the data source in the second `cfquery` tag to cfdocexamples.
- Insert `cfthrow` tags throwing custom exculpations in various places in the code and observe the effects.

#back to top

© 2014 Adobe Systems Incorporated. All rights reserved.

Using Persistent Data and Locking

Adobe ColdFusion provides several variable scopes in which data persists past the life of a single request. These are the Client, Application, Session, and Server scopes. These scopes let you save data over time and share data between pages and even applications. Use these scopes as *persistent* scopes. In particular, use the Client and Session scopes to maintain information about a user across multiple requests.

ColdFusion lets you lock access to sections of code to ensure that ColdFusion does not attempt to run the code, or access the data that it uses, simultaneously or in an unpredictable order. This locking feature is important for ensuring the consistency of all shared data, including data in external sources in addition to data in persistent scopes.

You can use persistent scopes to develop an application and use locking to ensure data consistency.

© 2014 Adobe Systems Incorporated. All rights reserved.

About persistent scope variables

ColdFusion provides four variable scopes, described in the following table, that let you maintain data that must be available to multiple applications or users or must last beyond the scope of the current request.

Variable scope	Description
Client	Contains variables that are available for a single client browser over multiple browser sessions in an application. For information about browser sessions, see, *What is a session?* in Configuring and using session variables.Useful for client-specific information, such as client preferences, that you want to store for a significant period of time.Data is stored as cookies, database entries, or Registry values. Client variables can time out after an extended period. Although do not have to use the Client scope prefix in the variable name, code that uses the prefix is more efficient and easier to maintain.
Session	Contains variables that are available for a single client browser for a single browser session in an application. Useful for client-specific information, such as shopping cart contents, that you want to persist while the client is visiting your application.Data is stored in memory and times out after a period of inactivity or when the server shuts down. ColdFusion Administrator lets you select between two kinds of session management, Standard ColdFusion Session management and J2EE session management. For information about types of session management, see *ColdFusion and J2EE session management* in Configuring and using session variables.Use the Session scope prefix in the variable name.
Application	Contains variables that are available to all pages in an application for all clients. Useful for application-specific information, such as contact information, that can vary over time and should be stored in a variable.Data is stored in memory and times out after a period of inactivity or when the server shuts down.Use the Application scope prefix in the variable name.

© 2014 Adobe Systems Incorporated. All rights reserved.

Server	Contains variables that are available to all applications in a server and all clients. Useful for information that applies to all pages on the server, such as an aggregate page-hit counter.Data is stored in memory. The variables do not time out, but you can delete variables you create, and all server variables are automatically deleted when the server stops running.Use the Server scope prefix in the variable name.

The following sections provide information that is common to all or several of these variables. Later sections describe how to use the Client, Session, Application, and Server scopes in your applications, and provide detailed information about locking code.

ColdFusion persistent variables and ColdFusion structures

All persistent scopes are available as ColdFusion structures. As a result, you can use ColdFusion structure functions to access and manipulate Client, Session, Application, and Server scope contents. Information about using these functions in detail is not covered, but information about features or limitations that apply to specific scopes is provided.

> **Note**
>
> Although you can use the `StructClear` function to clear your data from the Server scope, the function does not delete the names of the variables, only their values, and it does not delete the contents of the Server.os and Server.ColdFusion structures. Using the `StructClear` function to clear the Session, or Application scope clears the entire scope, including the built-in variables. Using the `StructClear` function to clear the Client scope clears the variables from the server memory, but does not delete the stored copies of the variables.

ColdFusion persistent variable issues

Variables in the Session, Application, and Server scopes are kept in ColdFusion server memory. This storage method has several implications:

- All variables in these scopes are lost if the server stops running.
- Variables in these scopes are not shared by servers in a cluster.
- To prevent race conditions and ensure data consistency, lock access to all code that changes variables in these scopes or reads variables in these scopes with values that can change.

> **Note**
>
> If you use J2EE session management and configure the J2EE server to retain session data between server restarts, ColdFusion retains session variables between server restarts.

Additionally, be careful when using client variables in a server cluster, where an application can run on multiple servers.

Locking memory variables

Because ColdFusion is a multi-threaded system in which multiple requests can share Session, Application, and Server scope variables, it is possible for two or more requests to try to access and modify data at the same time. ColdFusion runs in a J2EE environment, which prevents simultaneous data access, so multiple requests do not

© 2014 Adobe Systems Incorporated. All rights reserved.

cause severe system errors. However, such requests can result in inconsistent data values, particularly when a page changes more than one variable.
To prevent data errors with session, application, and server variables, lock code that writes and reads data in these scopes. For more information, see Locking code with cflock.

Using variables in clustered systems

Because memory variables are stored in memory, they are not available to all servers in a cluster. As a result, you generally do not use Session, Application, or Server scope variables in clustered environment. However, use these scope variables in a clustered system in the following circumstances:

- If the clustering system supports "sticky" sessions, in which the clustering system ensures that each user session remains on a single server. In this case, you can use session variables as you would on a single server.
- You can use Application and Server scope variables in a cluster for write-once variables that are consistently set, for example, from a database.
  To use client variables on a clustered system, store the variables as cookies or in a database that is available to all servers. If you use database storage, on one server only, select the Purge Data for Clients that Remain Unvisited option on the Client Variables, Add/Edit Client Store page in the Server Settings area in the ColdFusion Administrator.

© 2014 Adobe Systems Incorporated. All rights reserved.

Managing the client state

Because the web is a stateless system, each connection that a browser makes to a web server is unique to the web server. However, many applications must keep track of users as they move through the pages within the application. This is the definition of *client state management*.
ColdFusion provides tools to maintain the client state by seamlessly tracking variables associated with a browser as the user moves from page to page within the application. You can use these variables in place of other methods for tracking client state, such as URL parameters, hidden form fields, and HTTP cookies.

About client and session variables

ColdFusion provides two tools for managing the client state: client variables and session variables. Both types of variables are associated with a specific client, but you manage and use them differently, as described in the following table:

Variable type	Description
Client	Data is saved as cookies, database entries, or Registry entries. Data is saved between server restarts, but is initially accessed and saved more slowly than data stored in memory.Each type of data storage has its own time-out period. You can specify the database and Registry data time-outs in the ColdFusion Administrator. ColdFusion sets Cookie client variables to expire after approximately ten years.Data is stored on a per-user and per-application basis. For example, if you store client variables as cookies, the user has a separate cookie for each ColdFusion application provided by a server.Client variables must be simple variables, such as numbers, dates, or strings. They cannot be arrays, structures, query objects, or other objects.Client variable names can include periods. For example, My.ClientVar is a valid name for a simple client variable. Avoid such names, however, to ensure code clarity.You do not have to prefix client variables with the scope name when you reference them, However, if you do not use the Client prefix, you might unintentionally refer to a variable with the same name in another scope. Using the prefix also optimizes performance and increases program clarity.You do not lock code that uses client variables.You can use client variables that are stored in cookies or a common database in clustered systems.

© 2014 Adobe Systems Incorporated. All rights reserved.

Session	Data is stored in memory so it is accessed quickly.Data is lost when the client browser is inactive for a time-out period. You specify the time-out in the ColdFusion Administrator, the Application.cfc initialization code, or Application.cfm.As with client variables, data is available to a single client and application only.Variables can store any ColdFusion data type.You must prefix all variable names with the Session scope name.Lock code that uses session variables to prevent race conditions.You can use session variables in clustered systems only if the systems support "sticky" sessions, where a session is limited to a single server.

Session variables are normally better than client variables for values that need to exist for only a single browser session. Reserve client variables for client-specific data, such as client preferences that you want available for multiple browser sessions.

Maintaining client identity

Because the web is a stateless system, client management requires some method for maintaining knowledge of the client between requests. Normally you do this using cookies, but you can also do it by passing information between application pages. Information about how ColdFusion maintains client identity in a variety of configurations and environments, and the issues that can arise with client state management are described as follows:

About client identifiers

To use client and session variables, ColdFusion must be able to identify the client. It normally does so by setting the following two cookie values on the client's system:

- **CFID**: A sequential client identifier
- **CFToken:** A random-number client security token

These cookies uniquely identify the client to ColdFusion, which also maintains copies of the variables as part of the Session and Client scopes. You can configure your application so that it does not use client cookies, but in this case, pass these variables to all the pages that your application calls. For more information about maintaining client and session information without using cookies, see Using client and session variables without cookies section below .

You can configure ColdFusion to use J2EE servlet session management instead of ColdFusion session management for session variables. This method of session management does not use `CFID` and `CFToken` values, but does use a client-side `jsessionid` session management cookie. For more information on using J2EE session management, see *ColdFusion and J2EE session management* in Configuring and using session variables.

Using client and session variables without cookies

Often, users disable cookies in their browsers. In this case, ColdFusion cannot maintain the client state automatically. You can use client or session variables without using cookies, by passing the client identification information between application pages. However, this technique has significant limitations, as follows:

1. Client variables are effectively the same as session variables, except that they leave unusable data in the client data store.Because the client's system does not retain any identification information, the next time the user logs on, ColdFusion cannot identify the user with the previous client and must create a new client ID for the user. Any information about the user from a previous session is not available, but remains in client data storage until ColdFusion deletes it. If you clear the Purge Data for Clients that Remain Unvisited option in the ColdFusion Administrator, ColdFusion never deletes this data.Therefore, do not use client variables, if you

© 2014 Adobe Systems Incorporated. All rights reserved.

allow users to disable cookies. To retain client information without cookies, require users to login with a unique ID. You can then save user-specific information in a database with the user's ID as a key.

2. ColdFusion creates a new session each time the user requests a page directly in the browser, because the new request contains no state information to indicate the session or client.

> **Note**
>
> You can prevent ColdFusion from sending client information to the browser as cookies by setting This.setClientCookies variable in Application.cfc or the `setClientCookies` attribute of the `cfapplication` tag to No.

To use ColdFusion session variables without using cookies, each page must pass the `CFID` and `CFToken` values to any page that it calls as part of the request URL. If a page contains any HTML `href a=` links, `cflocation` tags, `form` tags, or `cfform` tags the tags must pass the `CFID` and `CFToken` values in the tag URL. To use J2EE session management, pass the `jsessionid` value in page requests. To use ColdFusion client variables and J2EE session variables, pass the `CFID`, `CFToken`, and `jsessionid` values in URLs.

> **Note**
>
> The behavior is as follows when CFID and CFTOKEN are provided in the URL: If session exists, the CFID and CFTOKEN from the URL are ignored. If the session does not exist, CFID and CFTOKEN from the URL are used to validate the session and the session is used if it is valid. If the session is not valid, a new session is created. CFID and CFTOKEN are regenerated.

ColdFusion provides the `URLSessionFormat` function, which 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 the information.
  The `URLSessionFormat` function automatically determines which identifiers are required, and sends only the required information. It also provides a more secure and robust method for supporting client identification than manually encoding the information in each URL, because it only sends the information that is required, when it is required, and it is easier to code.
  To use the `URLSessionFormat` function, enclose the request URL in the function. For example, the following `cfform` tag posts a request to another page and sends the client identification, if necessary:

```
<cfform method="Post" action="#URLSessionFormat("MyActionPage.cfm")#>
```

> If you use the same page URL in multiple `URLSessionFormat_` functions, you can gain a small performance improvement and simplify your code if you assign the formatted page URL to a variable, for example:_

```
<cfset myEncodedURL=URLSessionFormat("MyActionPage.cfm")>
<cfform method="Post" action="#myEncodedURL#">
```

Client identifiers and security

© 2014 Adobe Systems Incorporated. All rights reserved.

The following client identifier issues can have security implications:

- Ensuring the uniqueness and complexity of the `CFToken` identifier
- Limiting the availability of Session identifiers
  The next sections discuss these issues.

Ensuring CFToken uniqueness and security

By default, ColdFusion uses an eight-digit random number in the `CFToken` identifier. This `CFToken` format provides a unique, secure identifier for users under most circumstances. (In ColdFusion, the method for generating this number uses a cryptographic-strength random number generator that is seeded only when the server starts.) However, in the ColdFusion Administrator, you can enable the Settings page to produce a more complex `CFToken` identifier. If you enable the Use UUID for cftoken option, ColdFusion creates the `CFToken` value by prepending a 16-digit random hexadecimal number to a ColdFusion UUID. The resulting `CFToken` identifier looks similar to the following:

```
3ee6c307a7278c7b-5278BEA6-1030-C351-3E33390F2EAD02B9
```

Providing Session security

ColdFusion uses the same client identifiers for the Client scope and the standard Session scope. Because the `CFToken` and `CFID` values are used to identify a client over a period of time, they are normally saved as cookies on the user's browser. These cookies persist until the client's browser deletes them, which can be a considerable length of time. As a result, hackers could have more access to these variables than if ColdFusion used different user identifiers for each session.
A hacker who has the user's `CFToken` and `CFID` cookies could gain access to user data by accessing a web page during the user's session using the stolen `CFToken` and `CFID` cookies. While this scenario is unlikely, it is theoretically possible.
You can remove this vulnerability by selecting the Use J2EE Session Variables option on the ColdFusion Administrator Memory Variables page. The J2EE session management mechanism creates a new session identifier for each session, and does not use either the `CFToken` or the `CFID` cookie value.

Security-related changes

The following security-related specifications apply when you upgrade to *ColdFusion 9 Upgrade 1*:

- CFID, CFTOKEN, and jsessionid are marked `httpOnly`. This reduces the chance of session information being compromised on Cross Site Scripting (XSS) attack.
- Set the following system property for the session cookies to be httpOnly: `Dcoldfusion.sessioncookie.httponly=true`
- The support for session cookies to be `httpOnly`depends on the application server you use:
  - For Tomcat/JBoss, `httpOnly` is not supported for JSESSIONID
  - On JRun, add the system property, Dcoldfusion.sessioncookie.httponly=true , in the jvm.config file
  - For other application servers, see the relevant documentation for details on `httpOnly` support for session cookies.

Managing client identity information in a clustered environment

To maintain your application's client identity information in a clustered server environment, you must specify This.setdomaincookies="True" in the Application.cfc initialization code, or use the `cfapplication` `setdomaincookies` attribute in your Application.cfm page.

© 2014 Adobe Systems Incorporated. All rights reserved.

The `setdomaincookies` attribute specifies that the server-side copies of the `CFID` and `CFToken` variables used to identify the client to ColdFusion are stored at the domain level (for example, .adobe.com). If `CFID` and `CFToken` variable combinations exist on each host in the cluster, ColdFusion migrates the host-level variables on each cluster member to the single, common domain-level variable. Following the setting or migration of host-level cookie variables to domain-level variables, ColdFusion creates a new cookie variable (`CFMagic`) that tells ColdFusion that domain-level cookies have been set.

If you use client variables in a clustered system, you must also use a database or cookies to store the variables.

© 2014 Adobe Systems Incorporated. All rights reserved.

Configuring and using client variables

Use client variables for data that is associated with a particular client and application and that must be saved between user sessions. Use client variables for long-term information such as user display or content preferences.

Enabling client variables

To enable client variables, you specify This.clientmanagement="True" in the Application.cfc initialization code, or set the `cfapplication` tag `clientmanagement` attribute to Yes in the Application.cfm file. For example, to enable client variables in an application named SearchApp, you can use the following line in the application's Application.cfm page:

```
<cfapplication NAME="SearchApp" clientmanagement="Yes">
```

Choosing a client variable storage method

By default, ColdFusion stores client variables in the Registry. In most cases, however, it is more appropriate to store the information as client cookies or in a SQL database.
The ColdFusion Administrator Client Variables page controls the default client variable location. You can override the default location by specifying a This.clientStorage value in Application.cfc or by setting the `clientStorage` attribute in the `cfapplication` tag.
You can specify the following values for the client storage method:

- `Registry` (default). Client variables are stored under the key HKEY_LOCAL_MACHINE\SOFTWARE\Macromedia\ColdFusion\CurrentVersion\Clients.
- Name of a data source configured in ColdFusion Administrator
- `Cookie`
  Generally, it is most efficient to store client variables in a database. Although the Registry option is the default, the Registry has significant limitations for client data storage. The Registry cannot be used in clustered systems and its use for client variables on UNIX is not supported in ColdFusion.

Using cookie storage

When you set the client storage method to Cookie, the cookie that ColdFusion creates has the application's name. Storing client data in a cookie is scalable to large numbers of clients, but this storage mechanism has some limitations. In particular, if the client turns off cookies in the browser, client variables do not work.
Consider the following additional limitations before implementing cookie storage for client variables:

- Any Client variable that you set after a `cfflush` tag is not sent to the browser, so the variable value does not get saved.
- Some browsers allow only 20 cookies to be set from a particular host. ColdFusion uses two of these cookies for the `CFID` and `CFToken` identifiers, and also creates a cookie named `cfglobals` to hold global data about the client, such as `HitCount`, `TimeCreated`, and `LastVisit`. This limits you to 17 unique applications per client-host pair.
- Some browsers set a size limit of 4K bytes per cookie. ColdFusion encodes nonalphanumeric data in cookies with a URL encoding scheme that expands at a 3-1 ratio, which means you should not store large amounts of data per client. ColdFusion throws an error if you try to store more than 4,000 encoded bytes of data for a client.

Configuring database storage

When you specify a database for client variable storage, do not always have to manually create the data tables that store the client variables.

© 2014 Adobe Systems Incorporated. All rights reserved.

If ColdFusion can identify that the database you are using supports SQL creation of database tables, create the database in advance. When you click the Add button on the Select Data Source to Add as Client Store box on the Memory Variables page, the Administrator displays a Add/Edit Client Store page which contains a Create Client Database Tables selection box. Select this option to have ColdFusion create the necessary tables in your database. (The option does not appear if the database already has the required tables.)
If your database does not support SQL creation of tables, or if you are using the ODBC socket Macromedia driver to access your database, use your database tool to create the client variable tables. Create the CDATA and CGLOBAL tables.
The CDATA table must have the following columns:

Column	Data type
cfid	CHAR(64), TEXT, VARCHAR, or any data type capable of taking variable length strings up to 64 characters
app	CHAR(64), TEXT, VARCHAR, or any data type capable of taking variable length strings up to 64 characters
data	MEMO, LONGTEXT, LONG VARCHAR, CLOB, or any data type capable of taking long, indeterminate-length strings

The CGLOBAL table must have the following columns:

Column	Data type
cfid	CHAR(64), TEXT, VARCHAR, or any data type capable of taking variable length strings up to 64 characters
data	MEMO, LONGTEXT, LONG VARCHAR, CLOB, or any data type capable of taking long, indeterminate-length strings
lvisit	TIMESTAMP, DATETIME, DATE, or any data type that stores date and time values

Note

Different databases use different names for their data types. The names in the preceding tables are common, but your database might use other names.

To improve performance, create indexes when you create these tables. For the CDATA table, index these cfid and app columns. For the CGLOBAL table, index the cfid column.

Specifying client variable storage in your application

The override the default client variable storage location, set the This.clientstorage variable in the Application.cfc

© 2014 Adobe Systems Incorporated. All rights reserved.

initialization code, or use the cfapplication tag clientStorage attribute.
The following lines from an Application.cfc file tell ColdFusion to store the client variables in the mydatasource data source:

```
<cfscript>
 This.name"SearchApp";
 This.clientManagement="Yes";
 This.clientStorage="mydatasource";
</cfscript>
```

The following code from an Application.cfm file does the same thing as the previous example:

```
<cfapplication name"SearchApp"
 clientmanagement="Yes"
 clientstorage="mydatasource">
```

Using client variables

When you enable client variables for an application, you can use them to keep track of long-term information that is associated with a particular client.
Client variables must be simple data types: strings, numbers, lists, Booleans, or date and time values. They cannot be arrays, recordsets, XML objects, query objects, or other objects. If you must store a complex data type as a client variable, you can use the cfwddx tag to convert the data to WDDX format (which is represented as a string), store the WDDX data, and use the cfwddx tag to convert the data back when you read it. For more information on using WDDX, see Using WDDX.

> **Note**
>
> When saving client variable data in WDDX format, in the case of the registry and SQL Server, the limit is about 4K; with ORACLE, the limit is about 2K.

Creating a client variable

To create a client variable and set its value, use the cfset or cfparam tag and use the Client scope identifier as a variable prefix; for example:

```
<cfset Client.FavoriteColor="Red">
```

After you set a client variable this way, it is available for use within any page in your application that is accessed by the client for whom the variable is set.
The following example shows how to use the cfparam tag to check for the existence of a client parameter and set a default value if the parameter does not exist:

```
<cfparam name="Client.FavoriteColor" default="Red">
```

© 2014 Adobe Systems Incorporated. All rights reserved.

Accessing and changing client variables

You use the same syntax to access a client variable as for other types of variables. You can use client variables anywhere you use other ColdFusion variables.
To display the favorite color that has been set for a specific user, for example, use the following code:

```
<cfoutput>
 Your favorite color is #Client.FavoriteColor#.
</cfoutput>
```

To change the client's favorite color, for example, use code such as the following:

```
<cfset Client.FavoriteColor = Form.FavoriteColor>
```

Standard client variables

The Client scope has the following built-in, read-only variables that your application can use:

Variable	Description
Client.CFID	The client ID, normally stored on the client system as a cookie.
Client.CFToken	The client security token, normally stored on the client system as a cookie.
Client.URLToken	Value depends on whether J2EE session management is enabled. No session management or ColdFusion session management: A combination of the `CFID` and `CFToken` values, in the form `CFID`=IDNum{{&CFTOKEN=}}tokenNum. This variable is useful if the client does not support cookies and you pass the `CFID` and `CFToken` variables from page to page.J2EE session management: A combination of `CFID, CFToken`, and session ID values in the form {{CFID=}}IDNum{{&CFTOKEN=}}tokenNum{{&jsessionid=}}SessionID.
Client.HitCount	The number of page requests made by the client.
Client.LastVisit	The last time the client visited the application.
Client.TimeCreated	The time the `CFID` and `CFToken` variables that identify the client to ColdFusion were first created.

© 2014 Adobe Systems Incorporated. All rights reserved.

> **Note**
>
> ColdFusion lets you delete or change the values of the built-in client variables. As a general rule, avoid doing so.

You use the `Client.CFID`, `Client.CFToken`, and `Client.URLToken` variables if your application supports browsers that do not allow cookies. For more information on supporting browsers that do not allow cookies, see *Using client and session variables without cookies* in Managing the client state.
You can use the `Client.HitCount` and time information variables to customize behavior that depends on how often users visit your site and when they last visited. For example, the following code shows the date of a user's last visit to your site:

```
<cfoutput>
 Welcome back to the Web SuperShop. Your last
 visit was on #DateFormat(Client.LastVisit)#.
</cfoutput>
```

Getting a list of client variables

To obtain a list of the custom client parameters associated with a particular client, use the `GetClientVariablesList` function, as follows:

```
<cfoutput>#GetClientVariablesList()#</cfoutput>
```

The `GetClientVariablesList` function returns a comma-separated list of the names of the client variables for the current application. The standard system-provided client variables (`CFID`, `CFToken`, `URLToken`, `HitCount`, `TimeCreated`, and `LastVisit`) are not returned in the list.

Deleting client variables

To delete a client variable, use the `StructDelete` function or the `DeleteClientVariable` function. For example, the following lines are equivalent:

```
<cfset IsDeleteSuccessful=DeleteClientVariable("MyClientVariable")>
<cfset IsDeleteSuccessful=StructDelete(Client, "MyClientVariable")>
```

The Client Variables page in the ColdFusion Administrator lets you set a time-out period of inactivity after which ColdFusion removes client variables stored in either the Registry or a data source. (The default value is 10 days for client variables stored in the Registry, and 90 days for client variables stored in a data source.)

> **Note**
>
> You cannot delete the system-provided client variables (`CFID`, `CFToken`, `URLToken`, `HitCount`, `TimeCreated`, and `LastVisit`).

© 2014 Adobe Systems Incorporated. All rights reserved.

Using client variables with cflocation

If you use the `cflocation` tag to redirect ColdFusion to a path that ends with .dbm or .cfm, the `Client.URLToken` variable is automatically appended to the URL. You can prevent this behavior by adding the attribute `addtoken="No"` to the `cflocation` tag.

Caching client variable

When ColdFusion reads or writes client variables, it caches the variables in memory to help decrease the overhead of accessing the client data. As a result, ColdFusion only accesses the client data store when you read its value for the first time or, for values you set, when the request ends. Additional references to the client variable use the cached value in ColdFusion memory, thereby processing the page more quickly.

Exporting the client variable database

If your client variable database is stored in the Windows system Registry and you need to move it to another machine, you can export the Registry key that stores your client variables and take it to your new server. The system Registry lets you export and import Registry entries.
To export your client variable database from the Registry in Windows:

1. Open the Registry editor.
2. Find and select the following key:

```
HKEY_LOCAL_MACHINE\SOFTWARE\Macromedia\ColdFusion\CurrentVersion\Clients
```

3. On the Registry menu, click Export Registry File.
4. Enter a name for the Registry file.
   After you create a Registry file, you can copy it to a new machine and import it by clicking Import Registry File on the Registry editor Registry menu.

> **Note**
>
> On UNIX systems, the registry entries are kept in /opt/coldfusion/registry/cf.registry, a text file that you can copy and edit directly.

© 2014 Adobe Systems Incorporated. All rights reserved.

Configuring and using session variables

Use session variables when you need the variables for a single site visit or set of requests within a short period of time (such as hours). For example, use session variables to store a user's selections in a shopping cart application. (Use client variables if you need a variable in multiple visits, such as over days, weeks, or months.)

> Place code that uses session variables inside `cflock` tags in circumstances that could result in race conditions from multiple accesses to the same variable. For information on using cflock tags see `Locking code with cflock`.

What is a session?

A **session** refers to all the connections that a single client makes to a server in the course of viewing any pages associated with a given application. Sessions are specific to both the individual user and the application. As a result, every user of an application has a separate session and has access to a separate set of session variables.

This logical view of a session begins with the first connection to an application by a client and ends after that client's last connection. However, because of the stateless nature of the web, it is not always possible to define a precise point at which a session ends. A session should end when the user finishes using an application. In most cases, however, a web application has no way of knowing if a user has finished or is just lingering over a page.

Therefore, sessions always terminate after a time-out period of inactivity. If the user does not access a page of the application within this time-out period, ColdFusion interprets this as the end of the session and clears any variables associated with that session.

The default time-out for session variables is 20 mins. You can change the default time-out on the Memory Variables page in the Server Settings area in the ColdFusion Administrator.

You can also set the time-out period for session variables inside a specific application (thereby overruling the Administrator default setting) by setting the Application.cfc This.sessionTimeout variable or by using the `cfapplication` tag `sessionTimeout` attribute. However, you cannot set a time-out value for that is greater than the maximum session time-out value set on the Administrator Memory Variables page.

For detailed information on ending sessions and deleting session variables, see *Ending a session* in this page.

ColdFusion and J2EE session management

The ColdFusion server can use either of the following types of session management:

- ColdFusion session management
- J2EE session management

ColdFusion session management uses the same client identification method as ColdFusion client management.

J2EE session management provides the following advantages over ColdFusion session management:

- J2EE session management uses a session-specific session identifier, `jsessionid`, which is created afresh at the start of each session.
- You can share session variables between ColdFusion pages and JSP pages or Java servlets that you call from the ColdFusion pages.
- The Session scope is serializable (convertible into a sequence of bytes that can later be fully restored into the original object). With ColdFusion session management, the Session scope is not serializable. Only serializable scopes can be shared across servers.

© 2014 Adobe Systems Incorporated. All rights reserved.

Therefore, consider using J2EE session management in any of the following cases:

- You want to maximize session security, particularly if you also use client variables
- You want to share session variables between ColdFusion pages and JSP pages or servlets in a single application.
- You want to be able to manually terminate a session while maintaining the client identification cookie for use by the Client scope.
- You want to support clustered sessions; for example, to support session failover among servers.

Configuring and enabling session variables

To use session variables, enable them in two places:

- ColdFusion Administrator
- The Application.cfc initialization code This.sessionManagement variable or the active `cfapplication` tag.

  ColdFusion Administrator, Application.cfc, and the `cfapplication` tag also provide facilities for configuring session variable behavior, including the variable time-out.

Selecting and enabling session variables in ColdFusion Administrator

To use session variables, they must be enabled on the ColdFusion Administrator Memory Variables page. (They are enabled by default.) You can also use the Administrator Memory Variables page to do the following:

- Select to use ColdFusion session management (the default) or J2EE session management.
- Change the default session time-out. Application code can override this value. The default value for this time-out is 20 mins.
- Specify a maximum session time-out. Application code cannot set a time-out greater than this value. The default value for this time-out is two days.

Enabling session variables in your application

Enable session variables in the initialization code of your Application.cfc file or in the `cfapplication` tag in your Application.cfm file.

Do the following in the Application.cfc initialization code, below the `cfcomponent` tag, to enable session variables:

- Set This.sessionManagement="Yes".
- Set This.name to specify the name of the application.
- Optionally, set This.sessionTimeout to set an application-specific session time-out value. Use the `CreateTimeSpan` function to specify the number of days, hours, minutes, and seconds for the time-out.

Do the following in the Application.cfm file to enable session variables:

- Set `sessionManagement="Yes"`
- Use the `name` attribute to specify the name of the application.
- Optionally, use the `sessionTimeout` attribute to set an application-specific session time-out value. Use the `CreateTimeSpan` function to specify the number of days, hours, minutes, and seconds for the time-out.

  The following sample code enables session management for the GetLeadApp application and sets the session variables to time out after a 45-minute period of inactivity:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cfapplication name="GetLeadApp"
 sessionmanagement="Yes"
 sessiontimeout=#CreateTimeSpan(0,0,45,0)#>
```

Storing session data in session variables

Session variables are designed to store session-level data. They are a convenient place to store information that all pages of your application might need during a user session, such as shopping cart contents or score counters.

Using session variables, an application can initialize itself with user-specific data the first time a user accesses one of the pages of the application. This information can remain available while that user continues to use that application. For example, you can retrieve information about a specific user's preferences from a database once, the first time a user accesses any page of an application. This information remains available throughout that user's session, thereby avoiding the overhead of retrieving the preferences repeatedly.

Standard session variables

If you use ColdFusion session variables, the Session scope has four built-in, read-only variables that your application can use. If you use J2EE session management, the Session scope has two built-in variables. Generally, you use these variables in your ColdFusion pages only if your application supports browsers that do not allow cookies. For more information on supporting browsers that do not allow cookies, see *Using client and session variables without cookies* in Managing the client state. The following table describes the built-in session variables.

Variable	Description
Session.CFID	ColdFusion session management only: the client ID, normally stored on the client system as a cookie.
Session.CFToken	ColdFusion session management only: the client security token, normally stored on the client system as a cookie.
Session.URLToken	ColdFusion session management: A combination of the `CFID` and `CFToken` values in the form `CFID=}}IDNum{{&CFTOKEN=}}tokenNum. Use this variable if the client does not support cookies and you must pass the {{CFID` and `CFToken` variables from page to page.J2EE session management: A combination of the `CFID` and `CFToken` cookies and the J2EE session ID, in the form {{CFID=}}IDNum{{&CFTOKEN=}}tokenNum{{&jsessionid=}}SessionID.
Session.SessionID	A unique identifier for the session.ColdFusion session management: a combination of the application name and `CFID` and `CFToken` values.J2EE session management: the `jsessionid` value.

© 2014 Adobe Systems Incorporated. All rights reserved.

> **Note**
>
> ColdFusion lets you delete or change the values of the built-in session variables. As a general rule, avoid doing so.

If you enable client variables and ColdFusion session management, ColdFusion uses the same values for the Client and Session scope `CFID`, `CFToken`, and `URLtoken` variables. ColdFusion gets the values for these variables from the same source, the client's `CFID` and `CFTOKEN` cookies.

If you use J2EE session management, the Session scope does not include the `Session.CFID` or `Session.CFToken` variables, but does include the `Session.URLToken` and `Session.SessionID` variables. In this case, the `Session.SessionID` is the J2EE session ID and `Session.URLToken` consists of the string `jsessionid=` followed by the J2EE session ID.

Getting a list of session variables

Use the `StructKeyList` function to get a list of session variables, as follows:

```
<cflock timeout=20 scope="Session" type="Readonly">
 <cfoutput> #StructKeyList(Session)# </cfoutput>
</cflock>
```

> Always put code that accesses session variables inside `cflock`_ tags._

Creating and deleting session variables

Use a standard assignment statement to create a new session variable, as follows:

```
<cflock timeout=20 scope="Session" type="Exclusive">
 <cfset Session.ShoppingCartItems = 0>
</cflock>
```

Use the `structdelete` tag to delete a session variable; for example:

```
<cflock timeout=20 scope="Session" type="Exclusive">
 <cfset StructDelete(Session, "ShoppingCartItems")>
</cflock>
```

> **Note**
>
> If you set session variables on a CFML template that uses the `cflocation` tag, ColdFusion might not set the variables. For more information, see TechNote at www.adobe.com/go/tn_18171 .

Accessing and changing session variables

© 2014 Adobe Systems Incorporated. All rights reserved.

You use the same syntax to access a session variable as for other types of variables. However, lock any code that accesses or changes session variables.

For example, to display the number of items in a user's shopping cart, use the following code:

```
<cflock timeout=20 scope="Session" type="Exclusive">
 <cfoutput>
  Your shopping cart has #Session.ShoppingCartItems# items.
 </cfoutput>
</cflock>
```

To increase the number of items in the shopping cart, use the following code:

```
<cflock timeout=20 scope="Session" type="Exclusive">
 <cfset Session.ShoppingCartItems = Session.ShoppingCartItems + 1>
</cflock>
```

Ending a session

The following rules apply to ending a session and deleting Session scope variables:

- If you use ColdFusion session management, ColdFusion automatically ends sessions and deletes all Session scope variables if the client is inactive for the session time-out period. The session does not end when the user closes the browser.
- If you use J2EE session management, ColdFusion ends the session and deletes all Session scope variables if the client is inactive for the session time-out period. However, the browser continues to send the same session ID, and ColdFusion reuses this ID for sessions with this browser instance, as long as the browser remains active.
- Logging a user out does not end the session or delete Session scope variables.
- In many cases, you can effectively end a session by clearing the Session scope, as shown in the following line. The following list, however, includes important limitations and alternatives:

  ```
  <cfset StructClear(Session)>
  ```

- Clearing the Session scope does not clear the session ID, and future requests from the browser continue to use the same session ID until the browser exits. It also does not log out the user, even if you use Session scope storage for login information. Always use the `cflogout` tag to log out users.
- If you use J2EE session management, you can invalidate the session, as follows:

  ```
  <cfset getPageContext().getSession().invalidate()>
  ```

  This line creates a pointer to the servlet page context and calls an internal method to reset the session. This clears all session information, including the session ID Session scope variables, and if you are using session

© 2014 Adobe Systems Incorporated. All rights reserved.

login storage, the login information, for future request. However, the session information does remain available until the end of the current request. After you invalidate a session, attempts by the browser to access the application will generate an invalid session exception until the session times out.

> **Note**
>
> You cannot destroy the session and create a session on the same request, as creating a new session involves sending session cookies back.

- If you do not use client cookies, the Session scope and login state is available to your application only as long as you pass the session's CFID, CFTOKEN, and, for J2EE sessions, jsessionid values in the URL query string. After you stop using these values, however, the session data remains in memory until the session time-out period elapses.

© 2014 Adobe Systems Incorporated. All rights reserved.

Configuring and using application variables

Application variables are available to all pages within an application, that is, pages that have the same application name. Because application variables are persistent, you easily can pass values between pages. You can use application variables for information including the application name, background color, data source names, or contact information.

You set the application name in the `cfapplication` tag, normally on your application's Application.cfm page. The application name is stored in the `Application.applicationName` variable.

Unlike client and session variables, application variables do not require that a client name (client ID) be associated with them. They are available to any clients that use pages in the application.

> Place code that uses application variables inside `cflock` tags in circumstances that could result in race conditions from multiple accesses to the same variable. For information on using `cflock` tags, see Locking code with cflock.

Configuring and enabling application variables

To use application variables, do the following:

- Ensure that they are enabled in the ColdFusion Administrator. (They are enabled by default.)
- Specify the application name by setting the This.name variable in the initialization code of the Application.cfc or by setting the `name` attribute of the `cfapplication` tag for the current page.

> **Note**
>
> ColdFusion supports unnamed applications for compatibility with J2EE applications. For more information, see *Unnamed ColdFusion Application and Session scopes* in Interoperating with JSP pages and servlets

The ColdFusion Administrator also lets you specify the following information:

- A default variable time-out. If all pages in an application are inactive for the time-out period, ColdFusion deletes all the application variables. The Application.cfc file or `cfapplication` tag can override this value for a specific application. The default value for this time-out is two days.
- A maximum time-out. The application code cannot set a time-out greater than this value. The default value for this time-out is two days.
  You can set the time-out period for application variables within a specific application by using the This.applicationTimeout variable of Application.cfc or the `applicationTimeout` attribute of the `cfapplication` tag.

Storing application data in application variables

Application variables are a convenient place to store information that all pages of your application might need, no matter which client is running that application. Using application variables, an application could, for example, initialize itself when the first user accesses any page of that application. This information can then remain available indefinitely, thereby avoiding the overhead of repeated initialization.

Because the data stored in application variables is available to all pages of an application, and remains available until a specific period of inactivity passes or the ColdFusion server shuts down, application variables are convenient for application-global, persistent data.

However, because all clients running an application see the same set of application variables, these variables are not appropriate for client-specific or session-specific information. To target variables for specific clients, use client or session variables.

Using application variables

© 2014 Adobe Systems Incorporated. All rights reserved.

Generally, application variables hold information that you write infrequently. In most cases, the values of these variables are set once, most often when an application first starts. Then the values of these variables are referenced many times throughout the life of the application or the course of a session.

In circumstances that could result in race conditions from multiple accesses to the same variable, place code that writes to Application scope variables or reads Application scope variables with data that can change inside `cflock` tags.

Because each Application scope variable is shared in memory by all requests in the application, these variables can become bottlenecks if used inappropriately. Whenever a request is reading or writing an Application scope variable, any other requests that use the variable must wait until the code accessing the variable completes. This problem is increased by the processing time required for locking. If many users access the application simultaneously and you use Application scope variables extensively, your application performance might degrade. If your application uses many application variables, consider whether the variables must be in the Application scope or whether they can be Session or Request scope variables.

The application scope has one built-in variable, `Application.applicationName`, which contains the application name you specify in the `cfapplication` tag.

Access and manipulate application variables the same way you use session variables, except that you use the variable prefix Application, not Session, and specify Session as the lock scope. For examples of using session variables see *Creating and deleting session variables* and *Accessing and changing session variables* in Configuring and using session variables. For information on locking write-once read-many application variables efficiently, see *Locking application variables efficiently* in Locking code with cflock.

© 2014 Adobe Systems Incorporated. All rights reserved.

Using server variables

Server variables are associated with a single ColdFusion server. They are available to all applications that run on the server. Use server variables for data that must be accessed across clients and applications, such as global server hit counts.
Server variables do not time out, but they are lost when the server shuts down. You can delete server variables.
Server variables are stored on a single server. As a result, do not use server variables if you use ColdFusion on a server cluster.
You access and manipulate server variables the same way use Session and application variables, except you use the variable prefix Server.

> Place code that uses server variables inside `cflock_` tags in circumstances that could result in race conditions from multiple accesses to the same variable. You do not have to lock access to built-in server variables._

ColdFusion provides the following standard built-in read-only server variables:

Variable	Description
Server.ColdFusion.AppServer	The name of the J2EE application server ColdFusion is using. For ColdFusion server editions, which have an integrated application server, the name is JRun4.
Server.ColdFusion.Expiration	The date on which the ColdFusion license expires if it is the trial version.
Server.ColdFusion.ProductLevel	The server product level, such as Enterprise.
Server.ColdFusion.ProductName	The name of the product (ColdFusion).
Server.ColdFusion.ProductVersion	The version number for the server that is running, such as 6,0,0.
Server.ColdFusion.Rootdir	Directory under which ColdFusion is installed, such as C:\cfusion.
Server.ColdFusion.SerialNumber	The serial number assigned to this server installation.
Server.ColdFusion.SupportedLocales	The locales, such as English (US) and Spanish (Standard), supported by the server.
Server.OS.AdditionalInformation	Additional information provided by the operating system, such as the Service Pack number.
Server.OS.arch	The processor architecture, such as x86 for Intel Pentium processors.
Server.OS.BuildNumber	The specific operating system build, such as 1381

© 2014 Adobe Systems Incorporated. All rights reserved.

Server.OS.Name	The name of the operating system, such as Windows NT.
Server.OS.Version	The version number of the operating system, such as 4.0.

© 2014 Adobe Systems Incorporated. All rights reserved.

Locking code with cflock

The `cflock` tag controls simultaneous access to ColdFusion code. The `cflock` tag lets you do the following:

- Protect sections of code that access and manipulate shared data in the Session, Application, and Server scopes, and in the Request and Variables scopes for applications that use ColdFusion threads.
- Ensure that file updates do not fail because files are open for writing by other applications or ColdFusion tags.
- Ensure that applications do not try to simultaneously access ColdFusion extension tags written using the CFX API that are not thread-safe. This is important for CFX tags that use shared (global) data structures without protecting them from simultaneous access (not thread-safe). However, Java CFX tags can also access shared resources that could become inconsistent if the CFX tag access is not locked.
- Ensure that applications do not try to simultaneously access databases that are not thread-safe. (This is not necessary for most database systems.)

ColdFusion is a multi-threaded web application server that can process multiple page requests at a time. As a result, the server can attempt to access the same information or resources simultaneously, as the result of two or more requests.

Although ColdFusion is thread-safe and does not try to modify a variable simultaneously, it does not ensure the correct order of access to information. If multiple pages, or multiple invocations of a page, attempt to write data simultaneously, or read and write it at the same time, the resulting data can be inconsistent, as shown in the following *Sample locking scenarios* section.

Similarly, ColdFusion cannot automatically ensure that two sections of code do not attempt to access external resources such as files, databases, or CFX tags that cannot properly handle simultaneous requests. Nor can ColdFusion ensure that the order of access to these shared resources is consistent and results in valid data.

By locking code that accesses such resources so that only one thread can access the resource at a time, you can prevent race conditions.

Sample locking scenarios

The following examples present scenarios in which you need to lock ColdFusion code. These scenarios show only two of the circumstances where locking is vital.

Reading and writing a shared variable

If you have an application-wide value, such as a counter of the total number of tickets sold, you could have code such as the following on a login page:

```
<cfset Application.totalTicketsSold = Application.totalTicketsSold + ticketOrder>
```

When ColdFusion executes this code, it performs the following operations:

1. Retrieves the current value of Application.totalTicketsSold from temporary storage.
2. Increments this value.
3. Stores the result back in the Application scope.

Suppose that ColdFusion processes two ticket orders at approximately the same time, and that the value of Application.totalTicketsSold is initially 160. The following sequence might happen:

1. Order 1 reads the total tickets sold as 160.
2. Order 2 reads the total tickets sold as 160.
3. Order 1 adds an order of 5 tickets to 160 to get 165.

© 2014 Adobe Systems Incorporated. All rights reserved.

4. Order 2 adds an order of 3 tickets to 160 to get 163.
5. Order 1 saves the value 165 to Application.totalTicketsSold
6. Order 2 saves the value 163 to Application.totalTicketsSold

The application now has an inaccurate count of the tickets sold, and is in danger of selling more tickets than the auditorium can hold.
To prevent this from happening, lock the code that increments the counter, as follows:

```
<cflock scope="Application" timeout="10" type="Exclusive">
 <cfset Application.totalTicketsSold = Application.totalTicketsSold + ticketOrder>
</cflock>
```

The `cflock` tag ensures that while ColdFusion performs the processing in the tag body, no other threads can access the Application scope. As a result, the second transaction is not processed until the first one completes. The processing sequence looks something like the following:

1. Order 1 reaches the lock tag, which gets an Application scope lock.
2. Order 1 reads the total tickets sold as 160.
3. Order 2 reaches the lock tag. Because there is an active Application scope lock, ColdFusion waits for the lock to free.
4. Order 1 adds an order of 5 tickets to 160 to get 165.
5. Order 1 saves the value 165 to Application.totalTicketsSold.
6. Order 1 exits the lock tag. The Application scope lock is now free.
7. Order 2 gets the Application scope lock and can begin processing.
8. Order 2 reads the total tickets sold as 165.
9. Order 2 adds an order of 3 tickets to 165 to get 168.
10. Order 2 saves the value 168 to Application.totalTicketsSold.
11. Order 2 exits the lock tag, which frees the Application scope lock. ColdFusion can process another order.

The resulting Application.totalTickesSold value is now correct.

Ensuring consistency of multiple variables

Often an application sets multiple shared scope variables at one time, such as many values submitted by a user on a form. If the user submits the form, clicks the back button, and then resubmits the form with different data, the application can end up with a mixture of data from the two submissions, in much the same manner as shown in the previous section.
For example, an application stores information about order items in a Session scope shopping cart. If the user submits an item selection page with data specifying sage green size 36 shorts, and then resubmits the item specifying sea blue size 34 shorts, the application can end up with a mixture of information from the two orders, such as sage green size 34 shorts.
By placing the code that sets all of the related session variables in a single `cflock` tag, you ensure that all the variables get set together. In other words, setting all of the variables becomes an **atomic**, or single, operation. It is like a database transaction, where everything in the transaction happens, or nothing happens. In this example, the order details for the first order all get set, and then they are replaced with the details from the second order.
For more examples of using locking in applications, see Examples of cflock.

Using the cflock tag with write-once variables

You need not use `cflock` when you read a variable or call a user-defined function name in the Session, Application, or Server scope if it is set in *only one place* in the application, and is only read (or called, for a UDF) everywhere else. Such data is called *write-once*. If you set an Application or Session scope variable in

© 2014 Adobe Systems Incorporated. All rights reserved.

Application.cfm and never set it on any other pages, lock the code that sets the variable, but do not have to lock code on other pages that reads the variable's value. If you set the variable in the corresponding start method in Application.cfc (for example, `onApplicationStart` for Application scope variables), you do not have to lock the code that sets the variable.
However, although leaving code that uses write-once data unlocked can improve application performance, it also has risks. Ensure that the variables are written only once. For example, ensure that the variable is not rewritten if the user refreshes the browser or clicks a back button. Also, it can be difficult to ensure that you, or future developers, do not later set the variable in more than one place in the application.

Using the cflock tag

The `cflock` tag ensures that concurrently executing requests do not run the same section of code simultaneously and thus manipulate shared data structures, files, or CFX tags inconsistently. It is important to remember that `cflock` protects code sections that access or set data, *not* the variables themselves.
You protect access to code by surrounding it in a `cflock` tag; for example:

```
<cflock scope="Application" timeout="10" type="Exclusive">
 <cfif not IsDefined("Application.number")>
  <cfset Application.number = 1>
 </cfif>
</cflock>
```

Lock types

The `cflock` tag offers two modes of locking, specified by the `type` attribute:

- **Exclusive locks (the default lock type)** Allow only one request to process the locked code. No other requests can run code inside the tag while a request has an exclusive lock.Enclose all code that creates or modifies session, application, or server variables in exclusive `cflock` tags.
- **Read-only locks** Allow multiple requests to execute concurrently if no exclusive locks with the same scope or name are executing. No requests can run code inside the tag while a request has an exclusive lock.Enclose code that only reads or tests session, application, or server variables in read-only `cflock` tags. You specify a read-only lock by setting the `type="readOnly"` attribute in the `cflock`tag, for example:

  ```
  <cflock scope="Application" timeout="10" type="readOnly">
   <cfif IsDefined("Application.dailyMessage")>
    <cfoutput>#Application.dailyMessage#<br></cfoutput>
   </cfif>
  </cflock>
  ```

  Although ColdFusion does not prevent you from setting shared variables inside read-only lock tag, doing so loses the advantages of locking. As a result, be careful not to set any session, application, or server variables inside a read-only `cflock`tag body.

© 2014 Adobe Systems Incorporated. All rights reserved.

> **Note**
>
> You cannot upgrade or downgrade a lock from one type to another. In other words, do not nest an exclusive lock in a read-only lock of the same name or scope; the exclusive lock will always time out. Also, do not nest a read-only lock inside an exclusive lock with the same name or scope; doing so has no effect.

Lock scopes and names

The `cflock` tag prevents simultaneous access to sections of code, not to variables. If you have two sections of code that access the same variable, they must be synchronized to prevent them from running simultaneously. You do this by identifying the locks with the same `scope` or `name` attributes.

> **Note**
>
> ColdFusion does not require you to identify exclusive locks. If you omit the identifier, the lock is anonymous and you cannot synchronize the code in the `cflock` tag block with any other code. Anonymous locks do not cause errors when they protect a resource that is used in a single code block, but they are bad programming practice. You must always identify read-only locks.

Controlling access to data with the scope attribute

When the code that you are locking accesses session, application, or server variables, synchronize access by using the `cflock scope` attribute.
You can set the attribute to any of the following values:

Scope	Meaning
Server	All code sections with this attribute on the server share a single lock.
Application	All code sections with this attribute in the same application share a single lock.
Session	All code sections with this attribute that run in the same session of an application share a single lock.
Request	All code sections with this attribute that run in the same request share a single lock. You use this scope only if your application uses the `cfthread` tag to create multiple threads in a single request. Locking the Request scope also locks access to Variables scope data. For more information on locking the Request scope, see Locking thread data and resource access.

If multiple code sections share a lock, the following rules apply:

- When code is running in a `cflock` tag block with the `type` attribute set to `Exclusive`, code in `cflock` tag blocks with the same `scope` attribute is not allowed to run. They wait until the code with the exclusive lock completes.

© 2014 Adobe Systems Incorporated. All rights reserved.

- When code in a `cflock` tag block with the type `readOnly` is running, code in other `cflock` tag blocks with the same `scope` attribute and the `readOnly` `type` attribute can run, but any blocks with the same `scope` att ribute and an `Exclusive` type cannot run and must wait until all code with the read-only lock completes. However, if a read-only lock is active and code with an exclusive lock with the same scope or name is waiting to execute, read-only requests using the same scope or name that are made after the exclusive request is queued must wait until code with the exclusive lock executes and completes.

Controlling locking access to files and CFX tags with the name attribute

The `cflock` `name` attribute provides a second way to identify locks. Use this attribute when you use locks to protect code that manges file access or calls non-thread-safe CFX code.
When you use the `name` attribute, specify the same name for each section of code that accesses a specific file or a specific CFX tag.

Controlling and minimizing lock time-outs

Include a `timeout` attribute in your `cflock` tag. The `timeout` attribute specifies the maximum time, in seconds, to wait to obtain the lock if it is not available. By default, if the lock does not become available within the time-out period, ColdFusion generates a Lock type exception error, which you can handle using `cftry` and `cfcatch` tags.
If you set the `cflock` `throwOnTimeout` attribute to No, processing continues after the time-out at the line after the `</cflock>` end tag. Code in the `cflock` tag body does not run if the time-out occurs before ColdFusion can acquire the lock. Therefore, never use the `throwOnTimeout` attribute for CFML that must run.
Normally, it does not take more than a few seconds to obtain a lock. Very large time-outs can block request threads for long periods of time and radically decrease throughput. Always use the smallest time-out value that does not result in a significant number of time-outs.
To prevent unnecessary time-outs, lock the minimum amount of code possible. Whenever possible, lock only code that sets or reads variables, not business logic or database queries. One useful technique is to do the following:

1. Perform a time-consuming activity outside a `cflock` tag
2. Assign the result to a Variables scope variable
3. Assign the Variables scope variable's value to a shared scope variable inside a `cflock` block.
   For example, if you want to assign the results of a query to a session variable, first get the query results using a Variables scope variable in unlocked code. Then, assign the query results to a session variable inside a locked code section. The following code shows this technique:

```
<cfquery name="Variables.qUser" datasource="#request.dsn#">
 SELECT FirstName, LastName
 FROM Users
 WHERE UserID = #request.UserID#
</cfquery>
<cflock scope="Session" timeout="5" type="exclusive">
 <cfset Session.qUser = Variables.qUser>
</cflock>
```

Considering lock granularity

When you design your locking strategy, consider whether you should have multiple locks containing small amounts of code or few locks with larger blocks of code. There is no simple rule for making such a decision, and you might do performance testing with different options to help make your decision. However, consider the following issues:

- If the code block is larger, ColdFusion spends more time inside the block, which can increase the number of times an application waits for the lock to released.
- Each lock requires processor time. The more locks you have, the more processor time is spent on locking

© 2014 Adobe Systems Incorporated. All rights reserved.

code.

Nesting locks and avoiding deadlocks

Inconsistent nesting of `cflock` tags and inconsistent naming of locks can cause deadlocks (blocked code). If you are nesting locks, you must consistently nest `cflock` tags in the same order and use consistent lock scopes (or names).
A *deadlock* is a state in which no request can execute the locked section of the page. All requests to the protected section of the page are blocked until there is a time-out. The following table shows one scenario that would cause a deadlock:

User 1	User 2
Locks the Session scope.	Locks the Application scope.
Tries to lock the Application scope, but the Application scope is already locked by User 2.	Tries to lock the Session scope, but the Session scope is already locked by User 1.

Neither user's request can proceed, because it is waiting for the other to complete. The two are deadlocked.
Once a deadlock occurs, neither of the users can do anything to break the deadlock, because the execution of their requests is blocked until the deadlock is resolved by a lock time-out.
You can also cause deadlocks if you nest locks of different types. An example of this is nesting an exclusive lock inside a read-only lock of the same scope or same name.
To avoid a deadlock, lock code sections in a well-specified order, and name the locks consistently. In particular, to lock access to the Server, Application, and Session scopes, do so in the following 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**
>
> You can skip any pair of lock and unlock steps in the preceding list if you do not need to lock a particular scope. For example, you can omit steps 3 and 4 if you do not need to lock the Server scope.

Copying shared variables into the Request scope

You can avoid locking some shared-scope variables multiple times during a request by doing the following:

1. Copy the shared-scope variables into the Request scope in code with an exclusive lock in the Application.cfc `onRequestStart` method or the Application.cfm page.
2. Use the Request scope variables on your ColdFusion pages for the duration of the request.
3. Copy the variables back to the shared scope in code with an exclusive lock in the Application.cfc `onRequestEnd` method on the OnRequestEnd.cfm page.
   With this technique the "last request wins." For example, if two requests run simultaneously, and both requests change the values of data that was copied from the shared scope, the data from the last request to finish is saved in the shared scope, and the data from the previous request is not saved.

Locking application variables efficiently

© 2014 Adobe Systems Incorporated. All rights reserved.

The need to lock application variables can reduce server performance, because all requests that use Application scope variables must wait on a single lock. This issue is a problem even for write-once read-many variables, because you still must ensure that the variable exists, and possibly set the value before you can read it.
You can minimize this problem by using a technique such as the following to test for the existence of application variables and set them if they do not exist:

1. Use an Application scope flag variable to indicate if the variable or variables are initialized. In a read-only lock, check for the existence of the flag, and assign the result to a local variable.
2. Outside the `cflock` bock, test the value of the local variable
3. If it the local variable indicates that the application variables are not initialized, get an exclusive Application scope lock.
4. Inside the lock, again test the Application scope flag, to make sure that another page has not set the variables between step one and step four.
5. If the variables are still not set, set them and set the Application scope flag to true.
6. Release the exclusive lock.
   The following code shows this technique:

```
<!--- Initialize local flag to false. --->
<cfset app_is_initialized = False>
<!--- Get a readonly lock --->
<cflock scope="application" type="readonly">
 <!--- read init flag and store it in local variable --->
 <cfset app_is_initialized = IsDefined("APPLICATION.initialized")>
</cflock>
<!--- Check the local flag --->
<cfif not app_is_initialized >
<!--- Not initialized yet, get exclusive lock to write scope --->
 <cflock scope="application" type="exclusive">
  <!--- Check nonlocal flag since multiple requests could get to the
    exclusive lock --->
  <cfif not IsDefined("APPLICATION.initialized") >
   <!--- Do initializations --->
   <cfset APPLICATION.varible1 = someValue >
    ...
   <!--- Set the Application scope initialization flag --->
   <cfset APPLICATION.initialized = "yes">
  </cfif>
 </cflock>
</cfif>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

Examples of cflock

The following examples show how to use `cflock` blocks in a variety of situations.

Example with application, server, and session variables

This example shows how you can use `cflock` to guarantee the consistency of data updates to variables in the Application, Server, and Session scopes.
This example does not handle exceptions that arise if a lock times out. As a result, users see the default exception error page on lock time-outs.
The following sample code might be part of the Application.cfm file:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cfapplication name="ETurtle"
 sessiontimeout=#createtimespan(0,1,30,0)#
 sessionmanagement="yes">

<!--- Initialize the Session and Application
variables that will be used by E-Turtleneck. Use
the Session lock scope for the session variables. --->

<cflock scope="Session"
 timeout="10" 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 lock for the Application.number variable.
This variable keeps track of the total number of turtlenecks sold.
The following code implements the scheme shown in the Locking Application
variables effectively section --->

<cfset app_is_initialized = "no">
<cflock scope="Application" type="readonly">
 <cfset app_is_initialized = IsDefined("Application.initialized")>
</cflock>
<cfif not app_is_initialized >
 <cflock scope="application" timeout="10" type="exclusive">
  <cfif not IsDefined("Application.initialized") >
   <cfset Application.number = 1>
   <cfset Application.initialized = "yes">
  </cfif>
 </cflock>
</cfif>

<!--- Always display the number of turtlenecks sold --->

<cflock scope="Application"
 timeout="10"
 type ="ReadOnly">
 <cfoutput>
 E-Turtleneck is proud to say that we have sold
 #Application.number# turtlenecks to date.
 </cfoutput>
</cflock>
```

The remaining sample code could appear inside the application page where customers place orders:

```
<html>
<head>
<title>cflock Example</title>
</head>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<body>
<h3>cflock Example</h3>

<cfif IsDefined("Form.submit")>

<!--- Lock session variables --->
<!--- Note that we use the automatically generated Session
 ID as the order ID --->
<cflock scope="Session"
 timeout="10" type="ReadOnly">
 <cfoutput>Thank you for shopping E-Turtleneck.
 Today you have chosen a turtleneck in size
 <b>#form.size#</b> and in the color <b>#form.color#</b>.
 Your order ID is #Session.sessionID#.
 </cfoutput>
</cflock>

<!--- Lock session variables to assign form values to them. --->

<cflock scope="Session"
 timeout="10"
 type="Exclusive">
 <cfparam name=Session.size default=#form.size#>
 <cfparam name=Session.color default=#form.color#>
</cflock>
<
!--- Lock the Application scope variable application.number to
update the total number of turtlenecks sold. --->

<cflock scope="Application"
 timeout="30" type="Exclusive">
 <cfset application.number=application.number + 1>
</cflock>

<!--- Show the form only if it has not been submitted. --->
<cfelse>
<form action="cflock.cfm" method="Post">

<p> Congratulations! You have just selected
the longest-wearing, most comfortable turtleneck
in the world. Please indicate the color and size
you want to buy.</p>

<table cellspacing="2" cellpadding="2" border="0">
<tr>
<td>Select a color.</td>
<td><select type="Text" name="color">
  <option>red
  <option>white
  <option>blue
  <option>turquoise
  <option>black
  <option>forest green
  </select>
 </td>
</tr>
<tr>
 <td>Select a size.</td>
 <td><select type="Text" name="size">
```

© 2014 Adobe Systems Incorporated. All rights reserved.

```
    <option>small
    <option>medium
    <option>large
    <option>xlarge
    </select>
   </td>
  </tr>
  <tr>
   <td></td>
   <td><input type="Submit" name="submit" value="Submit">
   </td>
  </tr>
  </table>
  </form>
  </cfif>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

```
</body>
</html>
```

Note

In this simple example, the Application.cfm page displays the Application.number variable value. Because the Application.cfm file is processed before any code on each ColdFusion page, the number that displays after you click the submit button does not include the new order. One way you can resolve this problem is by using the OnRequestEnd.cfm page to display the value at the bottom of each page in the application.

Example of synchronizing access to a file system

The following example shows how to use a `cflock` block to synchronize access to a file system. The `cflock` tag protects a `cffile` tag from attempting to append data to a file already open for writing by the same tag executing on another request.
If an append operation takes more than 30 seconds, a request waiting to obtain an exclusive lock to this code might time out. Also, this example uses a dynamic value for the `name` attribute so that a different lock controls access to each file. As a result, locking access to one file does not delay access to any other file.

```
<cflock name=#filename# timeout=30 type="Exclusive">
 <cffile action="Append"
  file=#fileName#
  output=#textToAppend#>
</cflock>
```

Example of protecting ColdFusion extensions

The following example shows how you can build a custom tag wrapper around a CFX tag that is not thread-safe. The wrapper forwards attributes to the non-thread-safe CFX tag that is used inside a `cflock` tag.

```
<cfparam name="Attributes.AttributeOne" default="">
<cfparam name="Attributes.AttributeTwo" default="">
<cfparam name="Attributes.AttributeThree" default="">

<cflock timeout=5
  type="Exclusive"
  name="cfx_not_thread_safe">
 <cfx_not_thread_safe attributeone=#attributes.attributeone#
  attributetwo=#attributes.attributetwo#
  attributethree=#attributes.attributethree#>
</cflock>
```

#back to top

© 2014 Adobe Systems Incorporated. All rights reserved.

Using ColdFusion Threads

You can use threads in Adobe ColdFusion to simultaneously run multiple streams of execution in a ColdFusion page or CFC.

- About ColdFusion threads
- Creating and managing ColdFusion threads
- Using thread data
- Working with threads
- Using ColdFusion tools to control thread use
- Example: getting multiple RSS feeds

© 2014 Adobe Systems Incorporated. All rights reserved.

About ColdFusion threads

Threads are independent streams of execution. Multiple threads on a page or CFC can execute simultaneously and asynchronously, letting you perform asynchronous processing in CFML.
Threads are useful for two broad types of activities:

- When multiple actions can occur simultaneously
- When you do not have to wait for one action to complete before starting the next action
  Some typical uses for threads include the following examples:

- An application that aggregates information from multiple external sources, that takes significant time to respond, has the code that gets information from each source in a separate thread. This way, the application starts all requests quickly and has to wait only until the last response is received, instead of having to wait for a response to each request before the next request can start. One example of such usage is an RSS or Atom feed aggregator.
- A page that sends many mail messages runs the code that sends the mail messages in a separate thread and doesn't wait for it to complete to continue processing. The thread that sends the mail messages continues processing after the page-level processing is completed and the application starts processing another page.
- An application does maintenance of user data, such as using update queries, deleting records, and so on, whenever a user logs into the site. If the application does the maintenance in a separate thread, the user gets an immediate response after logging in, without having to wait for the updates to complete.
  When ColdFusion processes a page, the page executes in a single thread, called the page thread. The `cfthread` tag lets you create additional threads that can process independently of the page thread, and lets you synchronize thread processing, for example, by having the page thread wait until threads that you create complete their processing.

© 2014 Adobe Systems Incorporated. All rights reserved.

Creating and managing ColdFusion threads

You use the `cfthread` tag and the `Sleep` function to create and manage ColdFusion threads. You manage a thread by doing the following actions:

- Start the thread running.
- Temporarily suspend the processing of the thread. This action is useful if one thread must wait for another thread to do processing, but both threads must continue processing without joining.
- End a thread. You typically end a running thread if there is an error, or if it is still processing after a long time.
- Have the page or a thread wait until one or more other threads have completed processing before proceeding with its processing, called joining the threads. You typically join threads when one thread requires the results from another thread. For example, if a page uses multiple threads to get several news feeds for display, it joins all the feed threads before it displays the results.
  Each thread runs the code inside a `cfthread` tag body and normally exits when the tag body code completes processing.

Starting a thread

You start a thread by using a `cfthread` tag with an `action` attribute value of `run`. CFML code within the `cfthread` tag body executes on a separate thread while the page request thread continues processing. Only the page thread can create other threads. A thread that you create with a `cfthread` tag cannot create a child thread, so you cannot have multiple nested threads.
Optionally, when you start the thread, you can specify a priority level of high, normal (the default), or low to specify the relative amount of time that the processor should devote to the thread. Page-level code always runs at normal priority, so you can give your threads more or less processing time than the page.
For more information on using thread attributes, see *The Attributes scope and thread attributes* in Using thread data.

Suspending a thread

In some cases, one thread must wait until a second thread completes some operations, but should not wait until the second thread completes all processing, so you cannot just join the threads. For example, one thread might do initialization that multiple threads require, and then it might continue with additional processing. The other threads could suspend themselves until initialization is complete.
The `Sleep` function and `cfthread` tag with a `sleep` `action` attribute provide two equivalent mechanisms for doing such synchronization. They suspend the thread processing for a specified period of time. A code loop could test a condition variable and sleep for a period before retesting the condition. When the condition is true (or a value is reached, or some other test is valid), the program exits the loop and the thread continues processing.
The following example shows how one thread could use a sleep function to wait for a second thread to perform some actions.

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<!--- ThreadA loops to simulate an activity that might take time. --->
<cfthread name="threadA" action="run">
  <cfset thread.j=1>
  <cfloop index="i" from="1" to="1000000">
   <cfset thread.j=thread.j+1>
 </cfloop>
</cfthread>

<!--- ThreadB loops, waiting until threadA finishes looping 40000 times.
   the loop code sleeps 1/2 second each time. --->
<cfthread name="threadB" action="run">
 <cfscript>
  thread.sleepTimes=0;
  thread.initialized=false;
  while ((threadA.Status != "TERMINATED") && (threadA.j < 400000)) {
   sleep(500);
   thread.sleeptimes++;
  }
  // Don't continue processing if threadA terminated abnormally.
  If (threadA.Status != "TERMINATED") {
   thread.initialized=true;
   // Do additional processing here.
  }
 </cfscript>
</cfthread>

<!Join the page thread to thread B. Don't join to thread A.--->
<cfthread action="join" name="threadB" timeout="10000" />

<!--- Display the thread information. --->
<cfoutput>
 current threadA index value: #threadA.j#<br />
 threadA status: #threadA.Status#<br>
 threadB status: #threadB.Status#<br>
 threadB sleepTimes: #threadB.sleepTimes#<br>
 Is threadB initialized: #threadB.initialized#<br>
</cfoutput>
```

Ending a thread

If a thread never completes processing (is hung), it continues to occupy system resources, so it is good practice to have your application check for hung threads and end them. Also consider ending threads that take excessive time to process and might significantly reduce the responsiveness of your application or server.
To end a thread, use the `cfthread` tag with an `action` attribute value of `terminate`, as the following code snippet shows.

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<!--- Thread1 sleeps to simulate an activity that might hang. --->
<cfthread name="thread1" action="run">
  <cfset thread.j=1>
 <cfset sleep(50000) >
 </cfthread>

<!--- Thread2 loops to simulate an activity that takes less time. --->
<cfthread name="thread2" action="run">
  <cfset thread.j=1>
  <cfloop index="i" from="1" to="10">
   <cfset thread.j=thread.j+1>
 </cfloop>
</cfthread>

<!--- The page thread sleeps for 1/2 second to let thread
   processing complete. --->
<cfset sleep(500) >

<!--- The page thread loops through the threads and terminates
   any that are still running or never started.
  Note the use of the cfthread scope and associative array
  notation to reference the dynamically named threads without
  using the Evaluate function. --->
<cfloop index="k" from="1" to="2">
<cfset theThread=cfthread["thread#k#"]>
 <cfif ((theThread.Status IS "RUNNING") || (theThread.Status IS "NOT_STARTED"))>
  <cfthread action="terminate" name="thread#k#" />
 </cfif>
</cfloop>

<!--- Wait 1/2 second to make ensure the termination completes --->
<cfset sleep(500) >

<!--- Display the thread information. --->
<cfoutput>
 thread1 index value: #thread1.j#<br />
 thread1 status: #thread1.Status#<br>
 thread2 index value: #thread2.j#<br />
 thread2 status: #thread2.Status#<br>
</cfoutput>
```

Note

You can also have the ColdFusion Sever Monitor automatically check for and terminate hung threads.

Joining threads

You use the `cfthread tag` with an `action` attribute value of `join` to join two or more threads. You join threads when one thread depends on one or more other threads completing before it can do some processing. For example, a page can start multiple threads to do processing and join them before it processes the thread results. By default, the `join` action stops the current thread from doing further processing until all the specified threads complete processing.

© 2014 Adobe Systems Incorporated. All rights reserved.

You can use a `timeout` attribute to specify the number of milliseconds that the current thread waits for the thread or threads being joined to finish. If any thread does not finish by the specified time, the current thread proceeds without waiting for the remaining thread or threads to complete.
The following code, for example, joins three threads to the current thread (often, the main page thread). The current thread waits up to six seconds for the other threads to complete, and continues processing if one or more threads do not complete by then.

```
<cfthread action="join" name="t1,t2,t3" timeout="6000"/>
```

If the `timeout` attribute value is 0, the default value, the current thread continues waiting until all joining threads finish. In this case, if the current thread is the page thread, the page continues waiting until the threads are joined, even if you specify a page time-out. As a general rule, specify a `timeout` value to limit hung threads.

© 2014 Adobe Systems Incorporated. All rights reserved.

Using thread data

Because multiple threads can process simultaneously within a single request, applications must ensure that data from one thread does not improperly affect data in another thread. ColdFusion provides several scopes that you can use to manage thread data, and a request-level lock mechanism that you use to prevent problems caused by threads that access page-level data. ColdFusion also provides metadata variables that contain any thread-specific output and information about the thread, such as its status and processing time.

Thread scopes

Each thread has three special scopes:

- The thread-local scope
- The Thread scope
- The Attributes scope

The thread-local scope

The thread-local scope is an implicit scope that contains variables that are available only to the thread, and exist only for the life of the thread. Any variable that you define inside the `cfthread` tag body without specifying a scope name prefix is in the thread local scope and cannot be accessed or modified by other threads.
To create a thread-local variable, assign the variable in the `cfthread` tag body without specifying a scope prefix, as in the following lines:

```
<cfset var index=1>
<cfset index=1>
```

These two lines are equivalent, with one exception: If you use the `var` keyword, the assignment code *must* immedia tely follow the `cfthread` tag, before any other CFML tags.

The Thread scope

The Thread scope contains thread-specific variables and metadata about the thread. Only the owning thread can write data to this scope, but the page thread and all other threads in a request can read the variable values in this scope. Thread scope data remains available until the page and all threads that started from the page finish, even if the page finishes before the threads complete processing.
To create a Thread scope variable, in the `cfthread` tag body, use the keyword `Thread` or the name of the thread (for example, `myThread`) as a prefix. the following examples of creating a Thread scope variable are equivalent:

```
<cfset Thread.myValue = 27>
<cfset myThread.myValue = 27>
```

To access a thread's Thread scope variables outside the thread, prefix the variable with the thread's name, as in the following example:

```
<cfset nextValue=myThread.myValue + 1>
```

Thread scope variables are only available to the page that created the thread or to other threads created by that

© 2014 Adobe Systems Incorporated. All rights reserved.

page. No other page can access the data. If one page must access another page's Thread scope data, you must place the data in a database or file and access it from there.

The Thread scope of each thread is a subscope of a special scope, cfthread, that lasts as long as the request, or until the last thread that it starts completes, whichever is longer. Thus, if you have two threads, myThread1 and myThread2, you can access their Thread scopes as cfthread.myThread1 and cfthread.myThread2 until all threads and the request complete. In most cases, there is no need to use the `cfthread` scope directly. However, you can use the cfthread scope name in either of the following situations:

1. If you generate the thread name dynamically, you can avoid using the `Evaluate` function by using the `cfthreadscope` with associative array notation, as the following code snippet shows:

```
<cfset threadname="thread_#N#">
...
<!--- The following two lines are equivalent --->
<cfset threadscopeForNthThread = cfthread[threadname] >
<cfset threadscopeForNthThread = Evaluate(threadname) >
```

1. If you have a thread with the same name as a Variables scope variable, you can access that thread's Thread scope only by prefacing the Thread name with `cfthread`. Otherwise, you access the Variables scope variable, or get an error.

The Attributes scope and thread attributes

The Attributes scope contains attributes that are passed to the thread, either individually or in the `attributeCollection` attribute. The Attributes scope is available only within the thread and only for the life of the thread. ColdFusion makes a complete (deep) copy of all the attribute variables before passing them to the thread; therefore, the values of the variables inside the thread are independent of the values of any corresponding variables in other threads, including the page thread. For example, if you pass a CFC instance as an attribute to a thread, the thread gets a complete new copy of the CFC, including the contents of its This scope at the time that you create the thread. Any changes made to the original CFC outside the thread, for example, by calling a CFC function, have no effect on the copy that is in the thread. Similarly, any changes to the CFC instance in the thread have no effect on the original CFC instance.

Copying the data ensures that the values passed to threads are thread-safe, because the attribute values cannot be changed by any other thread. If you do not want duplicate data, do not pass it to the thread as an attribute or in the `attributeCollection` attribute. Instead, keep the data in a scope that the thread can access. An example of an object that should not be passed to the thread as an attribute is a singleton CFC that should never be duplicated. The singleton CFC must be kept in some shared scope and accessed by threads. For more information, see the *Using other scopes* in Using thread data.

Because ColdFusion copies all attributes by value, you can have multiple threads, for example, threads created dynamically in a loop, that use the same attribute names, but where each thread gets a different value, as shown in the following code excerpt, which creates separate threads to copy each of several files in a directory:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cfloop query="dir">
 <cfset threadname = "thread_" & #i#>
 <cfset i=i+1>
 <cfthread name="#threadname#" filename="#dir.name#">
  <cffile action="COPY" source="#src#\#filename#"
   destination="#dest#\#filename#\">
 </cfthread>
</cfloop>
```

Using other scopes

Threads have access to all the ColdFusion scopes. All the threads run by a page share the same Variables and This scope. All the threads run in a request share the same Form, URL, Request, CGI, Cookie, Session, Application, Server and Client scopes. Be careful to lock access to these scopes if more than one thread could try to modify the data in the scopes; otherwise you can get deadlocks between threads. For more information, see *Locking thread data and resource access* in Using thread data.
Although a thread can access all the scopes, it might not be able to write to scopes like Session, Cookie, or Request after the request page processing completes.

Scope precedence

If you do not specify a scope prefix on a variable inside a `cfthread` tag body, ColdFusion checks scopes in the following order to find the variable:

1. Function-local, in function definitions in the thread only
2. Thread-local
3. Attributes
4. Variables
5. Thread/cfthread
   Other scopes are checked in the standard scope checking order.

Locking thread data and resource access

When an application uses multiple threads, be careful to ensure that the threads do not simultaneously attempt to use or modify shared resources that are not themselves thread-safe, including the following items:

- If multiple threads modify a Variables or Request scope variable, use a Request scope lock to control access to the code that uses the variable to prevent deadlocks and race conditions. Similarly, use a Request scope lock around code that accesses built-in data structures or subscopes of the Variables scope, such as the Forms variable, that you change in multiple threads.
- Multiple threads should not try to access any other shared resource simultaneously. For example, do not use the same FTP connection from multiple threads. To prevent this behavior, place the code that uses the resource in named `cflock` tags. Use the same `name` attribute for all `cflock` tags around code that uses a specific resource.
  For more information on locking code, see `cflock` and Locking code with cflock__.

Metadata variables

The Thread scope contains the following variables that provide information about the thread, called metadata.

Variable	Description

© 2014 Adobe Systems Incorporated. All rights reserved.

Elapsedtime	The amount of processor time that has been spent handling the thread.
Error	A ColdFusion error structure that contains the keys that you can access in a `cfcatch` tag. This variable has a value only if an unhandled error occurred during thread processing. For information on handling thread errors, see *Handling ColdFusion thread errors* in Working with threads.
Name	The thread name.
Output	Output text that the thread generates. Threads cannot display output directly. For more information see *Handling thread output* in Working with threads.
Priority	The thread processing priority, as specified when you created the thread.
Starttime	The time at which the thread began processing.
Status	The current status of the thread. For information on using the Status in an application, see *Using the thread status* in Working with threads.

As with other variables in the Thread scope, thread metadata is available to all of a page's threads by specifying the thread name as a variable prefix. For example, the page thread can get the current elapsed time of the myThread1 thread from the `myThread1.ElapsedTime` variable.
The metadata is available from the time that you create the thread until the time when the page and all threads started on the page complete processing, even if the page finishes before the threads finish. This way, you can get thread output, error information, and processing information during and after the time when the thread is processing.

© 2014 Adobe Systems Incorporated. All rights reserved.

Working with threads

Multi-threaded applications use several building blocks, including the following:

- Starting threads in loops
- Getting information about the thread processing status
- Displaying thread results
- Handling thread errors
- Using database transactions with threads

Starting threads inside loops

Because threads run asynchronously, page level variables can change during thread execution. As a result of this behavior, if you start threads inside a `cfloop`, and code inside the threads uses the value of the loop iterator (like the index variable, query name, list item), pass the loop iterator to the thread as an attribute.
The following example shows the use of threads inside a loop. It uses an indexed `cfloop` tag to start five threads. Each thread gets the current loop index value in a `threadIndex` attribute. The thread adds an array entry with the `threadIndex` attribute value of the thread and the current value of the page `cfloop` index, `pageIndex`. After joining the threads, the page displays the array contents. When you run the example, particularly if you run it multiple times, you see that at the time the thread saves data to the array, the value of `pageIndex` has incremented past the `threadIndex` value, and multiple threads often have the same `pageIndex` value; but the multiple threads always have the correct `threadIndex` value.

```
<cfloop index="pageIndex" from="1" to="5">
 <cfthread name="thr#pageIndex#" threadIndex="#pageIndex#" action="run">
  <cfset Variables.theOutput[threadIndex]="Thread index attribute:" &
   threadIndex & "   Page index value: " & pageIndex>
 </cfthread>
</cfloop>

<cfthread action="join" name="thr1,thr2,thr3,thr4,thr5" timeout=2000/>

<cfloop index="j" from="1" to="5">
 <cfoutput>#theOutput[j]# <br /></cfoutput>
</cfloop>
```

Using the thread status

The Thread scope `status` metadata variable lets the page, or any other thread started by the page, determine the status of any thread. The page processing code can then take a necessary action, for example, if the thread has terminated abnormally or has hung. The `status` variable can have the following values:

Value	Meaning
NOT_STARTED	The thread has been queued but is not processing yet.
RUNNNG	The thread is running normally.

© 2014 Adobe Systems Incorporated. All rights reserved.

TERMINATED	The thread stopped running as a result of one of the following actions: • A `cfthread` tag with a `terminate` action stopped the thread. • An error occurred in the thread that caused it to terminate. • A ColdFusion administrator stopped the thread from the Server Monitor.
COMPLETED	The thread ended normally.
WAITING	The thread has run a `cfthread` tag with `action="join"`, and one or more of the threads being joined have not yet completed.

Applications can check the thread status to manage processing. For example, an application that requires results from a thread specifies a time-out when it joins the thread; in this case, it can check for the COMPLETED status to ensure that the thread has completed processing and the join did not just result from a time-out. Similarly, an application can check the status value of threads that might not start or might not complete normally, and terminate it if necessary. The example in *Ending a thread* in Creating and managing ColdFusion threadschecks thread status and terminates any threads with RUNNING or NOT_STARTED status.

Handling thread output

To prevent conflicts, only the page thread displays output. Therefore, named threads have the following limitations:

- ColdFusion places all output that you generate inside a thread, such as HTML and plain text, or the generated output of a `cfoutput` tag, in the Thread scope `output` metadata variable. The page-level code can display the contents of this variable by accessing the *threadName*`.output` variable.
- All tags and tag actions that directly send output to the client (instead of generating page text such as HTML output), do not work inside the thread. For example, to use the `cfdocument` or `cfreport` tags in a thread, specify a `filename` attribute; to use a `cfpresentation` tag, use a `directory` attribute.

Handling ColdFusion thread errors

If an error occurs in a thread, page-level processing is not affected, and ColdFusion does not generate an error message. If you do not handle the error by using a try/catch block in the thread code, the thread with the error terminates and the page-level code or other threads can get the error information from the thread metadata `Error` variable and handle the error appropriately.
You cannot use page- or application-based error handling techniques to manage errors that occur during thread execution. For that reason, you cannot use the `cferror` tag or the `onError` application event handler for thread errors. Instead, use either of the following techniques:

1. Use `cftry/cfcatch` tags or `try/catch` CFScript statements in the `cfthread` body to handle the errors inside the thread.
2. Handle the error outside the thread by using the thread error information that is available to the page and other threads in the Thread scope *threadName*`.Error` variable. Application code can check this variable for error information. For example, after you join to a thread that had an error, you could check the *threadname*`.status` variable for a value of `terminated`, which indicates that the thread terminated abnormally. You could then check the *threadName*`.Error` variable for information on the termination cause.

Handling database transactions

© 2014 Adobe Systems Incorporated. All rights reserved.

Database transactions cannot span threads. For example, consider a page with the following structure:

```
<cftransaction>
 <cfthread name ="t1" ...>
  <cfquery name="q1" ...>
   ...
  </cfquery>
 </cfthread>
 <cfquery name="q2" ...>
  ...
 </cfquery>
 <cfthread action="join" name="t1" ... />
</cftransaction>
```

In this case, query `q1` is *not* included in the transaction that contains query `q2`. To include both queries in the transaction, you must place the complete transaction in a single thread, using a structure such as the following:

```
<cfthread name ="t1" ...>
 <cftransaction>
  <cfquery name="q1" ...>
   ...
  </cfquery>
  <cfquery name="q2" ...>
   ...
  </cfquery>
 </cftransaction>
</cfthread>
<cfthread action="join" name="t1" ... />
```

© 2014 Adobe Systems Incorporated. All rights reserved.

Using ColdFusion tools to control thread use

The Tag Limit Settings section of the ColdFusion Administrator Server Settings > Request Tuning page lets you specify a maximum number of `cfthread`-started threads that can run at one time. When ColdFusion reaches this maximum, it queues additional `cfthread` requests and starts the queued threads when running threads end.

© 2014 Adobe Systems Incorporated. All rights reserved.

Example: getting multiple RSS feeds

The following example uses three threads to get the results of three RSS feeds. The user must submit the form with all three feeds specified. The application joins the threads with a time-out of 6 seconds, and displays the feed titles and the individual item titles as links.

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<!--- Run this code if the feed URL form has been submitted. --->
<cfif isDefined("Form.submit")>
 <cfloop index="i" from="1" to="3">
  <!--- Use array notation and string concatenation to create a variable
     for this feed. --->
  <cfset theFeed = Form["Feed"&i]>
  <cfif theFeed NEQ "">
   <!--- Use a separate thread to get each of the feeds. --->
   <cfthread action="run" name="t#i#" feed="#theFeed#">
    <cffeed source="#feed#"
     properties="thread.myProps"
     query="thread.myQuery">
   </cfthread>
  <cfelse>
   <!--- If the user didn't fill all fields, show an error message. --->
   <h3>This example requires three feeds.<br />
   Click the Back button and try again.</h3>
   <cfabort>
  </cfif>
 </cfloop>

 <!--- Join the three threads. Use a 6 second timeout. --->
 <cfthread action="join" name="t1,t2,t3" timeout="6000" />

 <!--- Use a loop to display the results from the feeds. --->
 <cfloop index="i" from="1" to="3">
  <!--- Use the cfthread scope and associative array notation to get the
    Thread scope. --->
  <cfset feedResult=cfthread["t#i#"]>
  <!--- Display feed information only if you got items,
     for example, the feed must complete before the join. --->
  <cfif isDefined("feedResult.myQuery")>
   <cfoutput><h2>#feedResult.myProps.title#</h2></cfoutput>
   <cfoutput query="feedResult.myQuery">
    <p><a href="#RSSLINK#">#TITLE#</a></p>
   </cfoutput>
  </cfif>
 </cfloop>

</cfif>

<!--- The form for entering the feeds to aggregate. --->
<cfform>
 <h3>Enter three RSS Feeds</h3>
 <cfinput type="text" size="100" name="Feed1" validate="url"
  value="http://rss.adobe.com/events.rss?locale=en"><br />
 <cfinput type="text" size="100" name="Feed2" validate="url"
  value="http://weblogs.macromedia.com/dev_center/index.rdf"><br />
 <cfinput type="text" size="100" name="Feed3" validate="url"
  value="http://rss.adobe.com/studio.rss?locale=en"><br />
 <cfinput type="submit" name="submit">
</cfform>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

#back to top

© 2014 Adobe Systems Incorporated. All rights reserved.

Securing Applications

Resource security (Adobe ColdFusion Standard) or *sandbox security* (Adobe ColdFusion Enterprise) restricts access to specific resources, such as tags and files. You use the ColdFusion Administrator to configure sandbox or resource security, and structure an application to take advantage of this security.
User security depends on a user identity. You can implement user security in Adobe ColdFusion applications.
For detailed information on using Administrator-controlled security features, see *Configuring and Administering ColdFusion*.

© 2014 Adobe Systems Incorporated. All rights reserved.

ColdFusion security features

ColdFusion provides scalable, granular security for building and deploying your ColdFusion applications. ColdFusion provides the following types of security resources:

- **Development** ColdFusion Administrator is password-protected. Additionally, you can specify a password for access to data sources from Dreamweaver. For more information on configuring Administrator security passwords, see the ColdFusion Administrator online Help.
- **CFML features**The CFML language includes the following features that you can use to enhance application security.
  - The `cfqueryparam` tag: This tag helps prevent users from injecting malicious SQL expressions. For more information on using this tag for database security, see Enhancing security with cfqueryparam,
  - Scriptprotect setting: You can use this setting to protect against cross-site scripting attacks. However, using Scriptprotect does not ensure complete protection. Set this value with the ColdFusion Administrator Enable Global Script Protection setting, in the Application.cfc This.scriptprotect variable, or in the corresponding `cfapplication` tag `scriptprotect` attribute. For more information on this feature, see cfapplication in the *CFML Reference*. For information on Application.cfc see Defining the application and its event handlers in Application.cfc.
  - Encryption and hashing functions: The `Encrypt`, `Decrypt,` and `Hash` functions let you select a secure algorithm for encrypting and decrypting data or generating a hash "fingerprint." You can select from among several secure algorithms that underlying Java security mechanisms support. For encryption, these include, AES, Blowfish, DES and Triple DES. For more information, see the Encrypt, Decrypt, and Hash, functions in the *CFML Reference*.
  - Data validation tools: ColdFusion includes a variety of tools for validating form input and other data values, including ways to ensure that users do not submit malicious form data. For information on data validation see Validating data; for specific information on security and validation, see *Security considerations* in About ColdFusion validation.

- **Resource/Sandbox** The ColdFusion Administrator can limit access to ColdFusion resources, including selected tags and functions, data sources, files, and host addresses. In the Standard Edition, you configure a single set of resource limitations that apply to all your ColdFusion applications.In the Enterprise Edition, you can have multiple sandboxes, based on the location of your ColdFusion pages, each with its own set of resource limitations. You can confine applications to secure areas, thereby flexibly restricting the access that the application has to resources.
- **User**ColdFusion applications can require users to log in to use application pages. You can assign users to roles (sometimes called groups); ColdFusion pages can determine the logged-in user's roles or ID and selectively determine what to do based on this information. User security is also called authentication and authorization security.

> **Note**
>
> You can also use the `cfencode` utility, located in the cf_root/bin directory, to obscure ColdFusion pages that you distribute. Although this technique cannot prevent persistent hackers from determining the contents of your pages, it does prevent inspection of the pages. The `cfencode` utility is not available on OS X.

© 2014 Adobe Systems Incorporated. All rights reserved.

About resource and sandbox security

ColdFusion provides two levels of resource-based security:

- ColdFusion Standard refers to its resource-based security as resource security. It lets you specify a single set of limitations on access to ColdFusion resources that apply to all ColdFusion applications.
- ColdFusion Enterprise refers to its resource-based security as sandbox security. Sandbox security is a superset of resource security. Sandbox security lets you create multiple *sandboxes*, each** corresponding to a different directory. For each sandbox, you specify a set of resource limitations that apply to all ColdFusion pages in the sandbox directory and its subdirectories. If you create a sandbox that is a subdirectory of a sandbox, the subdirectory's rules override the parent directory's rules.
  The ColdFusion Administrator Resource Security page (in Standard) and Sandbox Security page (in Enterprise) let you enable resource-based security. In ColdFusion Standard, the page lets you configure the resource settings that apply to all your ColdFusion applications. In ColdFusion Enterprise, the page lets you create sandboxes and configure the resource limitations for each sandbox individually.

Resource control

ColdFusion lets you control access to the following resources:

Resource	Description
Data sources	Enables access to specified data sources.
CF tags	Prevents pages from using CFML tags that access external resources. You can prevent pages in the directory from using any or all of the following tags:cfcollection, cfcontent, cfcookie, cfdirectory, cfdocument, cfexecute, cffile, cfftp, cfgridupdate, cfhttp, cfhttpparam, cfindex, cfinsert, cfinvoke, cfldap, cflog, cfmail, cfobject, cfobjectcache, cfpop, cfquery, cfregistry, cfreport, cfschedule, cfsearch, cfstoredproc, cftransaction, cfupdate
CF functions	Prevents pages from using CFML functions that access external resources. You can prevent pages from using any or all of the following functions:CreateObject (COM, Java, Web Service),{{}}DirectoryExists. ExpandPath, FileExists, GetBaseTemplatePath, GetDirectoryFromPath, GetFileFromPath, GetGatewayHelper, GetProfileString, GetTempDirectory, GetTempFile, GetTemplatePath, SendGatewayMessage, SetProfileString
Files/directories	Sets read, write, execute, and delete access to specified directories, directory trees, or files.
Server/ports	Controls access from ColdFusion to IP addresses and port numbers. You can specify host names or numeric addresses, and you can specify individual ports and port ranges.

© 2014 Adobe Systems Incorporated. All rights reserved.

Note

For more information on configuring resource and sandbox security, see Configuring and Administering ColdFusion and the ColdFusion Administrator online Help.

Sandbox security

In ColdFusion Enterprise, sandbox security lets you apply different sets of rules to different directory structures. Use it to partition a shared hosting environment so that a number of applications with different purposes, and possibly different owners, run securely on a single server. When multiple applications share a host, you set up a separate directory structure for each application, and apply rules that let each application access only its own data sources and files.

Sandbox security also lets you structure and partition an application to reflect the access rights that are appropriate to different functional components. For example, if your application has both employee inquiry functions and HR functions that include creating, accessing, and modifying sensitive data, you could structure the application as follows:

- HR pages go in one directory with access rules that enable most activities.
- Employee pages go in another directory whose rules limit the files they modify and the tags they use.
- Pages required for both HR and employee functions go in a third directory with appropriate access rules.

© 2014 Adobe Systems Incorporated. All rights reserved.

About user security

User security lets your application use security rules to determine what it shows. It has two elements:

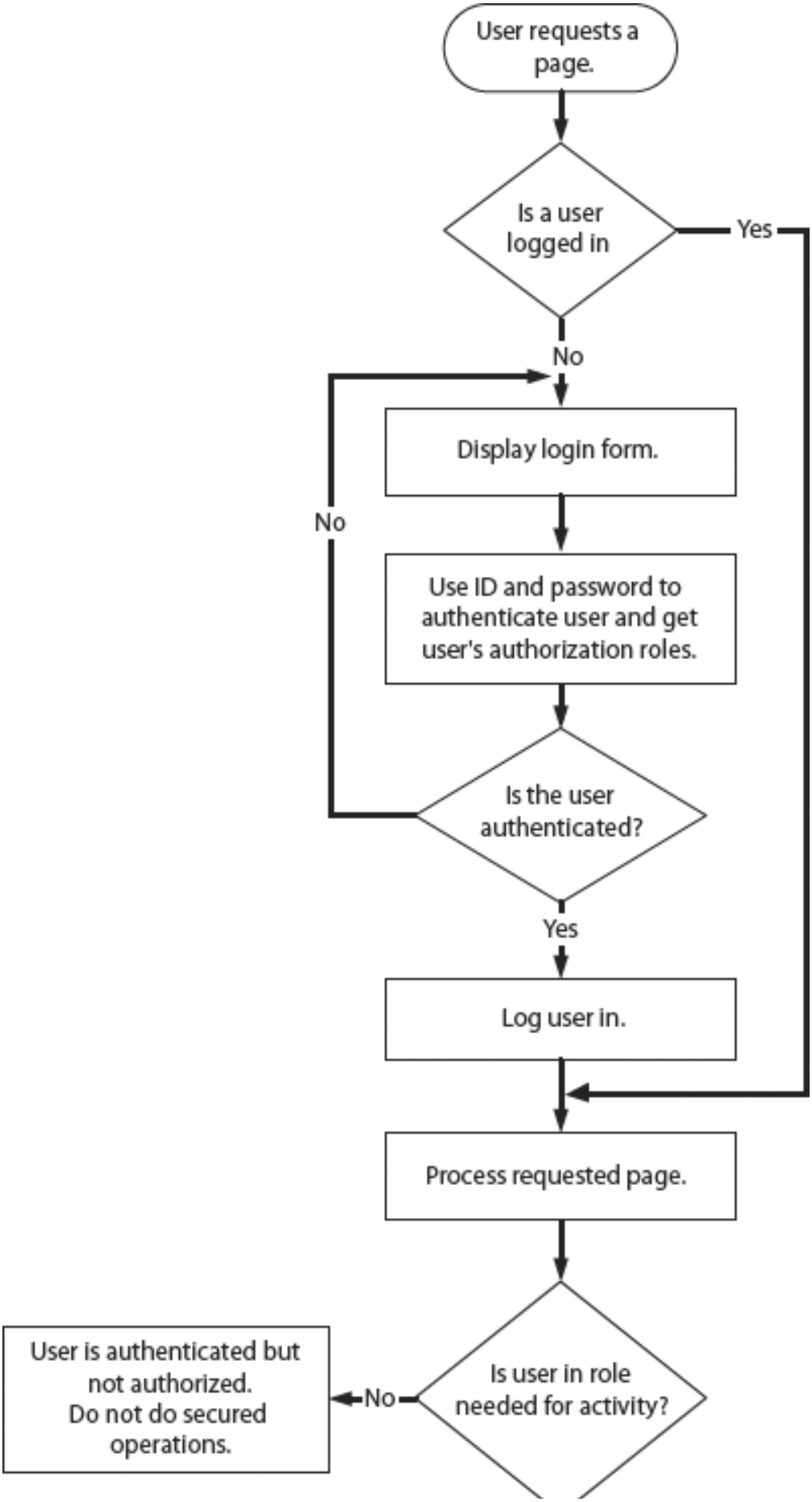
- **Authentication** Ensures that a valid user is logged-in, based on an ID and password provided by the user. ColdFusion (or, in some cases if you use web server authentication, the web server) maintains the user ID information while the user is logged-in.
- **Authorization** Ensures that the logged-in user is allowed to use a page or perform an operation. Authorization is typically based on one or more *roles*(sometimes called groups) to which the user belongs. For example, in an employee database, all users could be members of either the employee role or the contractor role. They could also be members of roles that identify their department, position in the corporate hierarchy, or job description. For example, someone could be a member of some or all of the following roles:
  - Employees
  - Human Resources
  - Benefits
  - Managers

    Roles enable you to control access in your application resources without requiring the application to maintain knowledge about individual users. For example, suppose you use ColdFusion for your company's intranet. The Human Resources department maintains a page on the intranet on which all employees can access timely information about the company, such as the latest company policies, upcoming events, and job postings. You want everyone to be able to read the information, but you want only certain authorized Human Resources employees to be able to add, update, or delete information. Your application gets the user's roles from the user information data store when the user logs in, and then enables access to specific pages or features based on the roles. Typically, you store user information in a database, LDAP directory, or other secure information store.You also use the user ID for authorization. For example, to let employees view customized information about their salaries, job levels, and performance reviews. You certainly would not want one employee to view sensitive information about another employee, but you would want managers to be able to see, and possibly update, information about their direct reports. By employing both user IDs and roles, you ensure that only the appropriate people access or work with sensitive data.The following image shows

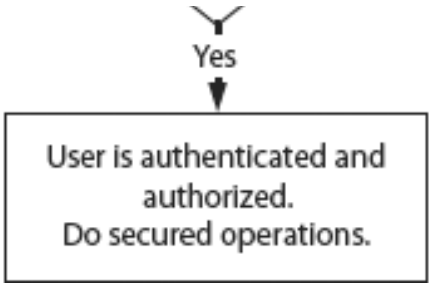
© 2014 Adobe Systems Incorporated. All rights reserved.

a typical flow of control for user authentication and authorization. Following sections expand on this diagram to describe how you implement user security in ColdFusion.

© 2014 Adobe Systems Incorporated. All rights reserved.



© 2014 Adobe Systems Incorporated. All rights reserved.



Authenticating users

Use either, or both, of the following forms of authentication to secure your ColdFusion application:

- Web server authentication, where the web server authenticates the user and does not allow access to the website by users without valid login IDs
- Application authentication, where the ColdFusion application authenticates the user and does not allow access to the application by users without valid login IDs

Web server authentication

All major web servers support basic HTTP authentication. Some web servers also support other authentication methods, including Digest HTTP authentication and Microsoft NTLM authentication.

> **Note**
>
> Dreamweaver and Studio MX do not support NTLM security with RDS. Therefore, you cannot use RDS with these applications if the ColdFusion RDS servlet (cf_root/CFIDE/main/ide.cfm) is in a directory that is protected using NTLM security.

In web server authentication, the web server requires the user to log in to access pages in a particular directory, as follows:

1. When the user first requests a page in the secured directory, the web server notifies the browser that the requested page requires credentials (a user ID and password).Basic HTTP authentication sends the user ID and password in a base64-encoded string with each request. Use SSL (Secure Sockets Layer) for all page transactions, to protect the user ID and password from unauthorized access. For more information on SSL and the keytool utility, see *About LDAP Server Security* in Advanced topics.
2. The browser prompts the user for the credentials.
3. The user supplies the credentials and the browser send the information back to the web server along with the original request.
4. The web server checks the user ID and password, using its own user authentication mechanism.
5. If the user logs in successfully, the browser caches the authentication information and sends it in an HTTP Authorization header with every subsequent page request from the user.
6. The web server processes the requested page and all future page requests from the browser that contain the HTTP Authorization header, if it is valid for the requested page.
   You can use web server authentication without using any ColdFusion security features. In this case, you configure and manage all user security through the web server's interfaces.
   You can also use web server authentication with ColdFusion application authentication, and thus use ColdFusion security for authorization. If the web server uses basic HTML authentication, the ColdFusion `cflogin` tag provides access to the user ID and password that the user entered to log in to the web server. If the web server uses Digest or NTLM authentication, the `cflogin` tag normally gets the user ID, but not the password.
   As a result, your application rely on the web server to authenticate the user against its user and password information, and does not have to display a login page. You use the `cflogin` and `cfloginuser` tags to log

© 2014 Adobe Systems Incorporated. All rights reserved.

the user into the ColdFusion user security system, and use the `IsUserInAnyRole` and `GetAuthUser` functi ons to ensure user authorization. For more information on this form of security, see *A web server authentication security scenario* in Security scenarios.

> **Note**
>
> If a user has logged in using web server authentication and has not logged in using ColdFusion application authentication, the `GetAuthUser` tag returns the web server user ID. You could use this feature to combine web server authentication with application authorization based on the user's ID.

Application authentication

With application authentication, you do not rely on the web server to enforce application security. The application performs all user authentication and authorization. The application displays a login page, checks the user's identity and login against its own authorization store, such as an LDAP directory or database, and logs the user into ColdFusion using the `cfloginuser` tag. The application then uses the `IsUserInAnyRole` and `GetAuthUser` fun ctions to check the user's roles or identity for authorization before running a ColdFusion page or specific code on a page. For an example of application authentication use, see *A web server authentication security scenario* in Securit y scenarios.

ColdFusion authentication storage and persistence

How ColdFusion application authentication information is maintained by the browser and ColdFusion, and therefore how long it is available, depends on the following:

- Whether the user's browser enables cookies
- Whether the application supports the Session scope for login storage

> **Note**
>
> For detailed information on Session scope, see Configuring and using session variables. Cookie scope contains the cookies that arthe browser sends; for more information on using cookies, see cfcookie in the CFML Reference.

Authentication and cookies

Because HTTP is connectionless, a login can last beyond a single web page viewing only if the browser provides a unique identifier that software on the server uses to confirm that the current user is authenticated. Normally, this is done by using memory-only cookies that are automatically destroyed when the user closes all open browser windows. The specific cookies and how they are used depend on whether the application supports the Session scope for login storage.

> **Note**
>
> For information on user logins without cookies, see *Using ColdFusion security without cookies* in About user security.

Using the Session scope

If you do the following, ColdFusion maintains login information in the Session scope instead of the Cookie scope:

- Enable the Session scope in the ColdFusion Administrator and the Application.cfc initialization code or `cfapp lication` tag.

© 2014 Adobe Systems Incorporated. All rights reserved.

- Specify `loginStorage="Session"` in the Application.cfc initialization code or `cfapplication` tag.When ColdFusion maintains login information in the Session scope, it stores the authentication details in a Session.cfauthorization variable, and ColdFusion uses the session cookie information to identify the user. Session-based authentication has the following advantages over less persistent login storage:
- After the user logs in, the user ID and password are not passed between the server and the browser.
- The login information and the session share a single time-out. You do not have to manually synchronize sessions and logins.
- If you use server clusters, the Session scope login ID is available across the cluster. For more information on server clustering, see *Configuring and Administering ColdFusion*.
  If you do not enable the Session scope, the authentication information is not kept in a persistent scope. Instead, the detailed login information is placed in a memory-only cookie (CFAUTHORIZATION_*applicationName*) with a base64-encoded string that contains the user name, password, and application name. The client sends this cookie to the web server each time it makes a page request while the user is logged-in. Use SSL for all page transactions to protect the user ID and password from unauthorized access.

Using ColdFusion security without cookies

Implement a limited-lifetime form of ColdFusion security if the user's browser does not support cookies. In this case you do not use the `cflogin` tag, only the `cfloginuser` tag. It is the only time you should use the `cfloginuser` tag outside a `cflogin` tag.
Without browser cookies, the effect of the `cfloginuser` tag is limited to a single HTTP request. Provide your own authentication mechanism and call `cfloginuser` on each page on which you use ColdFusion login identification.

© 2014 Adobe Systems Incorporated. All rights reserved.

Using ColdFusion security tags and functions

ColdFusion provides the following tags and functions for user security:

Tag or function	Purpose
cflogin	A container for user authentication and login code. The body of the tag runs only if the user is not logged in. When using application-based security, you place 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 of the tag includes a `cfloginuser` tag (or a ColdFusion page that contains a `cfloginuser` tag) to establish the authenticated user's identity in ColdFusion.
cfloginuser	Identifies (logs in) a user to ColdFusion. Specifies the user's ID, password, and roles. This tag is typically used inside a `cflogin` tag. The `cfloginuser` tag requires three attributes, `name`, `password`, and `roles`, and does not have a body. The `roles` attribute is a comma-delimited list of role identifiers to which the logged-in user belongs. All spaces in the list are treated as part of the role names, so you should not follow commas with spaces.While the user is logged-in to ColdFusion, security functions access the user ID and role information.
cflogout	Logs out the current user. Removes knowledge of the user ID and roles from the server. If you do not use this tag, the user is automatically logged out as described in *Logging out users* in Using ColdFusion security tags and functions.The `cflogout` tag does not take any attributes, and does not have a body.
cfNTauthenticate	Authenticates a user name and password against the NT domain on which ColdFusion server is running, and optionally retrieves the user's groups.
cffunction	If you include a `roles` attribute, the function executes only when there is a logged-in user who belongs to one of the specified roles.
IsUserInAnyRole	Returns True if the current user is a member of the specified role.

© 2014 Adobe Systems Incorporated. All rights reserved.

GetAuthUser	Returns the ID of the currently logged-in user.This tag first checks for a login made with `cfloginuser` tag. If none exists, it checks for a web server login (cgi.remote_user.

Using the cflogin tag

The `cflogin` tag executes only if there is no currently logged-in user. It has the following three optional arguments that control the characteristics of a ColdFusion login:

Attribute	Use
idleTimeout	If no page requests occur during the idleTimeout period, ColdFusion logs out the user. The default is 1800 seconds (30 mins). This is ignored if login information is stored in the Session scope.
applicationToken	Limits the login validity to a specific application as specified by a ColdFusion page's cfapplication tag. The default value is the current application name.
cookieDomain	Specifies the domain of the cookie used to mark a user as logged-in. You use cookieDomain if you have a clustered environment (for example, x.acme.com, x2.acme.com, and so on). This lets the cookie work for all the computers in the cluster.

Login identification scope and the applicationToken attribute

The login identification created by the `cflogin` tag is valid only for pages within the directory that contains the page that uses the `cflogin` tag and any of its subdirectories. Therefore, if a user requests a page in another directory tree, the current login credentials are not valid for accessing those pages. This security limitation lets you use the same user names and passwords for different sections of your application (for example, a UserFunctions tree and a SecurityFunctions tree) and enforce different roles to the users depending on the section.
ColdFusion uses the `applicationToken` value to generate a unique identifier that enforces this rule. The default a `pplicationToken` value is the current application name, as specified by a `cfapplication` tag or Application.cfc unitization code. In normal use, you need not specify an `applicationToken` value in the `cflogin` tag.

Specifying the Internet domain

Use the `cookieDomain` attribute to specify the domain of the cookie used to mark a user as logged-in. You use `co okieDomain` if you have a clustered environment (for example, www.acme.com, www2.acme.com, and so on). This lets the cookie work for all computers in the cluster. For example, to ensure that the cookie works for all servers in the acme.com domain, specify `cookieDomain=".acme.com".` To specify a domain name, start the name with a period.

> Before setting the cookie domain, consider the other applications or servers in the broader domain might have access to the cookie. For example, a clustered payroll application at payroll1.acme.com, payroll2.acme.com, and so on, might reveal sensitive information to the test computer at test.acme.com, if the cookie domain is broadly set to `".acme.com"`.

© 2014 Adobe Systems Incorporated. All rights reserved.

Getting the user ID and password

The `cflogin` tag has a built-in cflogin structure that contains two variables, cflogin.username and cflogin.password, if the page is executing in response to any of the following:

- Submission of a login form that contains input fields with the names `j_username` and `j_password`.
- A request that uses HTTP Basic authentication and, therefore, includes an Authorization header with the user name and password.
- A message from the Flash Remoting gatewayConnection object that has the `setCredentials` method set.
- A request that uses NTLM or Digest authentication. In this case, the user name and password are hashed using a one-way algorithm before they are placed in the Authorization header; ColdFusion gets the user name from the web server and sets the cflogin.password value to the empty string.
  You use the first three techniques with application authentication, and the last technique with web server authentication. The cflogin structure provides a consistent interface for determining the user's login ID and password, independent of the technique that you use for displaying the login form.

> Login forms send the user name and password without encryption. Basic HTTP authentication sends the user name and password in a base64-encoded string with each request; this format can easily be converted back to plain text. Use these techniques only with https requests, or when you are not concerned about password security.

Provide login information to your application for authentication as follows:

Use a login form to get user information

When you build an application that gets the User ID and password using a login form, the `cflogin` tag checks for the existence of a cflogin structure containing the user's login information. If the structure does not exist, it displays a login form, typically using a `cfinclude` tag on a login page; the following code shows this use.
In the Application.cfc `onRequestStart` method, or a ColdFusion page or CFC method called by the method, you have the following:

```
<cflogin>
 <cfif NOT IsDefined("cflogin")>
  <cfinclude template="loginform.cfm">
  <cfabort>
  <cfelse>
 <!--- Code to authenticate the user based on the cflogin.user and
  cflogin.password values goes here. --->
 <!--- If User is authenticated, determine any roles and use a line like the
  following to log in the user. --->
  <cfloginuser name="#cflogin.name#"
  Password = "#cflogin.password#"
  roles="#loginQuery.Roles#">
 </cfif>
</cflogin>
```

A simple login form looks like the following:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cfform name="loginform" action="#CGI.script_name#?#CGI.query_string#"
  method="Post">
 <table>
  <tr>
   <td>user name:</td>
   <td><cfinput type="text" name="j_username" required="yes"
    message="A user name is required"></td>
  </tr>
  <tr>
   <td>password:</td>
   <td><cfinput type="password" name="j_password" required="yes"
    message="A password is required"></td>
  </tr>
 </table>
 <br>
 <input type="submit" value="Log In">
</cfform>
```

Use a browser dialog box to get user information

Application authentication does not require you to use a login form; you can rely on the browser to display its standard login dialog box, instead. To do so, your `cflogin` tag body returns an HTTP status 401 to the browser if the user is not logged in or if the login fails; that is, if it does not have a valid cflogin structure. The browser displays its login dialog box. When the user clicks the login button on the dialog box, the browser returns the login information as an HTTP Authorization header to ColdFusion, which places the information in the `cflogin` tag's cflogin structure.

This technique has the advantage of simplicity; you do not need a login form and the user gets a familiar-looking login page. Be careful of security issues, however. The browser sends the user name and password in a base64-encoded string, not just when the user logs in, but with each request. Use SSL (Secure Sockets Layer) for all page transactions to protect the user ID and password from unauthorized access.

Note

Ensure that your web server is configured correctly to support browser-based login forms for this use. For example, in IIS 5, enable anonymous access and disable Basic authentication and Integrated Windows authentication.

The following `cflogin` tag tells the browser to display a login form if the user has not logged in:

```
<cflogin>
 <cfif NOT IsDefined("cflogin")>
  <cfheader statuscode="401">
  <cfheader name="www-Authenticate" value="Basic
   realm=""MM Wizard #args.authtype# Authentication""">
 </cfif>
 <cfabort>
 <cfelse>
  <!--- code to authenticate the user based on the cflogin.user and
   cflogin.password values goes here. --->
</cflogin>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

Log in a user using Flash Remoting

If you are developing a Rich Internet Application with Flash and Flash Remoting, your ColdFusion application does not need to be coded specially for a Flash login. The Flash Remoting gateway makes the user ID and password available to the `cflogin` tag in the `cflogin` structure.
In your Flash code, you use the ActionScript `SetCredentials` method to send login information to ColdFusion. Your Flash SWF file displays the user ID and password fields, and uses their contents in the `setCredentials` met hod, as follows:

```
if (inited == null)
{
 inited = true;
 NetServices.setDefaultGatewayUrl("http://localhost/flashservices/gateway");
 gatewayConnection = NetServices.createGatewayConnection();
 gatewayConnection.setCredentials(userID, password);
 myService = gatewayConnection.getService("securityTest.thecfc", this);
}
```

For more information on using Flash Remoting, see Using the Flash Remoting Service_ and_ Using Flash Remoting Update.

Logging out users

After a user logs in, the ColdFusion user authorization and authentication information remains valid until any of the following happens:

- The application uses a `cflogout` tag to log out the user, usually in response to the user clicking a log-out link or button.
- If your application uses the Session scope for login information, the session ends.
- If your application does not use the Session scope for login information, the user does not request a new page for the `cflogin` tag `idleTimeout` period.
- If your application does not use Session scope for login information, or if you use J2EE-based session identification, the user closes all browser windows.
  Logging out a user by using the `cflogout` tag does not close the user's session, but if you use session login storage, it does remove the login information (the Session.cfauthorization variable) from the Session scope. For more information on ending sessions, see *Ending a session in Configuring and using session variables*.

> If you use web server-based authentication or any form authentication that uses a Basic HTTP Authorization header, the browser continues to send the authentication information to your application until the user closes the browser, or in some cases, all open browser windows. As a result, after the user logs out and your application uses the `cflogout` tag, until the browser closes, the cflogin structure in the `cflogin_` tag will contain the logged-out user's UserID and password. If a user logs out and does not close the browser, another user can access pages with the first user's login.

© 2014 Adobe Systems Incorporated. All rights reserved.

Security scenarios

There are two detailed security scenarios. The first scenario uses the web server to perform the authentication against its user and password database. The second scenario uses ColdFusion for all authentication and authorization.

A web server authentication security scenario

An application that uses web server authentication could work as follows. The example in *Web server-based authentication user security example* in Implementing user security implements this scenario.

1. When the user requests a page from a particular directory on the server for the first time after starting the browser, the web server displays a login page and logs in the user. The web server handles all user authentication.
2. Because the user requested a ColdFusion page, the web server hands the request to ColdFusion.
3. When ColdFusion receives a request for a ColdFusion page, it instantiates the Application.cfc and runs `onRequestStart` method. If you use an Application.cfm page in place of the Application.cfc, it runs the contents of the Application.cfm page before it runs the requested page. The `onRequestStart` method or Application.cfm page contains a `cflogin` tag. ColdFusion executes the `cflogin` tag body if the user is not logged into ColdFusion. The user is logged in if the `cfloginuser` tag has run successfully for this application and the user has not been logged out.
4. Code in the `cflogin`tag body uses the user ID and password from the browser login, contained in the cflogin.name and cflogin.password variables, as follows. (With Digest or NTLM web server authentication, the cflogin.password variable is the empty string.)
   a. It checks the user's name against information it maintains about users and roles. In a simple case, the application has two roles, one for users and one for administrators. The CFML assigns the Admin role to any user logged on with the user ID *Admin* and assigns the User role to all other users.
   b. It calls the `cfloginuser` tag with the user's ID, password, and roles, to identify the user to ColdFusion.
5. Application.cfc or the Application.cfm page completes processing, and ColdFusion processes the requested application page.
6. The application uses the `IsUserInAnyRole` function to check whether the user belongs to a role before it runs protected code that must be available only to users in that role.
7. The application uses the `GetAuthUser` function to determine the user ID; for example, to display the ID for personalization. It can also use the ID as a database key to get user-specific data.

> If you use web server-based authentication or any form authentication that uses a Basic HTTP Authorization header, the browser continues to send the authentication information to your application until the user closes the browser, or in some cases, all open browser windows. As a result, after the user logs out and your application uses the `cflogout` tag, until the browser closes, the cflogin structure in the `cflogin_` tag will contain the logged-out user's UserID and password. If a user logs out and does not close the browser, another user can access pages with the first user's login.

An application authentication security scenario

An application that does its own authentication works as follows. The example in *Application-based user security example* in Implementing user security implements this scenario.

1. Whenever ColdFusion receives a request for a ColdFusion page, it instantiates the Application.cfc and runs the `onRequestStart` method. If you use an Application.cfm page in place of Application.cfc, ColdFusion runs the contents of the Application.cfm page before it runs the requested page. The `onRequestStart` method or Application.cfm page contains the `cflogin` tag. ColdFusion executes the `cflogin` tag body if the user

© 2014 Adobe Systems Incorporated. All rights reserved.

is not logged in. A user is logged in if the `cfloginuser` tag has run during the current session and the user had not been logged out by a `cflogout` tag.

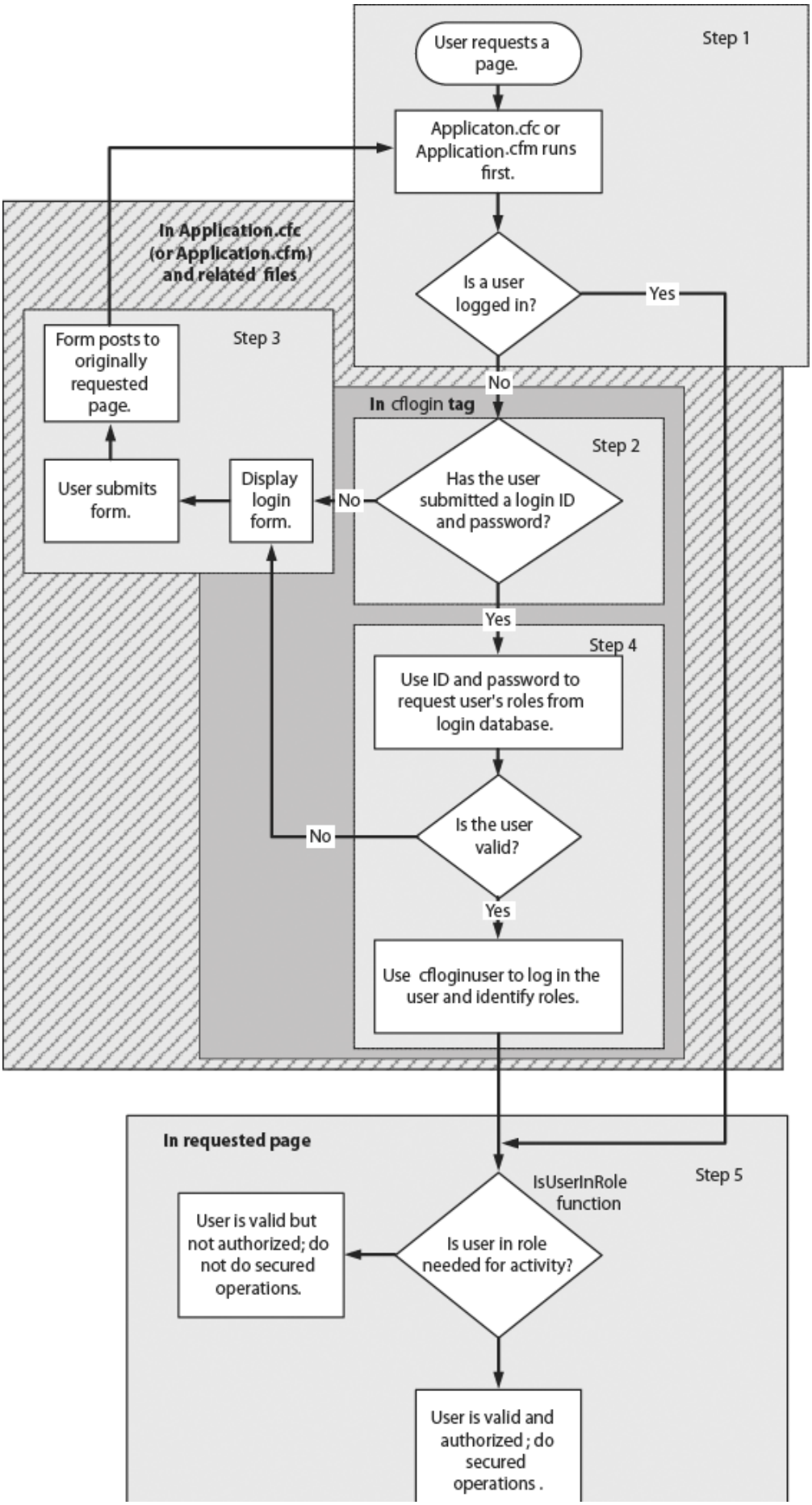
2. Code in the `cflogin` tag body checks to see if it has received a user ID and password, normally from a login form.
3. If there is no user ID or password, the code in the `cflogin` tag body displays a login form that asks for the user's ID and password. The form posts the login information back to the originally requested page, and the `cflogin` tag in the `onRequestStart` method or the Application.cfm page runs again. This time, the `cflogin` tag body code checks the user name and password against a database, LDAP directory, or other policy store, to ensure that the user is valid and get the user's roles.
4. If the user name and password are valid, the `cflogin` tag body code calls the `cfloginuser` tag with the user's ID, password, and roles, to identify the user to ColdFusion.
5. When the user is logged in, application pages use the `IsUserInAnyRole` function to check whether the user belongs to a role before they run protected code that must be available only to users in that role. The application can use the `GetAuthUser` function to determine the user ID; for example, to display the ID for personalization. It can also use the ID as a database key to get user-specific data.
6. Each application page displays a link to a logout form that uses the `cflogout` tag to log out the user. Typically, the logout link is in a page header that appears in all pages. The logout form can also be in the Application.cfc (for example, in the `onRequestStart` or `onRequestEnd` method) or on the Application.cfm page.

   Although this scenario shows one method for implementing user security, it is only an example. For example, your application could require users to log in for only some pages, such as pages in a folder that contains administrative functions. When you design your user security implementation, remember the following:

- Code in the `cflogin` tag body executes only if there is no user logged in.
- With application authentication, you write the code that gets the identification from the user and tests this information against a secure credential store.
- After you have authenticated the user, you use the `cfloginuser` tag to log the user into ColdFusion.

The following image shows this flow of control. For simplicity, it omits the log-out option.

© 2014 Adobe Systems Incorporated. All rights reserved.



© 2014 Adobe Systems Incorporated. All rights reserved.

© 2014 Adobe Systems Incorporated. All rights reserved.

Implementing user security

You can implement security in the following ways.

Using the Dreamweaver Login Wizard

ColdFusion installs a Login Wizard command in the Dreamweaver Commands menu that generates a skeleton set of pages for managing user authentication and authorization.
The wizard asks you to select how to authenticate the login information. Select one of the following options:

- **Simple** Specify a single user ID and password in the wizard. All users must enter this information to log in. Use this option for testing, or use the generated files as a template where you can replace the authentication code with more complex code. For example, to verify the ID and password against a database.
- **NT domain** Specify an NT domain in the wizard, and the wizard generates code that queries the domain.
- **LDAP** Specify the LDAP server and port, the user name and password required to access the login data, and the distinguished name to use to start the search for the user name. The wizard generates the code to query the LDAP server with the user ID and password.
  The wizard asks you to select one of the following options for displaying the request for login information:

- Browser Dialog Box
- ColdFusion Login Form

Structure code generated by the Login Wizard

The wizard generates or modifies the following files in the directory or site that you specify:

- **Application.cfc** If this file does not exist, the wizard creates it with a single `onRequestStart` method; it does not specify an application name or any other methods. If the file exists, but does not have an `onRequestStart` method, it adds the method. If Application.cfc and the `onRequestStart` method exist, the wizard inserts the required code at the beginning of the method. The resulting `onRequestStart` method has a `cfincludetag` that specifies mm_wizard_application_include.cfm; it also has a simple form with a logout button, which appears at the top of each page in the application.

  > **Note**
  >
  > If the wizard creates the Application.cfc file, change the file to specify the application name. For more information on Application.cfc, see Designing and Optimizing a ColdFusion Application.

- **mm_wizard_application_include.cfm** The Login Wizard uses the information specified in the wizard fields to set several CFC method arguments. It then uses them to invoke the performlogin method of the master login CFC, mm_wizard.authenticate.
- **mm_wizard_authenticate.cfc**This CFC contains all of the user authentication and login logic. The CFC consists of the following methods:
  - The ntauth, ldapauth, and simpleauth authentication methods check the user's name and ID against the valid login information, and return information about whether the user is authenticated. For the details of how they authenticate the user and the specific return values, see the methods.
  - The performLogin method is the master login method. It contains the `cflogin` tag, which displays the login form and calls the required authentication method. If the authentication method's return argument indicates a valid user, the method logs the user in.
  - The logout method logs out a user. If you specified Browser Dialog Box as the login page type, it also calls the closeBrowser method to close the browser window. This behavior is necessary because the browser continues to send the old login credentials after the user logs out, and the `cflogin` tag will automatically use them and log the user in again.
  - The closeBrowser method closes the browser window or tells the user to close the browser window to

© 2014 Adobe Systems Incorporated. All rights reserved.

complete the logout, depending on the browser type.

- **mm_wizard_login.cfm** This file contains a ColdFusion login form. The wizard generates this file for all options, but does not use it if you specify Browser Dialog login.
- **index.cfm or mm_wizard_index.cfm** The wizard generates an index.cfm page if the directory does not have one; otherwise, creates an mm_wizard_index.cfm page. These pages let you test the generated login code before you implement your application, or without using any of your standard application pages. To test your login, open the index.cfm page in your browser.

Modifying the login code for your application

The Login Wizard creates a basic framework for authenticating a user. Customize this framework to meet the needs of your application. Typical security-related changes include the following:

- Providing user-specific role information in the `cflogin` tag
- Authenticating users against a database

Providing user-specific role information

The Login Wizard sets all users in a single role. In mm_wizard_authenticate.cfc, the performlogin method is hard-coded to set the role to "user." The authentication routines handle roles differently. (For the details, see the mm_wizard_authenticate.cfc code.) If your application uses roles for authorization, change the authentication method to get and return valid role information, and change the performlogin method to use the information in the `roles` attribute of its `cfloginuser` tag.

Authenticating users against a database

If you use a database to maintain user IDs and passwords, create your login framework by specifying simple authentication, and modify the code to use the database. The following instructions describe a simple way to change the code to use a database. They do not include all the cleanup work (particularly, removing the hard-coded user name and password) needed for a well-formatted application.
Replace the following code:

```
<cfif sUserName eq uUserName AND sPassword eq uPassword>
 <cfset retargs.authenticated="YES">
<cfelse>
 <cfset retargs.authenticated="NO">
</cfif>
<cfreturn retargs>
```

With code like the following:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cfquery name="loginQuery" dataSource="#Application.DB#" >
 SELECT *
 FROM Users
 WHERE UserName = <cfqueryparam value="#uUserName#" CFSEQLType=
  'CF_SQL_VARCHAR'AND password = <cfqueryparam value="#uPassword#"
  CFSEQLType='CF_SQL_VARCHAR'>
</cfquery>

<cfif loginQuery.recordcount gt 0>
 <cfset retargs.authenticated="YES">
 <cfset retargs.roles=loginQuery.roles>
<cfelse>
 <cfset retargs.authenticated="NO">
</cfif>
<cfreturn retargs>
```

Note

For greater security, consider using a hashed password. Do not store the password directly in the database; instead, use the `hash` function to create a secure password fingerprint, and store it in the database. When the user provides a password, use the Hash function on the submitted string and compare it with the value in the database.

Web server based authentication user security example

The following example shows how to implement user security using web-server-based basic authentication and two roles, user and administrator.
This example has two ColdFusion pages:

1. The Application.cfc page logs the user into the ColdFusion security system and assigns the user to specific roles based on the user's ID.This page also includes the one-button form and logic for logging out a user, which appears at the top of each page.
2. The securitytest.cfm page is a sample application page. It displays the logged-in user's roles.
   This simple example does not provide a user log-out interface. Test the security behavior by adding your own pages to the same directory as the Application.cfc page.

Example: Application.cfc

The Application.cfc page consists of the following:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cfcomponent>
<cfset This.name = "Orders">
<cffunction name="OnRequestStart">
 <cfargument name = "request" required="true"/>
 <cflogin>
  <cfif IsDefined("cflogin")>
   <cfif cflogin.name eq "admin">
    <cfset roles = "user,admin">
   <cfelse>
    <cfset roles = "user">
   </cfif>
   <cfloginuser name = "#cflogin.name#" password = "#cflogin.password#"
    roles = "#roles#" />
  <cfelse>
   <!--- This should never happen. --->
   <h4>Authentication data is missing.</h4>
   Try to reload the page or contact the site administrator.
  <cfabort>
  </cfif>
 </cflogin>
</cffunction>
</cfcomponent>
```

Reviewing the code

The Application.cfc `onRequestStart` method executes before the code in each ColdFusion page in an application. For more information on the Application.cfc page and when it is executed, see Designing and Optimizing a ColdFusion Application.The following table describes the CFML code in Application.cfc and its function:

Code	Description

`<cfcomponent>`
`<cfset This.name = "Orders">`
`<cffunction name="OnRequestStart">`
 	
`<cfargument name = "request"`
`required="true"/>`
	Identifies the application and starts the `onRequestStart` method that runs at the starts of each request. The login information on this page only applies to this application.

`<cflogin>`
`<cfif IsDefined("cflogin")>`
`<cfif cflogin.name eq "admin">`
`<cfset roles = "user,admin">`
`<cfelse>`
`<cfset roles = "user">`
`</cfif>`
	Executes if there is no logged-in user.Makes sure that the user is correctly logged in by the web server. (Otherwise, there would be no cflogin variable.)Sets a roles variable based on the user's ID. Assigns users named "admin" to the admin role. Assigns all other users to the users role.

© 2014 Adobe Systems Incorporated. All rights reserved.

`<cfloginuser name = "#cflogin.name#" password = "#cflogin.password#" roles = "#roles#" />`	Logs the user into the ColdFusion security system and specifies the user's password, name, and roles. Gets the password and name directly from the cflogin structure.
`<cfelse> <!--- This should never happen. ---> <h4>Authentication data is missing.</h4> Try to reload the page or contact the site administrator. <cfabort>`	This code should never run, but if the user somehow got to this page without logging in to the web server, this message would display and ColdFusion would stop processing the request.
`</cfif> </cflogin> </cffunction> </cfcomponent>`	Ends the if/else block.Ends the `cflogin` tag body.Ends the `onRequestStart` method.Ends the Application component.

Example: securitytest.cfm

The securitytest.cfm page shows how any application page uses ColdFusion user authorization features. The web server ensures the existence of an authenticated user, and the Application.cfc page ensures that the user is assigned to roles the page content appears. The securitytest.cfm page uses the `IsUserInAnyRole` and `GetAuthUser` functions to control the information that is displayed.
The securitytest.cfm page consists of the following:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
 <title>Basic authentication security test page</title>
</head>

<body>
<cfoutput>
 <h2>Welcome #GetAuthUser()#!</h2>
</cfoutput>

ALL Logged-in Users see this message.<br>
<br>
<cfscript>
 if (IsUserInRole("admin"))
  WriteOutput("Users in the admin role see this message.<br><br>");
 if (IsUserInRole("user"))
  WriteOutput("Everyone in the user role sees this message.<br><br>");
</cfscript>

</body>
</html>
```

Reviewing the code

The following table describes the securitytest.cfm page CFML code and its function:

Code	Description

`<cfoutput>`
`  <h2>Welcome #GetAuthUser()#!</h2>`
 	
`</cfoutput>`
	User is already logged in by Application.cfc. Displays a welcome message that includes the user's login ID.

`ALL Logged-in Users see this` `message. ` ` `	Displays this message in all cases. The page does not display until a user is logged in.

© 2014 Adobe Systems Incorporated. All rights reserved.

`<cfscript>` ` if (IsUserInRole("admin"))` `  WriteOutput("Users in the admin role see this message.  ");` ` if (IsUserInRole("user"))` `  WriteOutput("Everyone in the user role sees this message.  ");` `</cfscript>`	Tests whether the user belongs to each of the valid roles. If the user is in a role, displays a message with the role name.The user sees one message per role to which the user belongs.

Application-based user security example

The following example shows how to implement user security by authenticating users and then allowing users to see or use only the resources that they are authorized to access.
This example has three ColdFusion pages:

- The Application.cfc page contains the authentication logic that checks whether a user is logged in, requests the login page if the user is not logged in, and authenticates the data from the login page. If the user is authenticated, it logs the user in.This page also includes the one-button form and logic for logging out a user, which appears at the top of each page.
- The loginform.cfm page displays the login form. The code on this page could also be included in Application.cfc.
- The securitytest.cfm page is a sample application page. It displays the logged-in user's roles.
  Test the security behavior by adding your own pages to the same directory as the Application.cfc page.
  The example gets user information from the LoginInfo table of the cfdocexamples database that is installed with ColdFusion. You can replace this database with any database containing UserID, Password, and Roles fields. The sample database contains the following data:

UserID	Password	Roles
BobZ	Ads10	Employee,Sales
JaniceF	Qwer12	Contractor,Documentation
RandalQ	ImMe	Employee,Human Resources,Manager

Because spaces are meaningful in roles strings, do not follow the comma separators in the Roles fields with spaces.

Example: Application.cfc

The Application.cfc page consists of the following:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cfcomponent>
<cfset This.name = "Orders">
<cfset This.Sessionmanagement="True">
<cfset This.loginstorage="session">

<cffunction name="OnRequestStart">
 <cfargument name = "request" required="true"/>
 <cfif IsDefined("Form.logout")>
  <cflogout>
 </cfif>

 <cflogin>
  <cfif NOT IsDefined("cflogin")>
   <cfinclude template="loginform.cfm">
   <cfabort>
  <cfelse>
   <cfif cflogin.name IS "" OR cflogin.password IS "">
    <cfoutput>
     <h2>You must enter text in both the User Name
     and Password fields.
     </h2>
    </cfoutput>
    <cfinclude template="loginform.cfm">
    <cfabort>
   <cfelse>
    <cfquery name="loginQuery" dataSource="cfdocexamples">
    SELECT UserID, Roles
    FROM LoginInfo
    WHERE
     UserID = '#cflogin.name#'
     AND Password = '#cflogin.password#'
    </cfquery>
    <cfif loginQuery.Roles NEQ "">
     <cfloginuser name="#cflogin.name#" Password = "#cflogin.password#"
      roles="#loginQuery.Roles#">
    <cfelse>
     <cfoutput>
      <H2>Your login information is not valid.<br>
      Please Try again</H2>
     </cfoutput>
     <cfinclude template="loginform.cfm">
     <cfabort>
    </cfif>
   </cfif>
  </cfif>
 </cflogin>

 <cfif GetAuthUser() NEQ "">
  <cfoutput>
    <form action="securitytest.cfm" method="Post">
    <input type="submit" Name="Logout" value="Logout">
   </form>
  </cfoutput>
 </cfif>

</cffunction>
</cfcomponent>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

Reviewing the code

The Application.cfc page executes before the code in each ColdFusion page in an application. For more information on the Application.cfc page and when it is executed, see Designing and Optimizing a ColdFusion Application.The following table describes the CFML code in Application.cfc and its function:

Code	Description

`<cfcomponent>`
`<cfset This.name = "Orders">`
`<cfset This.Sessionmanagement="True">`
`<cfset This.loginstorage="session">`
 	
`<cffunction name="OnRequestStart">`
 	
`<cfargument name = "request" required="true"/>`
	Identifies the application, enables session management, and enables storing login information in the Session scope.Begins the definition of the `onRequestStart` method that runs at the starts of each request.

`<cfif IsDefined("Form.logout")>`
`<cflogout>`
`</cfif>`
	If the user just submitted the logout form, logs out the user. The following cflogin tag runs as a result.

`<cflogin>`
`<cfif NOT IsDefined("cflogin")>`
`<cfinclude template="loginform.cfm">`
`<cfabort>`
	Runs if there is no logged-in user.Tests to see if the user has submitted a login form. If not, uses `cfinclude` to display the form. The built-in `cflogin` variable exists and contains the user name and password only if the login form used `j_username` and `j_password` for the input fields.The cfabort tag prevents processing of any code that follows on this page.

`<cfelse>`
`<cfif cflogin.name IS "" OR cflogin.password IS "">`
`<cfoutput>`
`<h2>You must enter text in both the User Name and Password fields. </h2>`
`</cfoutput>`
`<cfinclude template="loginform.cfm">`
`<cfabort>`
	Runs if the user submitted a login form. Tests to make sure that both name and password have data. If either variable is empty, displays a message, followed by the login form. The `cfabort` tag prevents processing of any code that follows on this page.

© 2014 Adobe Systems Incorporated. All rights reserved.

`<cfelse>`
` <cfquery name="loginQuery" dataSource="cfdocexamples">`
` SELECT UserID, Roles`
` FROM LoginInfo`
` WHERE`
` UserID = '#cflogin.name#'`
` AND Password = '#cflogin.password#'`
` </cfquery>`
	Runs if the user submitted a login form and both fields contain data. Uses the cflogin structure's `name` and `password` entries to find the user record in the database and get the user's roles.

`<cfif loginQuery.Roles NEQ "">` ` <cfloginuser name="#cflogin.name#" Password = "#cflogin.password#" roles="#loginQuery.Roles#">`	If the query returns data in the Roles field, logs in the user using the user's name and password and the Roles field from the database. In this application, every user must be in some role.
`<cfelse>` ` <cfoutput>` ` <H2>Your login information is not valid. ` `  Please Try again</H2>` ` </cfoutput>` ` <cfinclude template="loginform.cfm">` ` <cfabort>`	Runs if the query did not return a role. If the database is valid, this means there was no entry matching the user ID and password. Displays a message, followed by the login form. The `cfabort` tag prevents processing of any code that follows on this page.

`</cfif>`
`</cfif>`
`</cfif>`
`</cflogin>`
	Ends the loginquery.Roles test code.Ends the form entry empty value test.Ends the form entry existence test.Ends the `cflogin` tag body.

© 2014 Adobe Systems Incorporated. All rights reserved.

`<cfif GetAuthUser() NEQ "">`
`<cfoutput>`
`<form action="securitytest.cfm" method="Post">`
`<input type="submit" Name="Logout" value="Logout">`
`</form>`
`</cfoutput>`
`</cfif>`
	If a user is logged-in, displays the Logout button.If the user clicks the button, posts the form to the application's (theoretical) entry page, index.cfm. Application.cfc then logs out the user and displays the login form. If the user logs in again, ColdFusion displays index.cfm.

`</cffunction>` `</cfcomponent>`	Ends the `onRequestStart` methodEnds the Application component.

Example: loginform.cfm

The loginform.cfm page consists of the following:

```
<H2>Please Log In</H2>
<cfoutput>
 <form action="#CGI.script_name#?#CGI.query_string#" method="Post">
  <table>
   <tr>
    <td>user name:</td>
    <td><input type="text" name="j_username"></td>
   </tr>
   <tr>
    <td>password:</td>
    <td><input type="password" name="j_password"></td>
   </tr>
  </table>
  <br>
  <input type="submit" value="Log In">
 </form>
</cfoutput>
```

Reviewing the code

The following table describes the loginform.cfm page CFML code and its function:

Code	Description

© 2014 Adobe Systems Incorporated. All rights reserved.

`<H2>Please Log In</H2>` `<cfoutput>` ` <form` `action="#CGI.script_name#?#CGI.que` `ry_string#" method="Post">` ` <table>` ` <tr>` `  <td>user name:</td>` `  <td><input type="text"` `name="j_username"></td>` ` </tr>` ` <tr>` `  <td>password:</td>` `  <td><input type="password"` `name="j_password"></td>` ` </tr>` ` </table>` `  ` ` <input type="submit" value="Log` `In">` ` </form>` `</cfoutput>`	Displays the login form. Constructs the form `action` at tribute from CGI variables, with a ? character preceding the query string variable. This technique works because loginform.cfm is accessed by a `cfinclude` ta g on Application.cfc, so the CGI variables are those for the originally requested page.'The form requests a user ID and password and posts the user's input to the page specified by the newurl variable.Uses the field names j_username and j_password. ColdFusion automatically puts form fields with these values in the cflogin.name and cflogin.password variables inside the cflogin tag.

Example: securitytest.cfm

The securitytest.cfm page shows how any application page can use ColdFusion user authorization features. Application.cfc ensures the existence of an authenticated user before the page content appears. The securitytest.cfm page uses the `IsUserInAnyRole` and `GetAuthUser` functions to control the information that is displayed.
The securitytest.cfm page consists of the following:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
 <title>Security test page</title>
</head>

<body>
<cfoutput>
 <h2>Welcome #GetAuthUser()#!</h2>
</cfoutput>

ALL Logged-in Users see this message.<br>
<br>
<cfscript>
 if (IsUserInRole("Human Resources"))
  WriteOutput("Human Resources members see this message.<br><br>");
 if (IsUserInRole("Documentation"))
  WriteOutput("Documentation members see this message.<br><br>");
 if (IsUserInRole("Sales"))
  WriteOutput("Sales members see this message.<br><br>");
 if (IsUserInRole("Manager"))
  WriteOutput("Managers see this message.<br><br>");
 if (IsUserInRole("Employee"))
  WriteOutput("Employees see this message.<br><br>");
 if (IsUserInRole("Contractor"))
  WriteOutput("Contractors see this message.<br><br>");
</cfscript>

</body>
</html>
```

Reviewing the code

The following table describes the securitytest.cfm page CFML code and its function:

Code	Description

`<cfoutput>`
`  <h2>Welcome #GetAuthUser()#!</h2>`
 	
`</cfoutput>`
	Displays a welcome message that includes the user's login ID.

`ALL Logged-in Users see this message. ` ` `	Displays this message in all cases. The page does not display until a user is logged in.

© 2014 Adobe Systems Incorporated. All rights reserved.

`<cfscript>` `if (IsUserInRole("Human Resources"))` `  WriteOutput("Human Resources members see this message.  ");` `if (IsUserInRole("Documentation"))` `  WriteOutput("Documentation members see this message.  ");` `if (IsUserInRole("Sales"))` `  WriteOutput("Sales members see this message.  ");` `if (IsUserInRole("Manager"))` `  WriteOutput("Managers see this message.  ");` `if (IsUserInRole("Employee"))` `  WriteOutput("Employees see this message.  ");` `if (IsUserInRole("Contractor"))` `  WriteOutput("Contractors see this message.  ");` `</cfscript>`	Tests whether the user belongs to each of the valid roles. If the user is in a role, displays a message with the role name.Users see one message per role that they belong.

Using an LDAP directory for security information

LDAP directories are often used to store security information. The following example of a `cflogin` tag checks an LDAP directory to authenticate the user and retrieve the user's roles.
For more information on using LDAP directories with ColdFusion, see Managing LDAP Directories.

```
<cfapplication name="Orders" sessionmanagement="Yes" loginstorage="Session">
<cflogin>
 <cfif isDefined("cflogin")>
  <!--- setting basic attributes --->
  <cfset LDAP_root = "o=mycompany.com">
  <cfset LDAP_server = "ldap.mycompany.com">
  <cfset LDAP_port = "389">

  <!--- Create the prefix and suffix parts of the user's DN. --->
  <cfset userPrefix = "cn=">
  <cfset userSuffix = ",ou=Users,o=mycompany.com">

  <!--- Concatenate the user's DN and use it to authenticate. --->
  <cfset LDAP_username = userPrefix&cflogin.name&userSuffix>

  <!--- This filter will look for groups for containing the user's ID. --->
  <cfset userfilter =
   "(&(objectClass=groupOfUniqueNames)
   (uniqueMember=#LDAP_username#))">

  <!--- Search for groups containing the user's dn.
   The groups represent the user's roles.
```

© 2014 Adobe Systems Incorporated. All rights reserved.

```
    NOTE: Your LDAP permissions must allow authenticated users to search.
    groups. --->
   <cftry>
    <cfldap action="QUERY"
     name="auth"
     attributes="cn"
     referral="yes"
     start="#LDAP_root#"
     scope="SUBTREE"
     server="#LDAP_server#"
     port="#LDAP_port#"
     filter="#userfilter#"
     username="#LDAP_username#"
     password="#cflogin.password#"
    >
     <cfcatch type="any">
     <cfif FindNoCase("Invalid credentials", cfcatch.detail)>
      <cfoutput>
       <script>alert("User ID or
       Password invalid for user:
       #cflogin.name#")</script>
      </cfoutput>
      <cfabort>
     <cfelse>
      <cfoutput>
       <script>alert("Unknown error for user: #cflogin.name#
       #cfcatch.detail#")</script>
      </cfoutput>
      <cfabort>
     </cfif>
    </cfcatch>
   </cftry>

  <!--- If the LDAP query returned a record, the user is valid. --->
   <cfif auth.recordcount>
    <cfloginuser name="#cflogin.name#" password="#cflogin.password#"
     roles="#valueList(auth.cn)#">
   </cfif>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

```
 </cfif>
</cflogin>
```

Reviewing the code

The following table describes the code and its function. Comments and some tab characters have been removed for brevity.

Code	Description

`<cflogin>`
`<cfif isDefined("cflogin")>`
`<!--- setting basic attributes --->`
`<cfset LDAP_root = "o=mycompany.com">`
`<cfset LDAP_server = "ldap.mycompany.com">`
`<cfset LDAP_port = "389">`
 	
`<!--- Create the prefix and suffix parts of the user's DN. --->`
`<cfset userPrefix = "cn=">`
`<cfset userSuffix = ",ou=Users,o=mycompany.com">`
 	
`<!--- Concatenate the user's DN and use it to authenticate. --->`
`<cfset LDAP_username = userPrefix&cflogin.name&userSuffix>`
`<!--- This filter will look for groups for containing the user's ID. --->`
`<cfset userfilter = "(&(objectClass=groupOfUniqueNames)(uniqueMember=#LDAP_username#))">`
	Starts the `cflogin` tag body. Sets several variables to the values used as attributes in the `cfldap` tag. Sets prefix and suffix values used to create a distinquished name (dn) for binding to the LDAP server.
	
	Creates the user's bind dn by concatenating the prefix and suffix with cflogin.name. This variable is used for authenticating the user to the LDAP server.Sets the filter used to search the directory and retrieve the user's group memberships. The group membership represents the user's roles within the organization.

© 2014 Adobe Systems Incorporated. All rights reserved.

`<cftry>`
`<cfldap action="QUERY"`
`name="auth"`
`attributes="cn"`
`referral="yes"`
`start="#LDAP_root#"`
`scope="SUBTREE"`
`server="#LDAP_server#"`
`port="#LDAP_port#"`
`filter="#userfilter#"`
`username="#LDAP_username#"`
`password="#cflogin.password#"`
`>`
	In a `cftry` block, uses the user's concatenated dn to authenticate to the LDAP server and retrieve the common name (cn) attribute for groups to which the user is a member. If the authentication fails the LDAP server returns an error.
	
	Note: The LDAP permissions must allow an authenticated user to read and search groups in order for the query to return results.

`<cfcatch type="any">`
`<cfif FindNoCase("Invalid credentials", cfcatch.detail)>`
`<cfoutput>`
`<script>alert("User ID or Password invalid for user: #cflogin.name#")</script>`
`</cfoutput>`
`<cfabort>`
`<cfelse>`
`<cfoutput>`
`<script>alert("Unknown error for user: #cflogin.name# #cfcatch.detail#")</script>`
`</cfoutput>`
`<cfabort>`
`</cfif>`
`</cfcatch>`
`</cftry>`
	Catches any exceptions. Tests to see if the error information includes the string "invalid credentials", which indicates that either the dn or password is invalid. If so, displays a dialog box with an error message indicating the problem. Otherwise, displays a general error message.If an error is caught, the `cfabort` tag ends processing of the request after displaying the error description.End of `cfcatch` and `cftry` blocks.

`<cfif auth.recordcount>`
`<cfloginuser name="#cflogin.name#" password="#cflogin.password#" roles="#valueList(auth.cn)#">`
`</cfif>`
`</cfif>`
`</cflogin>`
	If the authorization query returns a valid record, logs in the user. Uses the `valueList` function to create a comma-separated list of the users retrieved group memberships, and passes them in the `cfloginuser roles` attribute.Ends the initial `isDefined("cflogin")` `cfif` block .Ends the `cflogin` tag body

© 2014 Adobe Systems Incorporated. All rights reserved.

Security enhancements in ColdFusion 10

Security enhancements in ColdFusion 10 let you reduce XSS and CSRF attack vulnerability. The enhancements also help you manage ColdFusion sessions effectively. The release also includes fixes that reduce other vulnerabilities.

XSS attack

Cross-site scripting (XSS) attacks bypass client-side security mechanisms imposed by web browsers. These methods use Open Web Application Security Project's (OWASP) Enterprise Security API for encoding. An attacker injects malicious scripts into a web page to access information stored in the browser.

- Only the following characters are allowed as values for the attribute `name` in the tag `cform`: alphanumeric characters, _ (underscore), - (hyphen), : (colon), and . (dot). It prevents stored XSS for the `scriptsrc` field.
- The following new encoding methods are added to reduce XSS attack vulnerability: `EncodeForHTML`, `EncodeForHTMLAttribute`, `EncodeForJavaScript`, `EncodeForCSS`, and `EncodeForURL`. Encode the user inputs depending on the contexts. To decode the input string, added a method: `Canonicalize`.

CSRF attack

Cross-Site Request Forgery (CSRF) forces users to execute unwanted actions on a web application for which they are authenticated. For example, sending a URL using an email or chat to a privileged user. Clicking the URL link forces the user to do an action of the attacker's choice.

- Following methods can be used to reduce CSRF vulnerability:
  - `CSRFGenerateToken`: Returns a random token and stores it in the session.
  - `CSRFVerifyToken`: Validates the given token and the key against the same stored in the session.*Example: CSRFGenerateToken*The following example lets you enter value and submit. The page generates a token and calls another ColdFusion page.

```
<cfset csrfToken=CSRFGenerateToken() />
<cfform method="post" action="sayHello.cfm">
 <cfinput name="userName" type="text" >
 <cfinput name="token" value="#csrfToken#" type="hidden" >
 <cfinput name="submit"  value="Say Hello!!" type="submit" >
</cfform>
```

  The following page, `sayHello.cfm`, validates the token generated and displays the output of `CSRFverifyToken(token)`.

```
<cfset token=form.token>
 <cfset validate = CSRFverifyToken(token)>
<cfoutput >#validate#</cfoutput>
```

Note

Enable SessionManagement for protection against CSRF. Disabling session variables in the administrator console disables CSRF protection.

Session improvements

© 2014 Adobe Systems Incorporated. All rights reserved.

You can manage ColdFusion session cookies effectively.

CF Session cookies (CFID, CFTOKEN, CFAuthorization_<app-name>)

The new features to manage session cookies are:

- The following properties of ColdFusion session cookies can be configured at server level or application level:
  - httponly: true by default
  - secure: false by default
  - domain
  - timeout: 30 years by default
    You can set the session cookies at the application level by specifying the settings as a struct in the `Application.cfm`as shown in the following example:

```
<cfset cookiest = {httponly='true', timeout=createTimeSpan(0, 0, 0, 10),
secure='true',domain=".domain.com"}>
<cfset cookieast = {timeout=createTimeSpan(0, 0, 00, 10)}>
<cfapplication name="sessionCookies_appcfm_allSetting"
sessionmanagement="Yes" sessiontimeout="#createTimeSpan(0,0,03,0)#"
scriptprotect="all" sessioncookie=#cookiest# authcookie=#cookieast#>
```

> **Note**
>
> The application level setting takes precedence over the server level setting.

Use the following new admin APIs to set session cookies at the server level by providing the parameters `getRuntimeProperty` and `setRuntimeProperty`. These methods are available in the `CFIDE\adminapi\runtime.cfc` file.The following example explains how to get the cookie parameters using the `getRuntimeProperty`() method. Set the cookie parameters in the similar way using the `setRuntimeProperty`() method.

```
GetRuntimeProperty("HttpOnlySessionCookie");
GetRuntimeProperty("SecureSessionCookie");
GetRuntimeProperty("SessionCookieTimeout");
GetRuntimeProperty("SessionCookieDomain");
```

- The session cookies can be set at the application level by specifying the following in the `Application.cfc`:
  - `this.sessioncookie.httponly="true"`
  - `this.sessioncookie.secure="true"`
  - `this.sessioncookie.domain="value"`
  - `this.sessioncookie.timeout="value"` (days)
  - `this.authcookie.timeout= "value"`(-1 by default. Cookie is valid until the browser is open.)

> **Note**
>
> You can define the `SetDomainCookies` property and set session cookies for domain at application and server level. In this case, the precedence is as follows: application settings, server settings, and the `SetDomianCookies` property.

© 2014 Adobe Systems Incorporated. All rights reserved.

Note

The system property, `coldfusion.sessioncookie.httponly=true`, that was added in ColdFusion 9.01 is not required in this release and therefore has been removed.

Note

Using CFCookie and CFHeader tags to manipulate ColdFusion cookie and authorization cookie can be controlled in application or server level configuration. Add the following in `application.cfc` or `application.cfm`: `sessioncookie.disableupdate=true` and `authcookie.disableupdate=true`. You can also use the following methods in the `CFIDE\adminapi\runtime.cfc`: `GetRuntimeProperty("CFInternalCookieDisableUpdate")` and {{SetRuntimeProperty("CFInternalCookieDisableUpdate", "true/false")}}To set the tags in the ColdFusion administrator, go Server Settings > Memory Variables > Session Cookies Settings. Select or deselect "Disable Updating ColdFusion internal cookies using ColdFusion Tags\Function ."

- The `<cflogin>` tag stores the password in cache. For longer authenticated sessions, you can enable `diskPeristent` by modifying `authcache` located in the `cfhome/lib/auth-ehcache.xml` file. The directory used for persistence should be secured.

CRLF attack

Carriage Return (ASCII 13, \r) Line Feed (ASCII 10, \n) (CRLF) attacks also referred as HTTP Response Splitting. Here, an attacker injects CRLF to an http stream. It is commonly done by modifying an HTTP parameter or URL. In this way, CRLF can be injected into an application and can be included in response. The CRLF interpreted by proxies and caches create serious security issues.

- Protection is added against CRLF attacks for the tags which create a header, for example, `cfheader`, `cfcontent`, `cfmail`, `cfmailpart`, and `cfmailparam`.

Information disclosure

This feature improves security-related issues on information disclosure.

- Passwords for all services are encrypted in this version. Change password seed only when the server is running without any load. Otherwise, you face unexpected behavior of the server.

New HMAC method

Hash-based Message Authentication Code (HMAC) is used to verify the data integrity and authenticity of a message transmitted. It involves a cryptographic hash function in combination with a secret key. The cryptographic hash function can be Message Digest 5 (MD5), and Secure Hash Algorithm (SHA), and so on.

cfcookie support in CFScript

Cookies can be set as a struct in CFScript. You can set the following parameters:

- expires
- value
- name
- secure
- httponly
- domain

© 2014 Adobe Systems Incorporated. All rights reserved.

- path
- preservecase
- encodevalue

Example 1

```
<cfscript >
cookie.mytest =
{value="Adobe",expires="10",secure="true",domain=".adobe.com",path="/coldfusion"};
</cfscript>
```

Example 2

```
<cfscript>
 cookie_example = structNew();
 cookie_example.value = "example";
 cookie_example.expires = "10";
 cookie_example.secure = "true";
 cookie.mycookie = cookie_example;
</cfscript>
```

Miscellaneous Changes

- The httponly cookies support is available on Tomcat supporting J2EE 1.6.
- A new parameter, `numIteration`, is added to the `hash()` method.This optional parameter specifies the number of times the hash is iterated. The updated `hash()`method is as follows:

```
hash(Object message, String algorithm, String encoding, int no-of-iterations)
```

The first argument can be an object of String or Byte type.

Example

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<!--- Do the following if the form is submitted. --->
<cfif IsDefined("Form.UserID")>
<!--- query the data base. --->
<cfquery name = "CheckPerson" datasource = "cfdocexamples">
 SELECT PasswordHash FROM SecureData WHERE UserID = <cfqueryparam value =
"#Form.userID#" cfsqltype = 'CF_SQL_VARCHAR'>
</cfquery>
<!--- Compare query PasswordHash field and the hashed form password  and
display the results. --->
<cfoutput>
 <cfif Hash(Form.password, "SHA","",4) is not checkperson.passwordHash> User
ID  #Form.userID# or password is not valid. Try again.
 <cfelse> Password is valid for User ID #Form.userID#.
 </cfif>
</cfoutput>
</cfif>
<!--- Form for entering ID and password. --->
<form action="#CGI.SCRIPT_NAME#" method="post">
<b>User ID: </b> <input type = "text" name="UserID" ><br>
<b>Password: </b> <input type = "text" name="password" ><br><br>
<input type = "Submit" value = "Encrypt my String"> </form>
```

- Strengthened `<cflogin>`and authorization cookies. In a clustered environment, enable sticky session. If sticky session is not enabled, do the following:

> **Note**
>
> Different ways of adding distributed cache can be found at the Ehcache website.

-
  - Configure authentication cache for the clustered environment. Do the following for each instance in the cluster:
  1. Open `CF_instance/lib/auth-ehcache.xml`.
  2. Search for the string, `Mandatory Default Cache configuration`and add the following entry:

```
<!-- distributed caching settings part 1 starts -->
<cacheManagerPeerProviderFactory
 class="net.sf.ehcache.distribution.RMICacheManagerPeerProviderFactory"
 properties="peerDiscovery=automatic, multicastGroupAddress=230.0.0.1,
 multicastGroupPort=4446, timeToLive=32"/>
<cacheManagerPeerListenerFactory
class="net.sf.ehcache.distribution.RMICacheManagerPeerListenerFactory"
 properties="port=40002, socketTimeoutMillis=3000"/>
<!-- distributed caching settings part 1 ends -->
```

  3. In the above entry, update the `cacheManagerPeerListenerFactory` properties port. It must be unique for each instance.
  4. Search for the string, `<cache name="authcache"`.
  5. Add the following entry after `clearOnFlush="true">`.

© 2014 Adobe Systems Incorporated. All rights reserved.

```
clearOnFlush="true">
<cacheEventListenerFactory
class="net.sf.ehcache.distribution.RMICacheReplicatorFactory"
properties="replicateAsynchronously=false, replicatePuts=true,
replicatePutsViaCopy=false, replicateUpdates=true,
replicateUpdatesViaCopy=true, replicateRemovals=true"
propertySeparator=","/>
</cache>
```

Note

ColdFusion administrator does not support cluster setup.

Note

For `Remember Me` type of functionalities or for keeping authentication cache alive for a long time, change the authentication cache settings. For example, increase time outs, enable persistent cache, and so on.

Note

Use new cookie configuration for more secured authentication. Depending on the requirement, do the configuration at the server lever or application level.

- You are logged out from one of the ColdFusion administrators, if:
  - From the same host, you log in to the ColdFusion (10) Administrator and the ColdFusion Administrator of an older version.
- For a user with RDS access, in the ColdFusion Administrator, you can set the data source and secured file path permissions.
- The default values for the new sandbox are changed to make it more secure.

#back to top

© 2014 Adobe Systems Incorporated. All rights reserved.

Developing Globalized Applications

Adobe ColdFusion lets you develop dynamic applications for the Internet. Users from different countries and geographical areas access many ColdFusion applications. One design detail that you must consider is the globalization of your application so that you can best serve customers in different areas.

© 2014 Adobe Systems Incorporated. All rights reserved.

Introduction to globalization

Globalization lets you create applications for all of your customers in all the languages that you support. In some cases, globalization can let you accept data input using a different character set than the one you used to implement your application. For example, you can create a website in English that lets customers submit form data in Japanese. Or, you can allow a request URL to contain parameter values entered in Korean.
Your application also can process data containing numeric values, dates, currencies, and times. Each of these types of data can be formatted differently for different countries and regions.
You can also develop applications in languages other than English. For example, you can develop your application in Japanese so that the default character encoding is Shift-JIS, your ColdFusion pages contain Japanese characters, and your interface displays in Japanese.
Globalizing your application requires that you perform one or more of the following actions:

- Accept input in more than one language.
- Process dates, times, currencies, and numbers formatted for multiple locales.
- Process data from a form, database, HTTP connection, e-mail message, or other input formatted in multiple character sets.
- Create ColdFusion pages containing text in languages other than English.

Defining globalization

You might probably find several different definitions for globalization. Here, globalization is defined as an architectural process where you place as much application functionality as possible into a foundation that can be shared among multiple languages.
Globalization is composed of the following two parts:

- **Internationalization** Developing language-neutral application functionality that can recognize, process, and respond to data regardless of its representation. That is, whatever the application can do in one language, it can also do in another. For example, think of copying and pasting text. A copy and paste operation should not be concerned with the language of the text it operates on. For a ColdFusion application, you might have processing logic that performs numeric calculations, queries a database, or performs other operations, independent of language.
- **Localization** Taking shared, language-neutral functionality, and applying a locale-specific interface to it. Sometimes this interface is referred to as a *skin*. For example, you can develop a set of menus, buttons, and dialog boxes for a specific language, such as Japanese, that represents the language-specific interface. You then combine this interface with the language-neutral functionality of the underlying application. As part of localization, you create the functionality to handle input from customers in a language-specific manner and respond with appropriate responses for that language.

Importance of globalization in ColdFusion applications

The Internet has no country boundaries. Customers can access websites from anywhere in the world, at any time, or on any date. Unless you want to lock your customers into using a single language, such as English, to access your site, consider globalization issues.
One reason to globalize your applications is to avoid errors and confusion for your customers. For example, a date in the form 1/2/2003 is interpreted as January 2, 2003 in the United States, but as February 1, 2003 in European countries.
Another reason to globalize your applications is to display currencies in the correct format. Think of how your customers would feel when they find out the correct price for an item is 15,000 American dollars, not 15,000 Mexican pesos (about 1600 American dollars).
Your website can also accept customer feedback or some other form of text input. You might want to support that feedback in multiple languages using a variety of character sets.

How ColdFusion supports globalization

© 2014 Adobe Systems Incorporated. All rights reserved.

ColdFusion is implemented in Java. As a Java application, ColdFusion uses Java globalization features. For example, ColdFusion stores all strings internally using the Unicode character set. Because it uses Unicode, ColdFusion can represent any text data from any language.
In addition, ColdFusion includes many tags and functions designed to support globalizing your applications. You can use these tags and functions to set locales, convert date and currency formats, control the output encoding of ColdFusion pages, and perform other actions.

Character sets, character encodings, and locales

When you discuss globalization issues, two topics that you must consider are the character sets or character encodings recognized by the application and the locales for which the application must format data.
A *character set* is a collection of characters. For example, the Latin alphabet is the character set that you use to write English, and it includes all of the lower- and uppercase letters from A to Z. A character set for French includes the character set used by English, plus special characters such as “é,” “à,” and “ç.”
The Japanese language uses three alphabets: Hiragana, Katakana, and Kanji. Hiragana and Katakana are phonetic alphabets that each contain 46 characters plus two accents. Kanji contains Chinese ideographs adapted to the Japanese language. The Japanese language uses a much larger character set than English because Japanese supports more than 10,000 different characters.
In order for a ColdFusion application to process text, the application must recognize the character set used by the text. The *character encoding* maps between a character set definition and the digital codes used to represent the data.
In general use, the terms character set (or charset) and character encoding are often used interchangeably, and most often a specific character encoding encodes one character set. However, this is not always true; for example, there are multiple encodings of the Unicode character set. For more information on character encodings, see About character encodings.

> **Note**
>
> ColdFusion uses the term charset to indicate character encoding in some attribute names, structure field keys, and function parameter names.

A *locale* identifies the exact language and cultural settings for a user. The locale controls how dates and currencies are formatted, how to display time, and how to display numeric data. For example, the locale English (US) determines that a currency value displays as:
$100,000.00
while a locale of Portuguese (Portugese) displays the currency as:
R$ 100.000
To correctly display date, time, currency, and numeric data to your customers, you must know the customer's locale. For more information on locales, see Locales.

© 2014 Adobe Systems Incorporated. All rights reserved.

About character encodings

A *character encoding* maps each character in a character set to a numeric value that a computer can represent. These numbers can be represented by a single byte or multiple bytes. For example, the ASCII encoding uses 7 bits to represent the Latin alphabet, punctuation, and control characters.
You use Japanese encodings, such as Shift-JIS, EUC-JP, and ISO-2022-JP, to represent Japanese text. These encodings can vary slightly, but they include a common set of approximately 10,000 characters used in Japanese. The following terms apply to character encodings:

- **SBCS** Single-byte character set; a character set encoded in one byte per character, such as ASCII or ISO 8859-1.
- **DBCS** Double-byte character set; a method of encoding a character set in no more than 2 bytes, such as Shift-JIS. Many character encoding schemes that are referred to as double-byte, including Shift-JIS, allow mixing of single-byte and double-byte encoded characters. Others, such as UCS-2, use 2 bytes for all characters.
- **MBCS** Multiple-byte character set; a character set encoded with a variable number of bytes per character, such as UTF-8.The following table lists some common character encodings; however, there are many additional character encodings that browsers and web servers support:

Encoding	Type	Description
ASCII	SBCS	7-bit encoding used by English and Indonesian Bahasa languages
Latin-1(ISO 8859-1)	SBCS	8-bit encoding used for many Western European languages
Shift_JIS	DBCS	16-bit Japanese encoding Note: Use an underscore character (_), not a hyphen (-) in the name in CFML attributes.
EUC-KR	DBCS	16-bit Korean encoding
UCS-2	DBCS	Two-byte Unicode encoding
UTF-8	MBCS	Multibyte Unicode encoding. ASCII is 7-bit; non-ASCII characters used in European and many Middle Eastern languages are two-byte; and most Asian characters are three-byte

The World Wide Web Consortium maintains a list of all character encodings supported by the Internet. You can find this information at www.w3.org/International/O-charset.html.Computers must often convert between character encodings. In particular, the character encodings most commonly used on the Internet are not used by Java or Windows. Character sets used on the Internet are typically single-byte or multiple-byte (including DBCS character sets that allow single-byte characters). These character sets are most efficient for transmitting data, because each character takes up the minimum necessary number of bytes. Currently, Latin characters are most frequently used on the web, and most character encodings used on the web represent those characters in a single byte. Computers, however, process data most efficiently if each character occupies the same number of bytes. Therefore, Windows and Java both use double-byte encoding for internal processing.

© 2014 Adobe Systems Incorporated. All rights reserved.

The Java Unicode character encoding

ColdFusion uses the Java Unicode Standard for representing character data internally. This standard corresponds to UCS-2 encoding of the Unicode character set. The Unicode character set can represent many languages, including all major European and Asian character sets. Therefore, ColdFusion can receive, store, process, and present text from all languages supported by Unicode.

The Java Virtual Machine (JVM) that is used to processes ColdFusion pages converts between the character encoding used on a ColdFusion page or other source of information to UCS-2. The page or data encodings that ColdFusion supports depend on the specific JVM, but include most encodings used on the web. Similarly, the JVM converts between its internal UCS-2 representation and the character encoding used to send the response to the client.

By default, ColdFusion uses UTF-8 to represent text data sent to a browser. UTF-8 represents the Unicode character set using a variable-length encoding. ASCII characters are sent using a single byte. Most European and Middle Eastern characters are sent as 2 bytes, and Japanese, Korean, and Chinese characters are sent as 3 bytes. One advantage of UTF-8 is that it sends ASCII character set data in a form that is recognized by systems designed to process only single-byte ASCII characters, while it is flexible enough to handle multiple-byte character representations.

While the default format of text data returned by ColdFusion is UTF-8, you can have ColdFusion return a page to any character set supported by Java. For example, you can return text using the Japanese language Shift-JIS character set. Similarly, ColdFusion can handle data that is in many different character sets. For more information, see *Determining the page encoding of server output* in Processing a request in ColdFusion.

Character encoding conversion issues

Because different character encodings support different character sets, you can encounter errors if your application gets text in one encoding and presents it in another encoding. For example, the Windows Latin-1 character encoding, Windows-1252, includes characters with hexadecimal representations in the range 80-9F, while ISO 8859-1 does not include characters in that range. As a result, under the following circumstances, characters in the range 80-9F, such as the euro symbol (), are not displayed properly:

- A file encoded in Windows-1252 includes characters in the range 80-9F.
- ColdFusion reads the file, specifying the Windows-1252 encoding in the `cffile` tag.
- ColdFusion displays the file contents, specifying ISO-8859 in the `cfcontent` tag.

  Similar issues can arise if you convert between other character encodings; for example, if you read files encoded in the Japanese Windows default encoding and display them using Shift-JIS. To prevent these problems, ensure that the display encoding is the same as the input encoding.

© 2014 Adobe Systems Incorporated. All rights reserved.

Locales

A *locale* identifies the exact language and cultural settings to use for a user. The locale controls how to format the following:

- Dates
- Times
- Numbers
- Currency amounts
  ColdFusion supports all locales supported by the JVM that it uses.

> **Note**
>
> Current JVM versions (through 1.4.2) do not support localized numbers such as Arabic-hindic numbers used in Arabic locales or hindic digits used in Hindi locales. ColdFusion uses Arabic numbers in all locales.

Locale names

ColdFusion supports two formats for specifying locale names: the standard Java locale names and the ColdFusion naming convention that was required through ColdFusion 6.1.

- You can specify all locales using a name consisting of the following:
  - Two lowercase letters to identify the language; for example, en for English, or zh for Chinese.
  - Optionally, an underscore and two uppercase letters to identify the regional variant of the language; for example, US for the United States, or HK for Hong Kong.For example, en_US represents United States English and es_MX represents Mexican Spanish. For a list of the Java locale identifiers supported in the Sun 1.4.2 JVM and their meanings, see http://java.sun.com/j2se/1.4.2/docs/guide/intl/locale.doc.html.
    Previous to ColdFusion MX 7, ColdFusion supported a limited set of locales, and used identifiers that consisted of the name of the language, followed, for most languages, by a regional identifier in parentheses, such as English (US) or German (Standard). ColdFusion continues to support these names; for a list, see `SetLocale` in the *CFML Reference*.
    The `Server.coldfusion.supportedlocales` variable is a comma-delimited list of the locale names that you can specify.
    ColdFusion also includes a `GetLocaleDisplayName` function that returns a locale name in a format that is meaningful to users. It lets you display the locale using words in the user's language; for example, francais (France).

Determining the locale

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 the JVM locale value explicitly in ColdFusion in the JVM Arguments field on the Java and JVM Settings page in the ColdFusion Administrator; for example:

```
-Duser.language=de -Duser.country=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

© 2014 Adobe Systems Incorporated. All rights reserved.

ColdFusion tags and functions (such as the functions that start with **LS**) format data. The last `SetLocale` fun ction 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` he ader.

- ColdFusion ignores any `SetLocale` functions that follow a `cfflush` tag.

Using the locale

The `SetLocale` function determines the default formats that ColdFusion uses to output date, time, number, and currency values. You use the `GetLocale` function to determine the current locale setting of ColdFusion, or you can use the `GetLocaleDisplayName` function to get the locale name in a format that is meaningful to users. If you have not made a call to `SetLocale`, `GetLocale` returns the locale of the JVM.
The current locale has two effects:

- When ColdFusion formats date, time, currency, or numeric output, it determines how to format the output. You can change the locale multiple times on a ColdFusion page to format information according to different locale conventions. This enables you to output a page that properly formats different currency values, for example.
- When ColdFusion returns a page to the client, it includes the HTTP `Content-Language` header. ColdFusion uses the last locale setting on the page for this information.

> **Note**
>
> In earlier versions of ColdFusion, the default locale was always English, not the operating system's locale. For the Japanese version of ColdFusion, the default was Japanese.

The following example uses the `LSCurrencyFormat` function to output the value 100,000 in monetary units for all the ColdFusion-supported locales. You can run this code to see how the locale affects the data returned to a browser.

```
<p>LSCurrencyFormat returns a currency value using the locale convention.
<!--- loop through list of locales; show currency values for 100,000 units --->
<cfloop LIST = "#Server.Coldfusion.SupportedLocales#"
 index = "locale" delimiters = ",">
<cfset oldlocale = SetLocale(locale)>
 <cfoutput><p><b><I>#locale#</I></b><br>
  Local: #LSCurrencyFormat(100000, "local")#<br>
  International: #LSCurrencyFormat(100000, "international")#<br>
  None: #LSCurrencyFormat(100000, "none")#<br>
  <hr noshade>
 </cfoutput>
</cfloop>
```

This example uses the ColdFusion variable `Server.Coldfusion.SupportedLocales`, which contains a list of all supported ColdFusion locales.

© 2014 Adobe Systems Incorporated. All rights reserved.

Processing a request in ColdFusion

When ColdFusion receives an HTTP request for a ColdFusion page, ColdFusion resolves the request URL to a physical file path and reads the file contents to parse it. A ColdFusion page can be encoded in any character encoding supported by the JVM used by ColdFusion, but need to be specified so that ColdFusion can identify it.
The content of the ColdFusion page on the server can be static data (typically HTML and plain text not processed by ColdFusion), and dynamic content written in CFML. Static content is written directly to the response to the browser, and dynamic content is processed by ColdFusion.
The default language of a website might be different from that of the person connecting to it. For example, you could connect to an English website from a French computer. When ColdFusion generates a response, the response must be formatted in the way expected by the customer. This includes both the character set of the response and the locale.
How ColdFusion determines the character set of the files that it processes, and how it determines the character set and locale of its response to the client are described as follows:

Determining the character encoding of a ColdFusion page

When a request for a ColdFusion page occurs, ColdFusion opens the page, processes the content, and returns the results back to the browser of the requestor. To process the ColdFusion page, though, ColdFusion has to interpret the page content.
One piece of information used by ColdFusion is the Byte Order Mark (BOM) in a ColdFusion page. The BOM is a special character at the beginning of a text stream that specifies the order of bytes in multibyte characters used by the page. The following table lists the common BOM values:

Encoding	BOM signature
UTF-8	EF BB BF
UTF-16 Big Endian	FE FF
UTF-16 Little Endian	FF FE

To insert a BOM character in a CFML page easily, your editor must support BOM characters. Many web page development tools support insertion of these characters, including Dreamweaver, which automatically sets the BOM based on the Page Properties Document Encoding selection.
If your page does not contain a BOM, you can use the `cfprocessingdirective` tag to set the character encoding of the page. If you insert the `cfprocessingdirective` tag on a page that has a BOM, the information specified by the `cfprocessingdirective` tag must be the same as for the BOM; otherwise, ColdFusion issues an error.
The following procedure describes how ColdFusion recognizes the encoding format of a ColdFusion page.

Determine the page encoding (performed by ColdFusion)

1. Use the BOM, if specified on the page.Adobe recommends that you use BOM characters in your files.
2. Use the `pageEncoding` attribute of the `cfprocessingdirective` tag, if specified. For detailed information on how to use this attribute, see the `cfprocessingdirective` tag in the *CFML Reference*.
3. Default to the JVM default file character encoding. By default, this is the operating system default character encoding.

Determining the page encoding of server output

Before ColdFusion can return a response to the client, it must determine the encoding to use for the data in the response. By default, ColdFusion returns character data using the Unicode UTF-8 format.

© 2014 Adobe Systems Incorporated. All rights reserved.

ColdFusion pages (.cfm pages) default to using the Unicode UTF-8 format for the response, even if you include the HTML `meta` tag in the page. Therefore, the following example does*not* modify the character set of the response:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
 "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title>Untitled Document</title>
<meta http-equiv="Content-Type"
 content="text/html;
 charset="Shift_JIS">
</head>
...
```

In this example, the response still uses the UTF-8 character set. Use the `cfcontent` tag to set the output character set.
However, within a ColdFusion page you can use the `cfcontent` tag to override the default character encoding of the response. Use the `type` attribute of the `cfcontent` tag to specify the MIME type of the page output, including the character set, as follows:

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

Note

ColdFusion also provides attributes that let you specify the encoding of specific elements, such as HTTP requests, request headers, files, and mail messages. For more information, see *Tags and functions for controlling character encoding* in Tags and functions for globalizing applications and Handling data in ColdFusion.

The rest of this chapter describes ColdFusion tags and functions that you use for globalization, and discusses specific globalization issues.

© 2014 Adobe Systems Incorporated. All rights reserved.

Tags and functions for globalizing applications

ColdFusion supplies many tags and functions that you can use to develop globalized applications.

Tags and functions for controlling character encoding

The following tags and functions let you specify the character encoding of text that ColdFusion generates and interprets:

Tag or function	Attribute or parameter	Use
`cfcontent`	type	Specifies the encoding in which to return the results to the client browser. For more information, see *Determining the page encoding of server output* in Processing a request in ColdFusion.
`cffile`	charset	Specifies how to encode data written to a file, or the encoding of a file being read. For more information, see *File data* in Handling data in ColdFusion.
`cfheader`	charset	Specifies the character encoding in which to encode the HTTP header value.
`cfhttp`	charset	Specifies the character encoding of the HTTP request.
`cfhttpparam`	mimeType	Specifies the MIME media type of a file; can positionally include the file's character encoding.
`cfmail`	charset	Specifies the character encoding of the mail message, including the headers.
`cfmailpart`	charset	Specifies the character encoding of one part of a multipart mail message.
`cfprocessingdirective`	pageEncoding	Identifies the character encoding of the contents of a page to be processed by ColdFusion. For more information, see *Determining the page encoding of server output* in Processing a request in ColdFusion.

© 2014 Adobe Systems Incorporated. All rights reserved.

`CharsetDecode`	encoding	Converts a string in the specified encoding to a binary object.
`CharsetEncode`	encoding	Converts a binary object to a string in the specified encoding.
`GetEncoding`		Returns the character encoding of text in the Form or URL scope.
`SetEncoding`	charset	Specifies the character encoding of text in the Form or URL scope. Used when the character set of the input to a form, or the character set of a URL, is not in UTF-8 encoding.
`ToBase64`	encoding	Specifies the character encoding of the string being converted to Base 64.
`ToString`	encoding	Returns a string encoded in the specified character encoding.
`URLDecode`	charset	Specifies the character encoding of the URL being decoded.
`URLEncodedFormat`	charset	Specifies the character encoding to use for the URL.

Functions for controlling and using locales

ColdFusion provides the following functions that let you specify and identify the locale and format text based on the locale:

Tag or function	Use
`GetLocale`	Returns the current locale setting.
`GetLocaleDisplayName`	Returns the name of a locale in the language of a specific locale. The default value is the current locale in the locale's language.
`LSCurrencyFormat`	Converts numbers into a string in a locale-specific currency format. For countries that use the euro, the result depends on the JVM version.
`LSDateFormat`	Converts the date part of a date/time value into a string in a locale-specific date format.

© 2014 Adobe Systems Incorporated. All rights reserved.

`LSEuroCurrencyFormat`	Converts a number into a string in a locale-specific currency format. Formats using the euro for all countries that use euro as the currency.
`LSIsCurrency`	Determines whether a string is a valid representation of a currency amount in the current locale.
`LSIsDate`	Determines whether a string is a valid representation of a date/time value in the current locale.
`LSIsNumeric`	Determines whether a string is a valid representation of a number in the current locale.
`LSNumberFormat`	Converts a number into a string in a locale-specific numeric format.
`LSParseCurrency`	Converts a string that is a currency amount in the current locale into a formatted number. For countries that use the euro, the result depends on the JVM version.
`LSParseDateTime`	Converts a string that is a valid date/time representation in the current locale into a date-time object.
`LSParseEuroCurrency`	Converts a string that is a currency amount in the current locale into a formatted number. Requires euro as the currency for all countries that use the euro.
`LSParseNumber`	Converts a string that is a valid numeric representation in the current locale into a formatted number.
`LSTimeFormat`	Converts the time part of a date/time value into a string in a locale-specific date format.
`SetLocale`	Specifies the locale setting.

Note

Many functions that have names starting with **LS** have corresponding functions that do not have this prefix: `DateFormat`, `IsDate`, `IsNumeric`, `NumberFormat`, `ParseDateTime`, and `TimeFormat`. These function use English (US) locale rules.

If you do not precede calls to the LS functions with a call to the `SetLocale` function, they use the locale defined by the JVM, which typically is the locale of the operating system.
The following example uses the `LSDateFormat` function to display the current date in the format for each locale supported by ColdFusion:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<!--- This example shows LSDateFormat --->
<html>
<head>
<title>LSDateFormat Example</title>
</head>
<body>
<h3>LSDateFormat Example</h3>
<p>Format the date part of a date/time value using the locale convention.
<!--- loop through a list of locales; show date values for Now()--->
<cfloop list = "#Server.Coldfusion.SupportedLocales#"
 index = "locale" delimiters = ",">
 <cfset oldlocale = SetLocale(locale)>

 <cfoutput><p><B><I>#locale#</I></B><br>
  #LSDateFormat(Now(), "mmm-dd-yyyy")#<br>
  #LSDateFormat(Now(), "mmmm d, yyyy")#<br>
  #LSDateFormat(Now(), "mm/dd/yyyy")#<br>
  #LSDateFormat(Now(), "d-mmm-yyyy")#<br>
  #LSDateFormat(Now(), "ddd, mmmm dd, yyyy")#<br>
  #LSDateFormat(Now(), "d/m/yy")#<br>
  #LSDateFormat(Now())#<br>
  <hr noshade>
 </cfoutput>
</cfloop>
</body>
</html>
```

Additional globalization tags and functions

In addition to the tags and functions that are specifically for globalized applications, you might find the following useful when writing a globalized application:

- All string manipulation functions. For more information, see the String functions list in ColdFusion Functions in the *CFML Reference*.
- The `GetTimeZoneInfo` function, which returns the time zone of the operating system.

© 2014 Adobe Systems Incorporated. All rights reserved.

Handling data in ColdFusion

Many of the issues involved with globalizing applications deal with processing data from the various sources supported by ColdFusion, including the following:

- General character encoding issues
- Locale-specific content
- Input data from URLs and HTML forms
- File data
- Databases
- E-mail
- HTTP
- LDAP
- WDDX
- COM
- CORBA
- Searching and indexing

General character encoding issues

Applications developed for earlier versions of ColdFusion that assumed that the character length of a string was the same as the byte length might produce errors in ColdFusion. The byte length of a string depends on the character encoding.

Locale-specific content

Generating multilocale content

In an application that supports users in multiple locales and produces output that is specific to multiple locales, you call the `SetLocale` function in every request to set the locale for that specific request. When processing has completed, the locale should be set back to its previous value. One useful technique is to save the user's desired locale in a Session variable once the user has selected it, and use the Session variable value to set the locale for each user request during the session.

Supporting the euro

The euro is the currency of many European countries, and ColdFusion supports the reading and writing of correctly formatted euro values. Unlike other supported currencies, the euro is not tied to any single country (or locale). The LSCurrencyFormat and LSParseCurrency functions rely on the underlying JVM for their operations, and the rules used for currencies depend on the JVM. For Sun JVMs, the 1.3 releases did not support euros and used the older country-specific currencies. The 1.4 releases use euros for all currencies that are in the euro zone as of 2002. If you are using a JVM that does not support the euro, use the `LSEuroCurrencyFormat` and `LSParseEuroCurrency` functions to format and parse euro values in locales that use euros as their currency.

Input data from URLs and HTML forms

A web application server receives character data from request URL parameters or as form data.
The HTTP 1.1 standard only allows US-ASCII characters (0-127) for the URL specification and for message headers. This requires a browser to encode the non-ASCII characters in the URL, both address and parameters, by escaping (URL encoding) the characters using the "%xx" hexadecimal format. URL encoding, however, does not determine how the URL is used in a web document. It only specifies how to encode the URL.
Form data uses the message headers to specify the encoding used by the request (Content headers) and the encoding used in the response (Accept headers). Content negotiation between the client and server uses this information.
There are several techniques for handling both URL and form data entered in different character encodings.

© 2014 Adobe Systems Incorporated. All rights reserved.

Handling URL strings

URL requests to a server often contain name-value pairs as part of the request. For example, the following URL contains name-value pairs as part of the URL:
http://company.com/prod_page.cfm?name=Stephen;ID=7645
As discussed previously, URL characters entered using any character encoding other than US-ASCII are URL-encoded in a hexadecimal format. However, by default, a web server assumes that the characters of a URL string are single-byte characters.
One common method used to support non-ASCII characters within a URL is to include a name-value pair within the URL that defines the character encoding of the URL. For example, the following URL uses a parameter called *encoding* to define the character encoding of the URL parameters:
http://company.com/prod_page.cfm?name=Stephen;ID=7645;encoding=Latin-1
Within the prod_page.cfm page, you can check the value of the encoding parameter before processing any of the other name-value pairs. This guarantees that you handle the parameters correctly.
You can also use the SetEncoding function to specify the character encoding of URL parameters. The `SetEncoding` function takes two parameters: the first specifies a variable scope and the second specifies the character encoding used by the scope. Since ColdFusion writes URL parameters to the URL scope, you specify "URL" as the scope parameter to the function.
For example, if the URL parameters are passed using Shift-JIS, you could access them as follows:

```
<cfscript>
 setEncoding("URL", "Shift_JIS");
 writeoutput(URL.name);
 writeoutput(URL.ID);
</cfscript>
```

Note

To specify the Shift-JIS character encoding, use the Shift_JIS attribute, with an underscore (_), not a hyphen (-).

Handling form data

The HTML `form` tag and the ColdFusion `cfform` tag let users enter text on a page, then submit that text to the server. The form tags are designed to work only with single-byte character data. Since ColdFusion uses 2 bytes per character when it stores strings, ColdFusion converts each byte of the form input into a two-byte representation. However, if a user enters double-byte text into the form, the form interprets each byte as a single character, rather than recognize that each character is 2 bytes. This corrupts the input text, as the following example shows:

1. A customer enters three double-byte characters in a form, represented by 6 bytes.
2. The form returns the six bytes to ColdFusion as six characters. ColdFusion converts them to a representation using 2 bytes per input byte for a total of 12 bytes.
3. Outputting these characters results in corrupt information displayed to the user.
   To work around this issue, use the `SetEncoding` function to specify the character encoding of input form text. The `SetEncoding` function takes two parameters: the first specifies the variable scope and the second specifies the character encoding used by the scope. Since ColdFusion writes form parameters to the Form scope, you specify "Form" as the scope parameter to the function. If the input text is double-byte, ColdFusion preserves the two-byte representation of the text.
   The following example specifies that the form data contains Korean characters:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cfscript>
 setEncoding("FORM", "EUC-KR");
</cfscript>
<h1> Form Test Result </h1>
<strong>Form Values :</strong>

<cfset text = "String = #form.input1# , Length = #len(Trim(form.input1))#">
<cfoutput>#text#</cfoutput>
```

File data

You use the cffile tag to write to and read from text files. By default, the `cffile` tag assumes that the text that you are reading, writing, copying, moving, or appending is in the JVM default file character encoding, which is typically the system default character encoding. For `cffile action="Read"`, ColdFusion also checks for a byte order mark (BOM) at the start of the file; if there is one, it uses the character encoding that the BOM specifies.

Problems can arise if the file character encoding does not correspond to JVM character encoding, particularly if the number of bytes used for characters in one encoding does not match the number of bytes used for characters in the other encoding.

For example, assume that the JVM default file character encoding is ISO 8859-1, which uses a single byte for each character, and the file uses Shift-JIS, which uses a two-byte representation for many characters. When reading the file, the `cffile` tag treats each byte as an ISO 8859-1 character, and converts it into its corresponding two-byte Unicode representation. Because the characters are in Shift-JIS, the conversion corrupts the data, converting each two-byte Shift-JIS character into two Unicode characters.

To enable the `cffile` tag to correctly read and write text that is not encoded in the JVM default character encoding, you can pass the `charset` attribute to it. Specify as a value the character encoding of the data to read or write, as the following example shows:

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

Databases

ColdFusion applications access databases using drivers for each of the supported database types. The conversion of client native language data types to SQL data types is transparent and is done by the driver managers, database client, or server. For example, the character data (SQL CHAR, VARCHAR) you use with JDBC API is represented using Unicode-encoded strings.

Database administrators configure data sources and usually are required to specify the character encodings for character column data. Many of the major vendors, such as Oracle, Sybase, and Informix, support storing character data in many character encodings, including Unicode UTF-8 and UTF-16.

The database drivers supplied with ColdFusion correctly handle data conversions from the database native format to the ColdFusion Unicode format. You do not have to perform any additional processing to access databases. However, always check with your database administrator to determine how your database supports different character encodings.

E-mail

ColdFusion sends e-mail messages using the cfmail, cfmailparam, and cfmailpart tags.

© 2014 Adobe Systems Incorporated. All rights reserved.

By default, ColdFusion sends mail in UTF-8 encoding. You can specify a different default encoding on the Mail page in the ColdFusion Administrator, and you can use the `charset` attribute of the `cfmail` and `cfmailpart` tags to specify the character encoding for a specific mail message or part of a multipart mail message.

HTTP

ColdFusion supports HTTP communication using the cfhttp and cfhttpparam tags and the GetHttpRequestData function.
The `cfhttp` tag supports making HTTP requests. The `cfhttp` tag uses the Unicode UTF-8 encoding for passing data by default, and you can use the `charset` attribute to specify the character encoding. You can also use the `cfhttpparam` tag `mimeType` attribute to specify the MIME type and character set of a file.

LDAP

ColdFusion supports LDAP (Lightweight Directory Access Protocol) through the cfldap tag. LDAP uses the UTF-8 encoding format, so you can mix all retrieved data with other data and safely manipulated it. No extra processing is required to support LDAP.

WDDX

ColdFusion supports the cfwddx tag. ColdFusion stores WDDX (Web Distributed Data Exchange) data as UTF-8 encoding, so it automatically supports double-byte character encodings. You do not have to perform any special processing to handle double-byte characters with WDDX.

COM

ColdFusion supports COM through the cfobject `type="com"` tag. All string data used in COM interfaces is constructed using wide characters (wchars), which support double-byte characters. You do not have to perform any special processing to interface with COM objects.

CORBA

ColdFusion supports CORBA through the cfobject `type="corba"` tag. The CORBA 2.0 interface definition language (IDL) basic type "String" used the Latin-1 character encoding, which used the full 8-bits (256) to represent characters.
As long as you are using CORBA later than version 2.0, which includes support for the IDL types wchar and wstring, which map to Java types char and string respectively, you do not have to do anything to support double-byte characters.
However, if you are using a version of CORBA that does not support wchar and wstring, the server uses char and string data types, which assume a single-byte representation of text.

Searching and indexing

ColdFusion supports Verity search through the cfindex, cfcollection, and cfsearch tags. To support multilingual searching, the ColdFusion product CD-ROM includes the Verity language packs that you install to support different languages.

#back to top

© 2014 Adobe Systems Incorporated. All rights reserved.

Debugging and Troubleshooting Applications

Adobe ColdFusion provides detailed debugging information to help you resolve problems with your application. You configure ColdFusion to provide debugging information, and use the `cftrace` and `cftimer` tags to provide detailed information on code execution. You can also use tools for validating your code before you run it and troubleshoot particular problems.

Note

Adobe Dreamweaver provides integrated tools for displaying and using ColdFusion debugging output. For information on using these tools, see the Dreamweaver online Help.

© 2014 Adobe Systems Incorporated. All rights reserved.

Configuring debugging in the ColdFusion Administrator

ColdFusion can provide important debugging information for every application page requested by a browser. The ColdFusion Administrator lets you specify which debugging information to make available and how to display it. The Administrator settings briefly described. For more information, see the online Help for the Debugging pages.

Debugging Settings page

In the Administrator, the following options on the Debugging Settings page determine the information that ColdFusion displays in debugging output:

Option	Description
Enable Robust Exception Information	Enables the display of the following information when ColdFusion displays the exception error page. (Cleared by default.) • Path and URL of the page that caused the error • Line number and short snippet of the code where the error was identified • Any SQL statement and data source • Java stack trace
Enable Debugging	Enables debugging output. When this option is cleared, no debugging information is displayed, including all output of `cftrace` and `cftimer` calls. (Cleared by default.)You should disable debugging output on production servers. Doing so increases security by ensuring that users cannot see debugging information. It also improves server response times. You can also limit debugging output to specific IP addresses; for more information, see Debugging IP Addresses page.
Select Debugging Output Format	Determines how to display debugging output: • The classic.cfm template (the default) displays information as plain HTML text at the bottom of the page. • The dockable.cfm template uses DHTML to display the debugging information using an expanding tree format in a separate window. This window can be either a floating pane or docked to the browser window. For more information on the dockable output format, see *Using the dockable.cfm output format* in Using debugging information from browser pages.
Report Execution Times	Lists ColdFusion pages that run as the result of an HTTP request and displays execution times, ColdFusion also highlights in red pages with processing times greater than the specified value, and you can select between a summary display or a more detailed, tree structured, display.

© 2014 Adobe Systems Incorporated. All rights reserved.

General Debug Information	Displays general information about the request: ColdFusion Version, Template, Time Stamp, User Locale, User Agent, User IP, and Host Name.
Database Activity	Displays debugging information about access to SQL data sources and stored procedures. (Selected by default.)
Exception information	Lists all ColdFusion exceptions raised in processing the request. (Selected by default.)
Tracing information	Displays an entry for each cftrace tag. When this option is cleared, the debugging output does not include tracing information, but the output page does include information for `cftrace` tags that specify `inline="Yes"`. (Selected by default.)For more information on using the `cftrace` tag, see Using the cftrace tag to trace execution.
Variables	Enables the display of ColdFusion variable values. When this option is cleared, disables display of all ColdFusion variables in the debugging output. (Selected by default.)When enabled, ColdFusion displays the values of variables in the selected scopes. You can select to display the contents of any of the ColdFusion scopes except Variables, Attributes, Caller, and ThisTag. To enhance security, Application, Server, and Request variable display is disabled by default,
Enable Performance Monitoring	Allows the standard NT Performance Monitor application to display information about a running ColdFusion application server.
Enable CFSTAT	Enables you to use of the `cfstat` command line utility to monitor real-time performance. This utility displays the same information that ColdFusion writes to the NT System Monitor, without using the System Monitor application. For information on the `cfstat` utility, see *Configuring and Administering ColdFusion*.

Debugging IP addresses page

By default, when you enable debugging output, the output is visible only to local users (that is, via IP address 127.0.0.1). You can specify additional IP addresses whose users can see debugging output, or even disable output to local users. In the Administrator, use the Debugging IPs page to specify the addresses that can receive debugging messages.

© 2014 Adobe Systems Incorporated. All rights reserved.

Note

If you must enable debugging on a production server, for example to help locate the cause of a difficult problem, use the Debugging IP Addresses page to limit the output to your development systems and prevent clients from seeing the debugging information.

© 2014 Adobe Systems Incorporated. All rights reserved.

Using debugging information from browser pages

The ColdFusion debugging output that you configure in the Administrator displays whenever an HTML request completes. It represents the server conditions at the end of the request. For information on displaying debugging information while a request is processed, see Using the cftrace tag to trace execution.
The dockable.cfm debugging output format shows the debugging output in collapsed format. The next sections show each of the debugging sections and describe how you can use the information they display.

General debugging information

ColdFusion displays general debugging information. In the classic.cfm output format, the information appears at the top of the debugging output and has no heading.
The general debugging information includes the following values. The table lists the names used in the classic output template view.

Name	Description
ColdFusion	The ColdFusion version.
Template	The requested template. (In the dockable.cfm format, this appears in the Page Overview section and is called Page.)
TimeStamp	The time the request was completed. (In the dockable.cfm format, this appears in the Page Overview section and is called Date.)
Locale	The locality and language that determines how information is processed, particularly the message language.
User Agent	The identity of the browser that made the HTTP request.
Remote IP	The IP address of the client system that made the HTTP request.
Host Name	The name of the host running the ColdFusion server that executed the request.

Execution Time

The Execution Time section displays the time required to process the request. It displays information about the time required to process all pages required for the request, including the Application.cfc, Application.cfm, and OnRequestEnd.cfm pages, if used, and any CFML custom tags, pages included by the `cfinclude` tag, and any ColdFusion component (CFC) pages.

> ✅ To display execution time for a specific block of code, use the `cftimer_` tag._

You can display the execution time in two formats:

© 2014 Adobe Systems Incorporated. All rights reserved.

- Summary
- Tree

> **Note**
>
> Execution time decreases substantially between the first and second time you use a page after creating it or changing it. The first time ColdFusion uses a page it compiles the page into Java bytecode, which the server saves and loads into memory. Subsequent uses of unmodified pages do not require recompilation of the code, and therefore are substantially faster.

Summary execution time format

The summary format displays one entry for each ColdFusion page processed during the request. If a page is processed multiple times it appears only once in the summary. For example, if a custom tag gets called three time in a request, it appears only once in the output.
The following table describes the display fields:

Column	Description
Total Time	The total time required to process all instances of the page and all pages that it uses. For example, if a request causes a page to be processed two times, and the page includes another page, the total time includes the time required to process both pages twice.
Avg Time	The average time for processing each instance of this page and the pages that it uses. The Avg Time multiplied by the Count equals the Total Time.
Count	The number of times the page is processed for the request.
Template	The path name of the page.

The page icon indicates the requested page.
Any page with an average processing time that exceeds the highlight value that you set on the Debugging Settings page in the ColdFusion Administrator appears in red.
The next to last line of the output displays the time that ColdFusion took to parse, compile, and load pages, and to start and end page processing. This image is not included in the individual page execution times. The last line shows the sum of all the time it took to process the request.

Tree execution time format

The tree execution time format is a hierarchical, detailed view of how ColdFusion processes each page. If a page includes or calls second page, the second page appears below and indented relative to the page that uses it. Each page appears once for each time it is used. Therefore, if a page gets called three times in processing a request, it appears three times in the tree. Therefore the tree view displays both processing times and an indication of the order of page processing.
As in the summary view, the execution times (in parentheses) show the times to process the listed page and all pages required to process the page, that is, all pages indented below the page in the tree.
By looking at this output in this image you can determine the following information:

© 2014 Adobe Systems Incorporated. All rights reserved.

- ColdFusion took 0 ms to process an Application.cfm page as part of the request.
- The requested page was tryinclude.cfm. It took 203 ms to process this page and all pages required to execute it. The code directly on this page took 71 milliseconds (203 - 93 - 16 - 23) to process.
- The mytag2.cfm page was processed three times. All processing took 93 milliseconds, and the average processing time was 31 milliseconds. (This page does not call any other pages.)
- The mytag1.cfm page was processed two times. All processing took 78 milliseconds, and the average processing time was 39 milliseconds. This time included the time to process mytag2.cfm (this tag calls the mytag2 custom tag); therefore, the code directly on the page took an average of 8 milliseconds and a total of 16 milliseconds to process.
- The includeme.cfm page took about 62 ms to process. This processing time includes the time to process the mytag1.cfm, and therefore also the time to process mytag2.cfm once. Therefore the code directly on the page took 23 milliseconds (62-39) to process.
- ColdFusion took 125 ms for processing that was not associated with a specific page.
- The total processing time was 328 milliseconds, the sum of 125 + 203.

Database Activity

In the Administrator, when Database Activity is selected on the Debugging Settings page, the debugging output includes information about database access.

SQL Queries

The SQL Queries section provides information about tags that generate SQL queries or result in retrieving a cached database query: `cfquery`, `cfinsert`, `cfgridupdate`, and `cfupdate`.
The output displays the following information:

- Page on which the query is located.
- The time when the query was made.
- Query name.
- An indicator if the result came from a cached query.
- SQL statement, including the results of processing any dynamic elements such as CFML variables and `cfqueryparam` tags. This information is useful because it shows the results of all ColdFusion processing of the SQL statement.
- Data source name.
- Number of records returned; 0 indicates no match to the query.
- Query execution time.
- Any query parameters values from `cfqueryparam` tags.

Stored Procedures

The stored procedures section displays information about the results of using the `cfstoredproc` tag to execute a stored procedure in a database management system.
The output displays the following information:

- Stored procedure name
- Data source name
- Query execution time
- Page on which the query is located.
- The time when the query was made.
- A table displaying the procedure parameters sent and received, as specified in the `cfprocparam` tags, including the `ctype`, `CFSQLType`, `value`, `variable`, and `dbVarName` attributes. The `variable` information for OUT and INOUT parameters includes the returned value.
- A table listing the procedure result sets returned, as specified in the `cfprocresult` tag.

Exceptions

© 2014 Adobe Systems Incorporated. All rights reserved.

In the Administrator, when Exception Information is selected on the Debugging Settings page, the debugging output includes a list of all ColdFusion exceptions raised in processing the application page.
The exception information includes information about any application exceptions that are caught and handled by your application code or by ColdFusion.
Exceptions represent events that disrupt the normal flow of an application. You should catch and, whenever possible, recover from foreseeable exceptions in your application, as described in Handling Errors. However, you might also want to be alerted to caught exceptions when you are debugging your application. For example, if a file is missing, your application can catch the `cffile` exception and use a backup or default file instead. If you enable exception information in the debugging output, you can immediately see when this happens.

Trace points

In the Administrator, when Tracing Information is selected on the Debugging Settings page, the debugging output includes the results of all `cftrace` tags, including all tags that display their results inline. Therefore, the debugging output contains a historical record of all trace points encountered in processing the request.
For more information on using the `cftrace` tag, see Using the cftrace tag to trace execution.

Scope variables

In the Administrator, when the Variables option and one or more variable scopes are selected on the Debugging Settings page, the debugging output displays the values of all variables in the selected scopes. The debugging output displays the values that result after all processing of the current page.
By displaying selected scope variables you can determine the effects of processing on persistent scope variables, such as application variables. This can help you locate problems that do not generate exceptions.
The Form, URL, and CGI scopes are useful for inspecting the state of a request. They let you inspect parameters that affect page behavior, as follows:

- **URL variables** Identify the HTTP request parameters.
- **Form variables** Identify the form fields posted to an action page.
- **CGI variables** Provide a view of the server environment following the request.Similarly, the Client, Session, Application, and Server scope variables show the global state of the application, and can be useful in tracing how each page affects the state of the ColdFusion persistent variables.

Using the dockable.cfm output format

The dockable.cfm output format has several features that are not included in the classic.cfm debugging display.

Application page selections

ColdFusion displays two buttons at the bottom of each page, as described in the following table:

Button	Description
Debug This page	Tells ColdFusion to display the debugging information for the selected frame. Refreshes the debug pane if you select it for the current frame (or the application does not use frames).
Floating/Docked debug pane	Toggles the display between a floating window and a pane docked to the left of the selected frame.

Debug pane features

The debug pane has the following features:

© 2014 Adobe Systems Incorporated. All rights reserved.

- You can expand and collapse each debugging information category, such as Exceptions, by clicking the plus or minus sign (+ or -) in front of each category heading. You can also expand and collapse each scope data type display in the Scoped Variables section.
- The top of the debug pane displays the URL of the application page being debugged (as identified by the cgi.script_name variable). Click this link to refresh the page and display the debugging information that results. (You can also refresh the page and debugging information by using your browser's Refresh button or key.)
- The debug pane also displays a box where you can enter a page path or URL. When you click the Go button, ColdFusion processes the page and the debug pane is updated with the debugging information for the new page.

© 2014 Adobe Systems Incorporated. All rights reserved.

Controlling debugging information in CFML

The following sections describe how you can use CFML tags and functions to display or hide debugging and tracing information.

Generating debugging information for an individual query

In the Administrator, the `cfquery` tag `debug` attribute overrides the Database Activity setting on the Debugging Settings page. The `debug` attribute has an effect *only* when debugging output is enabled on the Debugging Settings page, as follows:

- If Database Activity is selected in the Administrator, specify `debug="No"` to prevent ColdFusion from displaying the query's SQL and statistics in the debugging output.
- If Database Activity is not selected in the Administrator, specify `debug="Yes"` or `debug` to have ColdFusion display the query's SQL and statistics in the debugging output.
  For example, if Database Activity is not selected in the Administrator, you can use the following code to show the query execution time, number of records returned, ColdFusion page, timestamp, and the SQL statement sent to the data source for this query only:

```
<cfquery name="TestQuery" datasource="cfdocexamples" debug>
 SELECT * FROM TestTable
</cfquery>
```

The `debug` attribute can be useful to disable query debugging information generated by queries in custom tags that you call frequently, so that you only see the debugging information for queries in pages that call the tags.
You can also view stored procedure-specific debugging information by specifying the `debug` attribute in the `cfstoredproc` tag.

Controlling debugging output with the cfsetting tag

Use the `cfsetting` tag `showDebugOutput` attribute to turn off debugging output for a specific page. The attribute controls debugging output *only* if the Debugging Settings page in the ColdFusion Administrator enables debugging output. The default value of the attribute is Yes. The following tag suppresses all debugging output for the current page:

```
<cfsetting showDebugOutput="No">
```

You can put this tag in the initialization code of the Application.cfc file or on your Application.cfm page to suppress all debugging output for an application, and override it on specific pages by setting `showDebugOutput="Yes"` in `cfsetting` tags on those pages. Conversely, you can leave debugging on for the application, and use the `cfsetting showDebugOutput="No"` tag to suppress debugging on individual pages where the output could cause errors or confusion.
You can also use the `showDebugOutput` attribute to control debugging output if you do not have access to the ColdFusion Administrator, but only if the Administrator enables debugging.

Using the IsDebugMode function to run code selectively

The `IsDebugMode` function returns True if debugging is enabled. You can use this function in a `cfif` tag condition to selectively run code only when debugging output is enabled. The `IsDebugMode` function lets you tell ColdFusion to run *any* code in debug mode, so it provides more flexibility than the `cftrace` tag for processing and displaying

© 2014 Adobe Systems Incorporated. All rights reserved.

information.

You can use the `IsDebugMode` function to selectively log information only when debugging is enabled. Because you control the log output, you have the flexibility of silently logging information without displaying trace information in the browser. For example, the following code logs the application page, the current time, and the values of two variables to the log file MyAppSilentTrace.log when debugging is enabled:

```
<cfquery name="MyDBQuery" datasource="cfdocexamples">
 SELECT *
 FROM Employee
</cfquery>
<cfif IsDebugMode()>
 <cflog file="MyAppSilentTrace" text="Page: #cgi.script_name#,
 completed query MyDBQuery; Query Execution time:
 #cfquery.ExecutionTime# Status: #Application.status#">
</cfif>
```

If you use `cfdump` tags frequently for debugging, place them in `<cfif IsDebugMode()>` tags; for example `<cfif IsDebugMode()><cfdump var=#myVar#></cfif>`. This way you ensure that if you leave any `cfdump_` tags in production code, they are not displayed when you disable debugging output._

© 2014 Adobe Systems Incorporated. All rights reserved.

Using the cftrace tag to trace execution

The `cftrace` tag displays and logs debugging data about the state of your application at the time the `cftrace` tag executes. You use it to provide "snapshots" of specific information as your application runs.

About the cftrace tag

The `cftrace` tag provides the following information:

- A severity identifier specified by the `cftrace` tag `type` attribute
- A timestamp indicating when the `cftrace` tag executed
- The time elapsed between the start of processing the request and when the current `cftrace` tag executes.
- The time between any previous `cftrace` tag in the request and the current one. If this is the first `cftrace` ta g processed for the request, the output indicates "1st trace". ColdFusion does not display this information in inline trace output, only the log and in the standard debugging output.
- The name of the page that called the `cftrace` tag
- The line on the page where the `cftrace` call is located
- A trace category specified by the `category` attribute
- A message specified by the `text` attribute
- The name and value, at the time the `cftrace` call executes, of a single variable specified by the `var` attribut e
  A typical `cftrace` tag might look like the following:

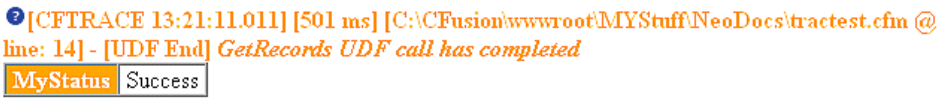
```
<cftrace category="UDF End" inline = "True" var = "MyStatus"
 text = "GetRecords UDF call has completed">
```

You can display the `cftrace` tag output in either or both of the following ways:

- **As a section in the debugging output:** To display the trace information in the debugging output, in the Administrator, select Tracing Information on the Debugging Settings page.
- **Inline in your application page**: When you specify the `inline` attribute in a `cftrace` tag, ColdFusion displays the trace output on the page at the `cftrace` tag location{{. (}}An inline `cftrace` tag does not display any output if it is inside a `cfsilent` tag block.)
  The `cftrace` tag executes only if you select Enable Debugging on the ColdFusion Administrator Debugging Settings page. To display the trace results in the debugging output, you must also specify Tracing Information on the Debugging Settings page; otherwise, the trace information is logged and inline traces are displayed, but no trace information appears in the debugging output.

> **Note**
>
> When you use inline trace tags, ColdFusion sends the page to the browser after all page processing is completed, but before it displays the debugging output from the debug template. As a result, if an error occurs after a trace tag but before the end of the page, ColdFusion might not display the trace for that tag.

The following image shows inline trace messages:



© 2014 Adobe Systems Incorporated. All rights reserved.

The following table lists the displayed information:

Entry	Meaning
?	
Trace type (severity) specified in the `cftrace` call; in this case, Information.	
[CFTRACE 13:21:11.011]	Time when the `cftrace` tag executed.
501 ms	Time taken for processing the current request to the point of the `cftrace` tag.
[C:\CFusion\wwwroot\MYStuff\mydocs\tractest.cfm]	Path in the web server of the page that contains the `cftrace` tag.
@ line:14	The line number of the `cftrace` tag.
[UDF End]	Value of the `cftrace` tag `category` attribute.
GetRecords UDF call has completed	The `cftrace` tag `text` attribute with any variables replaced with their values.
MyStatus Success	Name and value of the variable specified by the `cftrace` tag `var` attribute.

ColdFusion logs all `cftrace` output to the file logs\cftrace.log in your ColdFusion installation directory.
A log file entry looks like the following:

```
"Information","web-29","04/01/02","13:21:11","MyApp","[501 ms (1st trace)]
[C:\ColdFusion9\wwwroot\MYStuff\mydocs\tractest.cfm @ line: 14] - [UDF End]
[MyStatus = Success] GetRecords UDF call has completed "
```

This entry is in standard ColdFusion log format, with comma-delimited fields inside double-quote characters. The information displayed in the trace output is in the last, message, field.
The following table lists the contents of the trace message and the log entries. For more information on the log file format, see Logging errors with the cflog tag.

Entry	Meaning
Information	The Severity specified in the `cftrace` call.
web-29	Server thread that executed the code.
04/01/02	Date the trace was logged.

© 2014 Adobe Systems Incorporated. All rights reserved.

13:21:11	Time the trace was logged.
MyApp	The application name, as specified in a cfapplication tag.
501 ms (1st trace)]	The time ColdFusion took to process the current request up to the `cftrace` tag, This is the first `cftrace` tag processed in this request. If there had been a previous `cftrace` tag, the parentheses would contain the number of milliseconds between when the previous `cftrace` tag ran and when this tag ran.
[C:\CFusion\wwwroot\MYStuff\mydocs\tracetest.cfm @ line: 14]	Path of the page on which the trace tag is located and the line number of the `cftrace` tag on the page.
[UDF End]	Value of the `cftrace` tag `category` attribute.
[MyStatus = Success]	Name and value of the variable specified by the `cftrace` tag `var` attribute. If the variable is a complex data type, such as an array or structure, the log contains the variable value and the number of entries at the top level of the variable, such as the number of top-level structure keys.
GetRecords UDF call has completed	The `cftrace` tag `text` attribute with any variables replaced with their values.

Using tracing

As its name indicates, the `cftrace` tag is designed to help you trace the execution of your application. It can help you do any of several things:

- You can time the execution of a tag or code section. This capability is useful for tags and operations that can take substantial processing time. Typical candidates include all ColdFusion tags that access external resources, including `cfquery`, `cfldap`, `cfftp`, `cffile`, and so on. To time execution of any tag or code block, call the `cftrace` tag before and after the code you want to time.
- You can display the values of internal variables, including data structures. For example, you can display the raw results of a database query.
- You can display an intermediate value of a variable. For example, you could use this tag to display the contents of a raw string value before you use string functions to select a substring or format it.
- You can display and log processing progress. For example, you can place a `cftrace` call at the head of pages in your application or before critical tags or calls to critical functions. (Doing this could result in massive log files in a complex application, so use this technique with care.)
- If a page has many nested `cfif` and `cfelseif` tags you can place `cftrace` tags in each conditional block to trace the execution flow. When you do this, use the condition variable in the message or `var` attribute.
- If you find that the ColdFusion server is hanging, and you suspect a particular block of code (or call to a cfx tag, COM object, or other third-party component), you can place a `cftrace` tag before and after the suspect code, to log entry and exit.

Calling the cftrace tag

© 2014 Adobe Systems Incorporated. All rights reserved.

The `cftrace` tag takes the following attributes. All attributes are optional.

Attribute	Purpose
abort	A Boolean value. If you specify True, ColdFusion stops processing the current request immediately after the tag. This attribute is the equivalent of placing a `cfabort` tag immediately after the `cftrace` tag. The default is False. If this attribute is True, the output of the `cftrace` call appears only in the cftrace.log file. The line in the file includes the text "ABORTED".
category	A text string specifying a user-defined trace type category. This attribute lets you identify or process multiple trace lines by categories. For example, you could sort entries in a log according to the category.The `category` attribute is designed to identify the general purpose of the trace point. For example, you might identify the point where a custom tag returns processing to the calling page with a "Custom Tag End" category. You can also use finer categories; for example, by identifying the specific custom tag name in the category.You can include simple ColdFusion variables, but not arrays, structures, or objects, in the category text by enclosing the variable name in number signs (#).
inline	A Boolean value. If you specify True, ColdFusion displays trace output in-line in the page. The default is False.The `inline` attribute lets you display the trace results at the place that the `cftrace` call is processed. This provides a visual cue directly in the ColdFusion page display.Trace output also appears in a section in the debugging information display.
text	A text message describing this trace point. You can include simple ColdFusion variables, but not arrays, structures, or objects, in the text output by enclosing the variable name in number signs (#).
type	A ColdFusion logging severity type. The inline trace display and dockable.cfm output format show a symbol for each type. The default debugging output shows the type name, which is also used in the log file. The type name must be one of the following: -Information (default) -Warning -Error -Fatal Information

© 2014 Adobe Systems Incorporated. All rights reserved.

var	The name of a single variable that you want displayed. This attribute can specify a simple variable, such as a string, or a complex variable, such as a structure name. Do not surround the variable name in number signs.Complex variables are displayed in inline output in cfdump format; the debugging display and log file report the number of elements in the complex variable, instead of any values. You can use this attribute to display an internal variable that the page does not normally show, or an intermediate value of a variable before the page processes it further. To display a function return value, place the function inside the message. Do not use the function in the `var` attribute, because the attribute cannot evaluate functions.

Note

If you specify inline trace output, and a `cftrace` tag is inside a `cfsilent` tag block, ColdFusion does not display the trace information in line, but does include it in the standard debugging display.

The following `cftrace` tag displays the information in the example output and log entry in *About the cftrace tag* :

```
<cftrace abort="False" category="UDF End" inline = "True" text = "GetRecords UDF
 call has completed" var = "MyStatus">
```

© 2014 Adobe Systems Incorporated. All rights reserved.

Using the cftimer tag to time blocks of code

The cftimer tag displays execution time for a specified section of CFML code.

Using timing

Use this tag to determine how long it takes for a block of code to execute. This is useful when ColdFusion debugging output indicates excessive execution time, but does not pinpoint the long-running block of code.
To use this tag, enable debugging in the ColdFusion Administrator Debugging Settings page. In the Debugging Settings page, you must also specifically enable usage of the cftimer tag by checking the Timer Information check box.

> ✅ If you enable debugging for the cftimer_ tag only and display timing information in an HTML comment, you can generate timing information without disturbing production users._

Calling the cftimer tag

You can control where the cftimer tag displays timing information, as follows:

- **Inline**: Displays timing information following the </cftimer> tag.
- **Outline**: Displays timing information at the beginning of the timed code and draws a box around the timed code. (This requires browser support for the HTML FIELDSET attribute.)
- **Comment**: Displays timing information in an HTML comment in the format **<!***label*: *elapsed-time*}}**ms >.** The default label is {{cftimer.
- **Debug**: Displays timing information in the debugging output under the heading CFTimer Times.
  The following example calls the cftimer tag multiple times, each time using a different type attribute:

```
<HTML>
<body>
<h1>CFTIMER test</h1>
<!--- type="inline" --->
 <cftimer label="Query and Loop Time Inline" type="inline">
  <cfquery name="empquery" datasource="cfdocexamples">
    select *
    from Employees
  </cfquery>

  <cfloop query="empquery">
   <cfoutput>#lastname#, #firstname#</cfoutput><br>
  </cfloop>
 </cftimer>
<hr><br>
<!--- type="outline" --->
 <cftimer label="Query and CFOUTPUT Time with Outline" type="outline">
  <cfquery name="coursequery" datasource="cfdocexamples">
    select *
    from CourseList
  </cfquery>

 <table border="1" width="100%">
  <cfoutput query="coursequery">
  <tr>
  <td>#Course_ID#</td>
  <td>#CorName#</td>
  <td>#CorLevel#</td>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

```
  </tr>
  </cfoutput>
  </table>
 </cftimer>
<hr><br>
<!--- type="comment" --->
 <cftimer label="Query and CFOUTPUT Time in Comment" type="comment">
  <cfquery name="parkquery" datasource="cfdocexamples">
    select *
    from Parks
  </cfquery>
<p>Select View &gt; Source to see timing information</p>
 <table border="1" width="100%">
  <cfoutput query="parkquery">
  <tr>
  <td>#Parkname#</td>
  </tr>
  </cfoutput>
  </table>
 </cftimer>

<hr><br>
<!--- type="debug" --->
 <cftimer label="Query and CFOUTPUT Time in Debug Output" type="debug">
  <cfquery name="deptquery" datasource="cfdocexamples">
    select *
    from Departments
  </cfquery>
<p>Scroll down to CFTimer Times heading to see timing information</p>
 <table border="1" width="100%">
  <cfoutput query="deptquery">
  <tr>
  <td>#Dept_ID#</td>
  <td>#Dept_Name#</td>
  </tr>
  </cfoutput>
  </table>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

```
  </cftimer>
 </body>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

Using the Code Analyzer

The Code Analyzer has two purposes:

- It can validate the CFML syntax of your application. To do so, the analyzer runs the ColdFusion compiler on your pages, but does not execute the compiled code. It reports errors that the compiler encounters.
- It provides information about the incompatibility (and its severity), and suggests a remedy where one is required.
- It can identify places where ColdFusion might behave differently than previous versions. The analyzer identifies the following kinds of features:
  - **No longer supported**: Their use results in errors. For example, the closable attribute is not supported for the tag cflayoutarea in border layout (cflayout with `type="border"`).
  - **Deprecated**: They are still available, but their use is not recommended and they might not be available in future releases. Deprecated features might also behave differently now than in previous releases. For example, in cfcache tag the following attributes are deprecated: `directory`, `cachedirectory`, `port`, and `protocol`.
  - **Modified behavior**: They might behave differently than in previous versions. For example, if you use cfcache tag in ColdFusion 9 without end tag (`</cfcache>`), then instead of caching only the current page (which was the behavior in the previous releases), the entire request is cached.
  - **New:** These are features newly added to ColdFusion 9. For example, if you use `throw` as a user-defined function in a CFM, analyzer informs that `throw` is a built-in ColdFusion function and suggests you to rename. If you use `throw` as a user-defined function in a CFC, analyzer informs that throw is a built-in function and suggests you to prefix it with object scope. For more details on new features, see example, *What's New in ColdFusion 9*.
    You can run the Code Analyzer from the ColdFusion Administrator. Select Code Analyzer from the list of Debugging & Logging pages.

Note

The Code analyzer does not execute the pages that it checks. Therefore, it cannot detect invalid attribute combinations if the attribute values are provided dynamically at runtime.

© 2014 Adobe Systems Incorporated. All rights reserved.

Troubleshooting common problems

A few common problems that you might encounter and ways to resolve them are described here.
For more information on troubleshooting ColdFusion, see the ColdFusion Support Center Testing and Troubleshooting page at www.adobe.com/go/learn_cfu_troubleshoot_en. For common tuning and precautionary measurements that can help you prevent technical problems and improve application performance, see the ColdFusion tech tips article, TechNote number 13810. A link to the article is located near the top of the Testing and Troubleshooting page.

CFML syntax errors

Problem: You get an error message such as the following:

```
Encountered "function or tag name" at line 12, column 1.
Encountered "\"" at line 37, column 20.
Encountered "," at line 24, column 61.
Unable to scan the character '\"' which follows "" at line 38, column 53.
```

These errors typically indicate that you have unbalanced <, ", or # characters. One of the most common coding errors is to forget to close quoted code, number sign-delimited variable names, or opening tags. Make sure the code in the identified line and previous lines do not have missing characters.
The line number in the error message often does **not** identify the line that causes the error. Instead, it identifies the first line where the ColdFusion compiler encountered code that it could not handle as a result of the error.
Problem: You get an error message you do not understand.
Make sure all your CFML tags have matching end tags where appropriate. It is a common error to omit the end tag for the `cfquery`, `cfoutput`, `cftable`, or `cfif` tag.
As with the previous problem, the line number in the error message often does **not** identify the line that causes the error, but the first line where the ColdFusion compiler encounters code that it could not handle as a result of the error. Whenever you have an error message that does not appear to report a line with an error, check the code that precedes it for missing text.
Problem: Invalid attribute or value.
If you use an invalid attribute or attribute values, ColdFusion returns an error message. To prevent such syntax errors, use the CFML Code Analyzer. Also see Using the cftrace tag to trace execution.
Problem: You suspect that there are problems with the structure or contents of a complex data variable, such as a structure, array, query object, or WDDX-encoded variable.
Use the `cfdump` tag to generate a table-formatted display of the variable's structure and contents. For example, to dump a structure named `relatives`, use the following line. Surround the variable name with number signs (#).

```
<cfdump var=#relatives#>
```

Data source access and queries

Problem: You cannot make a connection to the database.
Create the data source before you can connect. Connection errors can include problems with the location of files, network connections, and database client library configuration.
Create data sources before you refer to them in your application source files. Verify that you can connect to the database by clicking the Verify button on the Data Sources page of the ColdFusion Administrator. If you are unable to make a simple connection from that page, you might need to consult your database administrator to help solve the problem.

© 2014 Adobe Systems Incorporated. All rights reserved.

Also, check the spelling of the data source name.
Problem: Queries take too long.
Copy and paste the query from the Queries section of the debugging output into the query analysis tool of your database. Then retrieve and analyze the execution plan generated by the query optimizer of the database server. (The method for doing this varies from dbms to dbms.) The most common cause of slow queries is the lack of a useful index to optimize the data retrieval. In general, avoid table scans (or "clustered index" scans) whenever possible.

HTTP/URL

Problem: ColdFusion cannot correctly decode the contents of your form submission.
The `method` attribute in forms sent to the ColdFusion server must be Post, for example:

```
<form action="test.cfm" method="Post">
```

Problem: The browser complains or does not send the full URL string when you include spaces in URL parameters.
Some browsers automatically replace spaces in URL parameters with the %20 escape sequence, but others might display an error or just send the URL string up to the first character (as does Netscape 4.7).
URL strings cannot have embedded spaces. Use a plus sign (+) or the standard HTTP space character escape sequence (%20), wherever you want to include a space. ColdFusion correctly translates these elements into a space.
A common scenario in which this error occurs is when you dynamically generate your URL from database text fields that have embedded spaces. To avoid this problem, include only numeric values in the dynamically generated portion of URLs.
Or, you can use the `URLEncodedFormat` function, which automatically replaces spaces with %20 escape sequences. For more information on the `URLEncodedFormat` function, see the *CFML Reference*.

#back to top

© 2014 Adobe Systems Incorporated. All rights reserved.

Using the ColdFusion Debugger

Adobe ColdFusion provides debugging information for individual pages. However, for complex development tasks, you require a robust and interactive debugger. ColdFusion provides a line debugger that you can use when developing ColdFusion applications in Eclipse or Adobe Flash Builder. You can set breakpoints, step over, into, or out of code, and inspect variables. You can also view ColdFusion log files.

© 2014 Adobe Systems Incorporated. All rights reserved.

About the ColdFusion Debugger

The ColdFusion Debugger is an Eclipse plugin. It runs in the Eclipse Debug perspective. You can use the ColdFusion Debugger to perform debugging tasks, including the following:

- Setting breakpoints
- Viewing variables
- Stepping over, into, and out of function calls

© 2014 Adobe Systems Incorporated. All rights reserved.

Install and uninstall the ColdFusion Debugger

To use the ColdFusion Debugger, you must have the following software installed:

- Eclipse version 3.1.2, Eclipse version 3.2, Flex Builder 2, or Flash Builder
- ColdFusion 9

To install the ColdFusion Debugger, you install the ColdFusion Eclipse plugins. For more information, see *Installing ColdFusion*.

© 2014 Adobe Systems Incorporated. All rights reserved.

Set up ColdFusion to use the Debugger

Before you can use the Debugger, enable debugging in the ColdFusion Administrator.

1. In the ColdFusion Administrator, select Debugging & Logging > Debugger Settings.
2. Enable the Allow Line Debugging option.
3. Specify the port to use for debugging if different from the default that appears.
4. Specify the maximum number of simultaneous debug session if different from the default.
5. Click Submit Changes.
6. You may have to increase the time after which requests time-out by doing the following:
   a. Select Server Settings > Settings.
   b. Enable the Timeout Requests After (Seconds) option.
   c. Enter 300 or other appropriate number in the text box.
7. The debugger server listens for commands from the Eclipse client from a separate port from the one specified in step 3. By default, ColdFusion launches the debugger server with a random available port. This could be a problem if ColdFusion (and hence debugger server) is behind a firewall and the firewall blocks the random port that the debugger is listening.. To prevent this problem, you can specify a fixed debugger server port number and allow this port in the firewall. To set a fixed debugger server port number, specify the following JVM argument on the Java and JVM page of the ColdFusion Administrator (or the appropriate place for your JEE Application Server), replacing portNumber with the port you want to use:

```
-DDEBUGGER_SERVER_PORT=portNumber
```

8. Restart ColdFusion. If you are running the JEE configuration of ColdFusion, restart the server in debug mode with the debug port as specified.
9. To modify the debug settings, in Eclipse, select Window > Preferences > ColdFusion > Debug Settings. You can specify the home page URL, which points to the page that appears in the Debug Output Buffer of the debugger when you click the Home button. You can also specify the extensions of the types of files that you can debug and variable scopes that you want the Debugger to recognize. To improve performance when debugging large files, deselect all scopes for which you do not require information.

   Note

   To ensure that the debugger stops in the template you are debugging on the line that causes a ColdFusion error, select Preferences > ColdFusion > Debug Settings and select the Enable Robust Exception Information checkbox.

10. To configure an RDS server, in Eclipse, select Window > Preferences > ColdFusion > RDS Configuration.If you are running ColdFusion on the same computer as Eclipse, localhost is configured by default. To use any additional RDS servers, enter the configuration information.
11. If ColdFusion and Eclipse are not running on the same computer, in Eclipse, select Window > Preferences > ColdFusion > Debug Mappings. Then specify the path that Eclipse uses to open files on the ColdFusion server and the path that ColdFusion uses to find the files that you are editing in Eclipse.Mapping ensures that Eclipse and ColdFusion are working on the same file. For example, if you are editing files in an Eclipse project that points to D:\MyCoolApp. Then, when you deploy the files to the ColdFusion server, you copy them to W:\websites\MyCoolSite\, which the ColdFusion server recognizes as D:\Shared\websites\MyCoolSite. The mapping in Eclipse specifies that the Eclipse directory is D:\MyCoolApp and the server is D:\Shared\websites\MyCoolSite. Eclipse translates the file path (D:\MyCoolApp\index.cfm) to a path that the ColdFusion server recognizes (D:\Shared\websites\MyCoolSite\index.cfm). To see more information about the interaction between the client and the server, add the following to the JVM arguments in the ColdFusion Administrator:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
-DDEBUGGER_TRACE=true
```

12. If you are not running the server configuration of ColdFusion, specify Java debugging parameters in the configuration file or startup script of the application server you are running. The parameters should look like the following:

    ```
    -Xdebug -Xrunjdwp:transport=dt_socket,server=y,suspend=n,address=<port_number>
    ```

    Ensure that the port number you specify is the same port number specified on the Debugger Settings page of ColdFusion Administrator.If you are running the server configuration, ColdFusion writes these debugging parameters to the jvm.config file when you use the Debugger Settings page of the ColdFusion Administrator.

13. If you are not running the server configuration and your application server is not running on JRE 1.6, copy the tools.jar file of the JDK version that your application server is running to the \lib folder of ColdFusion. For example, if you are running JRun that runs on JRE 1.4, copy the tools.jar file of JDK 1.4 to the \lib folder of ColdFusion.
14. If you are running the server version of ColdFusion and you specify a JRE version other than JRE 1.6 in the jvm.config file, copy the tools.jar file of the JDK version specified in your jvm.config file to the \lib folder of ColdFusion.

Test and modify the debugger in Eclipse

You can determine whether the Debugger server is configured correctly in Eclipse by selecting Window > Preferences > ColdFusion > RDS Configuration, and then clicking Test Debugger.
You can modify RDS configurations easily from the RDS Fileview or RDS Dataview by clicking the Edit RDS Preferences button in the view.

© 2014 Adobe Systems Incorporated. All rights reserved.

About the Debug perspective

After you install the ColdFusion Plugin, enable the debugger in ColdFusion, and configure Eclipse, you can use the ColdFusion Debugger in Eclipse. It is available in the Eclipse Debug perspective.

The Debug perspective includes the following:

- Debug pane, which keeps the results of each completed session. The following buttons appear at the top of this pane:
  - Resume - Resumes a debugging session
  - Suspend - Pauses a debugging session
  - Terminate - Stops a debugging session
  - Disconnect - Disconnects the debugger from the selected debug target when debugging remotely
  - Remove All Terminated Launches - Clears all terminated debug targets from the display
  - Step Into - Executes code line by line, including included code, UDFs, CFCs, and the like
  - Step Over - Executes code line by line, excluding included code, UDFs, CFCs, and the like
  - Step Return - Returns to the original page from which you entered the included code, UDF, CFC, or the like
  - Drop to Frame -Reenters a specified stack frame, which is analogous to going in reverse and restarting your program partway through
  - Use Step Filters/Step Debug - Ensures that all step functions apply step filters
  - Menu - Displays the menu that lets you manage the view, show system threads, show qualified names, and show monitors
- Variables pane, which shows the current variables, including the variable scope. The following buttons appear at the top of this pane:
  - Show Type Names - Displays the type of the variables
  - Show Logical Structure - This button is not supported
  - Collapse All - Collapses the information in the panel to show only variable types
- Breakpoints pane - Lists breakpoints in the ColdFusion application. The following buttons appear at the top of this pane:
  - Remove Selected Breakpoints - Removes a breakpoint
  - Remove All Breakpoints - Removes all breakpoints
  - Show Breakpoints Supported by Selected Targets - Displays the breakpoints for what you are currently debugging
  - Go to File for Breakpoint - Goes to the file in which the selected breakpoint is set
  - Skip All Breakpoints - Ignores all breakpoints
  - Expand All - Expands the information in the pane
  - Collapse All - Collapses the information in the pane
  - Link with Debug View - Highlights the selected breakpoint when the application stops execution in the Debug View
  - Add Java Exception Breakpoint - Lets you specify which Java exception to throw when you reach the selected breakpoint
  - Menu - Lets you specify the type of information to display in the Breakpoints pane
- Debug Output Buffer - Contains two panes: Browser, which displays what appears in the browser during application execution; Server Output Buffer, which displays the debug output.
- Edit pane, which displays the stacked source panes, one for each source file you have open.
- Outline pane, which displays the current source file's content in outline form

© 2014 Adobe Systems Incorporated. All rights reserved.

Using the ColdFusion Debugger-Developing guide

After you enabled the debugger in the ColdFusion Administrator and configure Eclipse, you can debug ColdFusion pages that are in an Eclipse project.

You can use the ColdFusion Debugger to do the following tasks:

- Setting a breakpoint
- Executing code line by line
- Inspecting variables

Begin debugging a ColdFusion application

1. Open the file in the Eclipse project to debug.You do not have to create an Eclipse project in the same folder as CFML source. You can create a project in a different folder, create a folder under that project, and then link it to the folder where CFML sources reside.
2. Click Debug in the upper-right corner of the Eclipse workbench to go to the Debug perspective.
3. Select Window > Show View > Debug Output Buffer to see the output from your application and how your application appears in a browser.
4. Select Window > Preferences and specify the home page for your debugging session, the extensions of the file types that you can debug, and the variable scopes of the variables to show in the Variables pane. Click OK.The home page is the page that appears in the Debug Output Buffer pane when you click the Home button in the Debug Output Buffer pane.
5. To begin debugging the file whose source appears in the Edit pane, click the Debug icon in the Eclipse toolbar.
6. Click New to create a new debugging configuration.
7. Specify the home page for the active debug session. This is the page that appears in the Debug Output Buffer pane when you click the Debug Session Home button in the Debug Output Buffer pane.
8. Click Debug to start the debug session.

> **Note**
>
> If you are in the process of debugging a template and then try to browse to or refresh that page, doing so can result in unexpected behavior in the Debugger.

Setting a breakpoint

You can set breakpoints in your CFML file to stop execution of the page at particular points. When you set a breakpoint on a line, execution of the CFML stops just before that line. For example, if you set a breakpoint on the third line in the following CFML page, execution stops before `<cfset myName = "Wilson">`.

```
<cfset yourName = "Tuckerman">
<cfoutput>Your name is #yourName#.</cfoutput>
<cfset myName = "Wilson"
```

Run the page that you want to debug before setting any breakpoints to compile it before debugging it. This improves performance during debugging. You cannot set a breakpoint in a file that is not part of a project.

1. In Eclipse, open the file in which you want to set a breakpoint.
2. While highlighting the line where you want to set the breakpoint, do one of the following:
   - Double-click in the marker bar that appears to the left of the editor area.
   - Right click, and then select Toggle Breakpoint.
   - Press Alt+Shift+B.

© 2014 Adobe Systems Incorporated. All rights reserved.

A blue dot appears before the line on which you set the breakpoint. Also, you can view a list of breakpoints set in the current Eclipse project in the Breakpoints panel.ColdFusion breakpoints have four states in the Eclipse debugger:

- Enabled and Valid - This is a breakpoint at a valid location. It is represented by a solid blue circle and stops code execution when encountered.
- Unresolved - ColdFusion sets the breakpoint for the page that is loaded in its memory. If you modify the page and do not execute it, the source is not in sync with the page that ColdFusion sees on the server. In this situation, ColdFusion may consider the line where you want to set breakpoint to be invalid. However, you have not yet executed the page; when you do so, that line may be valid. This type of breakpoint is represented by a question mark (?) icon.For performance reasons, ColdFusion does not try to resolve unresolved breakpoints every time you execute the page. It tries to resolve them when you modify the page and execute it. If you think that the line at which ColdFusion shows an unresolved breakpoint is valid, delete the breakpoint and set it again.
- Invalid - If ColdFusion determines that the CFML that you edit in Eclipse is the same as the CFML in its memory, and that the breakpoint you have set is at an invalid line, the breakpoint appears as a red X.
- Disabled.

Executing code line by line

You can use the Step Into, Step Over, and Step Return buttons to proceed through your CFML application line by line. Use Step Into to proceed into included files, such as UDFs or CFCs. Use the Step Over button to proceed through your CFML application, bypassing included files, such as UDFs or CFCs. Use the Step Return button to return to the original page from which you entered the included file, such as UDFs or CFCs.

For the stepping process to work properly, clear the cache of compiled classes. To do so, recompile all CFML pages compiled with an earlier version of ColdFusion. In large files, you might find that stepping and breakpoints are slow. To improve performance, in Eclipse, select Windows > Preferences > ColdFusion > Debug Settings and deselect all scopes for which you do not require information.

Avoid using Step In on CFML instructions such as the `cfset` tag. Step In is more performance intensive than Step Over. You can use Step In for UDFs, CFCs, custom tags, and included files.

When stepping into functions, tags, and files, Eclipse expects the file to be displayed in one of the open projects. The file that you are stepping in must be in an open Eclipse project.

Sometimes Eclipse 3.2.1 does not show the stack trace, and step buttons are disabled, even though the debugger has stopped at a line. To enable the step buttons, click the debugger server instance in the Debug window. To see the stack trace, click either Step In or Step Out.

Inspecting variables

As you observe execution of your code, you can see the values and scope of variables in the Variables panel. The Variables panel displays the scope and value of variables as the CFML code executes. Only variables whose scopes are those you selected in the Preferences dialog box appear in the Variables pane.

© 2014 Adobe Systems Incorporated. All rights reserved.

Viewing ColdFusion log files

You can easily see the contents of all the log files that ColdFusion generates by using the Log File Viewer.

1. In Eclipse, select Window > Show View > Other > ColdFusion > CF Log Viewer.
2. To view details of a log file, double-click the name of the file.
3. To include the log files in another folder, click the Add Log Folder button, select the folder, and then click OK.
4. To remove a folder from the list, without deleting it from the computer's file system, click the Remove Log Folder button, select the folder, and then click OK.
5. To remove a log file from the computer's file system, click the Delete Log File button.
6. To remove the contents of the detail pane, click the Menu button, and then click Clear Log.
7. To update the contents of the detail pane, click the Menu button, and then click Refresh Log.

© 2014 Adobe Systems Incorporated. All rights reserved.

Using Scheduler

Scheduler enhancements in ColdFusion 10 let you schedule tasks in a granular, scalable, and organized way. The enhancements include:

- **Quartz scheduling service:** For details, see http://www.quartz-scheduler.org/.
- **Grouping:**Arrange tasks in to different groups as shown here:

```
<cfschedule
 action=update
 task=t1 url="www.adobe.com"
 group="G1"/>
```

  You can club tasks into groups so that later on, you can resume or pause all the tasks in a same group rather than repeat for individual tasks.

- **Application-specific tasks:**Apart from scheduling tasks at server level, you can schedule tasks at application level, visible only to the application, as shown here:

```
<cfschedule
 action=update
 task=t1
 url="www.adobe.com"
 group="G1"
 mode="application"/>
```

  The default mode is `server`.

- **Event Handling:** Attach listeners to scheduled tasks.For example, you can write a listener CFC that does the following:
- On completion of the task, sends a mail to all stakeholders (`onTaskEnd`)
- Decide if the task should execute (`onTaskStart`)
- In the case of exception, refires the task (`onError`)
- Executes the code provided inside a method instead of invoking the URL (`execute`)

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cfcomponent implements="CFIDE.scheduler.ITaskEventHandler">
 <cffunction name="onTaskStart" returntype="boolean">
  <cfargument name="context" type="struct"/>
  <cfmail from="a@adobe.com" subject="Scheduler_Scenario_Testing"
to="a@adobe.com">
   The Report is about to be generated.
  </cfmail>
  <cfreturn true>
 </cffunction>
 <cffunction name="onMisfire" returntype="void">
  <cfargument name="context" type="struct" required="false"/>
  <cfmail from="a@adobe.com" subject="Scheduler_Scenario_Testing"
to="a@adobe.com">
   The Report generation task has misfired.
  </cfmail>
 </cffunction>
 <cffunction name="onTaskEnd" access="public" returntype="void">
  <cfargument name="context" type="struct" required="false"/>
  <cfmail from="a@adobe.com" subject="Scheduler_Scenario_Testing"
to="a@adobe.com">
   The Report generation task has Completed.
  </cfmail>
 </cffunction>
 <cffunction name="onError" returntype="void">
  <cfargument name="context" type="struct" required="false"/>
  <cfmail from="a@adobe.com" subject="Scheduler_Scenario_Testing"
to="a@adobe.com">
   The Report generation task has errored out.
  </cfmail>
 </cffunction>
  <cffunction name="execute" returntype="void">
  <cffile action="append" file="#Expandpath('.')#/log.txt" output="<br><b>In
Execute</b><br>">
 </cffunction>
</cfcomponent>
```

Note

The listener has to extend CFIDE.scheduler.ITaskEventHandler.cfc.

Note

The `context` argument of Event Handling functions is a struct and contains the following keys. For functions, `onMisfire`, `onTaskEnd` , and `onTaskStart`, the keys are `group`, `mode`, and `task`. For the `onError` function, the keys are `exceptionMessage`, `group`, `mode`, and `task`

- **Chaining:**Lets you define dependent tasks. If the parent task is executed, all dependent tasks are also executed.

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cfschedule
 action=update
 task=t1
 url="www.adobe.com"
 group="G1"
 mode="application"
 onComplete="t2:DEFAULT:application">
```

In this example, the value of `onComplete` must be provided in the following format: `<task>:<group>:<mode>`

> **Note**
>
> You cannot chain an application-specific task after a server-specific task.

- **Cluster:** You can run scheduler in cluster setups. Currently clustering works only with JDBC job store. Features include load-balancing and job fail-over. A single application can have both clustered as well as non-clustered setup. So tasks can go to either of the setups.The system time of all clustered servers have to be same. To enable cluster setup, use the ColdFusion Administrator. For details, see Scheduled Tasks.
- **Cron commands:** You can trigger a scheduled task using cron commands. Cron-Expressions are strings made up of seven sub-expressions (`Seconds`, `Minutes`, `Hours`, `Day-of-Mont}}h, {{Month`, `Day-of-Week`, and `Year` (optional field)), that describe individual details of the schedule.Sub-expression are separated with white-space.For example, `"0 */2 3-10,21-23 * * ?"` the task should be executed every 2 minutes from 3:00 AM to 10:00 AM and 9:00 PM to 11:58 PM daily.For details, see http://www.quartz-scheduler.org/docs/tutorial/TutorialLesson06.html .
- **Prioritize tasks:** You can set priorities for the tasks. Assume that you have 50 tasks that have to start at the same time. But there are only 10 worker threads currently available. Then, the first 10 tasks with the highest priority start before others. Priority can be any integer value. The default value is `5`.
- **Exclude dates**: You can exclude dates from the scheduling process.For example, you can set to execute a job from September 1 to December 30, except on December 25.For exclusion, you can specify a comma-separated list of dates, date, or date range. For example,

```
<cfschedule .... exclude="date1 TO date2"  .../>
```

```
<cfschedule  ... exclude="02/02/2011,03/03/2011"  .../>
```

All the dates from `date1` to `date2` (inclusive) are excluded from scheduling. You can provide an array of date strings or an array of date objects.

- **In case of error:**You can specify the action to be taken.

```
<cfschedule task="job1" onException="REFIRE,PAUSE,INVOKEHANDLER" ....>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

That is, if an error occurs while running the job, you can decide if to refire the task, pause the job, or invoke `onError` method of the attached eventhandler.

- **If task misfires:** You can specify the action to be taken.That is, if to refire, reschedule, or invoke `onMisfire` method of the attached eventhandler.

```
<cfschedule task="trigger1" onmisfire="FIRE_NOW, FIRENOW,
NOW_EXISTING,NOW_REMAINING,NEXT_EXISTING,NEXT_REMAINING,INVOKEHANDLER..../>
```

A misfire occurs if a persistent trigger misses the firing time because the scheduler was shutdown, or because there are no available threads in thread pool to execute the job. The threshold value for a task to be considered as misfired is 60 seconds.

> **Note**
>
> All misfired tasks are logged in scheduler.log available in cf_root\WEB-INF\cfusion\logs\scheduler.log file on J2EE configurations, or the cf_root\logs\scheduler.log file on server configurations.

> **Note**

- **Pause and resume/Pause all and resume all:**Pause or resume the tasks of whole group as follows:

```
<cfschedule group="group1" action="pause" .......>
```

```
<cfschedule group="group1" action="resume" .......>
```

This can be specified at the server-level or application level as follows:

```
<cfschedule  action="pauseall" mode=application/>
```

```
<cfschedule  action="resumeall" mode=server/>
```

- **List tasks:**Lists all scheduled tasks at the server level or application level:

```
<cfschedule  action="list" mode="application" result ="res" />
```

- **Retry:**If running a job results in an exception, you can preset to continue re-firing, till retry count is over, as follows:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cfschedule task="job1" onException="REFIRE" retryCount="3" ....>
```

- **Repeat:** Specify the number of times a given schedule has to repeat.For example, run `Job1`at 2 pm today and repeat it 20 times at the interval of 10 minutes.In this case, you need not specify the end time of the schedule.

```
<cfschedule task="task1" repeat="20" interval="60" ....>
```

- **Customize quartz:** Advanced users can customize Quartz using quartz.properties available in cfusion\lib\quartz. The current version of Quartz being shipped with ColdFusion is 2.02.

When you are using the Scheduler, output can be saved only as .log or .txt files. The same restriction is applicable for validation queries on databases.

Also, for the <cfinclude> tag, this restriction is applicable. By default, you can include only CFM files. However, you can modify allowedextinclude key in neoruntime.xml file to add your own file type.

© 2014 Adobe Systems Incorporated. All rights reserved.

Client-side CFML (for mobile development)

- The new <cfclient> tag
  - How does the transformation work
  - Supported CFML language constructs
  - Supported CFML tags
  - Where client-side CFML differs from the server-side CFML
- Client-side CFML and JavaScript
- Loading JavaScript files
- Support for ColdFusion Functions
  - Using CFML simple types
- List of all supported functions
- Support for custom tags
  - Paths for custom tags
  - Invoking custom tags
  - Passing values
  - Tag instance data
  - Aborting custom tag processing
  - Deviation list for custom tags
- Support for database queries
  - What is supported
  - What is NOT supported
  - Usage example

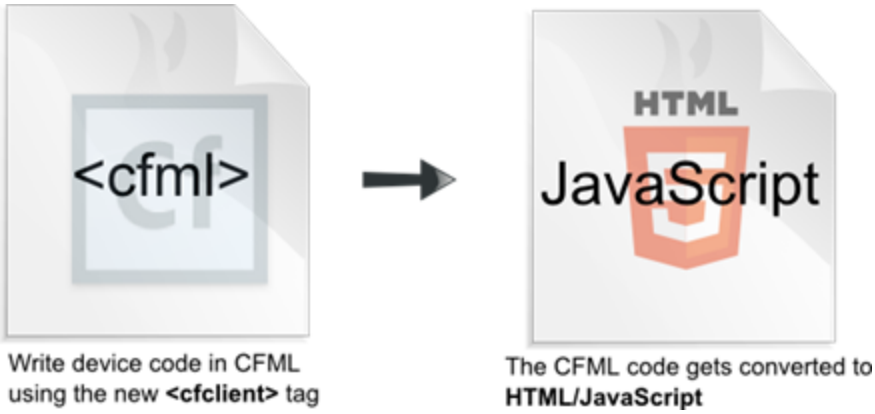
This document describes the client-side CFML capabilities. Client-side CFML allows the development of client-side applications using CFML. Client-side CFML can be used to develop CF-based mobile applications wherein the CFML code in the application is converted to HTML/JavaScript by the ColdFusion Server.

> **Before you begin** – To try out the examples provided in this document, you need to set up the ColdFusion mobile development environment. See Configuring the development environment.

The new <cfclient> tag

<cfclient> is a new tag introduced in ColdFusion 11 to support mobile development. This tag has been introduced to convert the CFML code that it encloses into JavaScript code. A ColdFusion developer can now develop mobile applications using CFML by leveraging the transformation functionality offered through the <cfclient> tag. So, you do not need to know JavaScript to write mobile web applications.

© 2014 Adobe Systems Incorporated. All rights reserved.



Even if you are an experienced JavaScript developer, <cfclient> can still be used to simplify the mobile application development as it abstracts the complexities involved in building a mobile application using JavaScript and HTML.

Note: The CFML constructs to be executed at the client-side have to be embedded within the <cfclient> tag. Not all tags, functions, and CFML functionalities are supported for conversion to HTML/JavaScript. For the complete list of CFML tags and functions that <cfclient> tag supports, see Supported CFML language constructs and Supported CFML tags.

The rationale behind choosing to support only a certain set of tags and functions is to strengthen the relevance of CFML for client-side mobile application development.

How does the transformation work

Let us see how the regular <cfoutput> tag gets rendered on a browser.

Your ColdFusion code:

```
<cfoutput>Hello World</cfoutput>
```

What the browser gets from the ColdFusion Server:

Hello World

Let us revisit the Hello World <cfclient> example mentioned in the A customary hello world example section.

Now, if your ColdFusion code is:

```
<cfclient>
     <cfset myvar = “Hello World”>
  <cfoutput>#myvar#</cfoutput>
</cfclient>
```

Check the source of the web page translated by ColdFusion Server. It will be pure JavaScript wrapped in an HTML page.

As you can infer, the CFML code available in the <cfclient> block gets converted to JavaScript. Though this example is simple, the translation works the same way for complex CFML code.

© 2014 Adobe Systems Incorporated. All rights reserved.

Supported CFML language constructs

The following CFML language constructs are supported in client-side CFML, which includes all the logical/conditional and flow constructs:

- IF/ELSE/ELSEIF
- WHILE/DO WHILE
- CFLoop/CONTINUE/BREAK
- SWITCH/CASE/DEFAULTCASE
- TRY/CATCH/FINALLY
- FOR
- TERNARY OPERATOR
- THROW
- IMPORT
- INCLUDE
- ABORT
- EXIT
- FUNCTION/ARGUMENT/RETURN
- FUNCTION INVOCATION
- CUSTOM TAGS

Supported CFML tags

The following CFML tags are supported in client-side CFML:

- CFSET
- CFOUTPUT
- CFINCLUDE
- CFSCRIPT
- CFOBJECT
- CFINVOKE
- CFMODULE
- CFSAVECONTENT
- CFPARAM
- CFPROPERTY
- CFCOMPONENT
- CFABORT
- CFEXIT
- CFRETURN
- CFBREAK
- CFCONTINUE
- CFQUERY
- CFQUERYPARAM
- CFFLUSH

Note that member functions are also supported in client-side CFML.

Where client-side CFML differs from the server-side CFML

Though you can have any valid CFML code in the <cfclient> code block, there are behavioral restrictions on the CFML tags and constructs. Some of the behavioral restrictions are listed here:

- The keys for implicit structure will be static. For instance, you cannot declare {“#a#”:”value”}. Also, {a:a} will become {‘a’:a}.

© 2014 Adobe Systems Incorporated. All rights reserved.

- In the <cfinclude> tag, dynamic template name(##) is not supported:

```
<cfclient>
      <cfset x="abc.cfm">
      <cfinclude template="#x#">
</cfclient>
```

  This limitation is applicable for the <cfmodule> tag too.

- Also, the <cfinclude> tag only supports files with extensions .cfm, .js, and .css.
- The Boolean behavior differs in <cfclient>. For example, In ColdFusion, 0/1, true/false, 'true'/'false', yes/no are all treated as Boolean. However, in <cfclient>, only true/false are Boolean.
- In ColdFusion, x="1" is still a number even with the quotes. However, <cfclient> treats this as a string. Ensure that you follow strict data types for the functions to avoid abnormal behavior.
- In ColdFusion, <cfset x="1a"> is a date but inside <cfclient> it is not. Note that <cfclient> follows JavaScript date format instead of ColdFusion date format.
- In ColdFusion, <cfset x = 1/2> is 0 but inside <cfclient> it is 0.5.
- ColdFusion server exceptions will not work on client side.
- The format of the Date/Time/DateTime objects created by createDate, createTime, and createDateTime respectively differs from the server side CFML behavior. For instance, the following code:

```
<cfoutput>#CreateDateTime( 1776, 7, 4, 12, 59, 0)#</ cfoutput >
```

  On server side, you will get the output:
  {ts '1776-07-04 12:59:00'}

  When you use parseDateTime on client side CFML, ensure that you pass the output obtained from the creatDateTime function as an argument to create the DateTime object.

- The following code will not work because of the strict data types:

```
<cfset date1 = #createdatetime(2001,07,04,6,30,45)#>
<cfset mytimespan = #createtimespan(2, 1, 16, 30)#>
<cfoutput>#date1 + mytimespan#</cfoutput>
<cfoutput>#DateFormat(date1 + mytimespan)#</cfoutput>
```

  In the above example, timespan when added to datetime, becomes a string.

- Function naming convention – Functions supported by browser and PhoneGap will have server CFML syntax. For instance, FileXXX. The PhoneGap functions will follow the Object access approach. For instance, Camera.XXX.
- Scopes available on server side is not supported on client side.
- Argument Collection will not be supported for passing arguments.
- arraySort function differs in the behavior when its numeric numbers like 0002, 00001, 1.0E+5 and the sort type is text.

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cfscript>
anumeric = arrayNew(1);
anumeric[1] = 01;
anumeric[2] = 001;
anumeric[3] = 1;
anumeric[4] = 1.001;
anumeric[5] = 1.1;
anumeric[6] = 1.101;
anumeric[7] = 1.109;
anumeric[8] = 1.11;
anumeric[9] = 2;
anumeric[10] = 02;
anumeric[11] = 00002;
anumeric[12] = 20;
anumeric[13] = 50;
anumeric[14] = 1.0E+2;
anumeric[15] = 100;
anumeric[16] = 1000;
anumeric[17] = 1.0E+5;
</cfscript>
<cfset arraySort(anumeric, "text")>
<cfloop array="#anumeric#" index=i>
<cfoutput>#i#</cfoutput>
</cfloop>
```

Actual output for the above code:
1 1 1 1.001 1.1 1.101 1.109 1.11 100 100 1000 100000 2 2 2 20 50
Expected output for the above code:
00002 001 01 02 1 1.001 1.0E+2 1.0E+5 1.1 1.101 1.109 1.11 100 1000 2 20 50
This is because JavaScript represents 02, 002, 2 in the same way as '2' and hence differs in sort.

- Duplicate function behavior for struct differs when the struct internally has a reference to another struct more than once. On server side, changing the value in the duplicated struct's referred key will also change the values at other referred points. However, in the case of client side CFML, this does not happen.

```
<cfset str1 = {name: 'str1', value: 'org struct'}>
<cfset str2 = {name: 'str2', value1: str1, value2: str1}>
<cfset str2dup = duplicate(str2)>
```

In the above example, if you change the value of str2dup.value1.value, on the server side, value of str2dup.value2.value is also changed automatically as they both refers to same structure. But on client side, this is not the behavior.

- On client side, calling a super function from an included CFM or CFC is not supported.
- Positional arguments are not supported.

When you use the <cfoutput> tag inside the <cfclient> tag, the contents of the <cfoutput> tag is not immediately processed. Hence, you may encounter certain issues while using this code:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cfclient>
 <cfoutput>
  <div id="result"></div>
 </cfoutput>
 <cfset doucment.getElementById("result").innerHTML = "Hello">
<cfclient>
```

In this case, while the document.getElementById() statement is being invoked, cfoutput is not processed. Hence, you will not find an element with id "result", which will result in a runtime error.

In this case, you can directly write to the DOM, instead of using cfoutput:

```
<div id="myDiv"></div>
<cfclient>
 <cfset document.getElementById("myDiv").innerHTML += "<div id=""result""></div>
 <cfset doucment.getElementById("result")+="Hello">
</cfclient>
```

Or another workaround is to use flush explicitly after <cfoutput>:

```
<cfclient>
 <cfoutput>
  <div id="result"></div>
 </cfoutput>
 <cfflush>
 <cfset doucment.getElementById("result").innerHTML = "Hello">
<cfclient>
```

If you follow this approach, ensure that the HTML content in the cfoutput is well formed.

Client-side CFML and JavaScript

Important: The variable names and function names in CFML are case sensitive.

From **CFML**, when you are invoking **JavaScript**:

- Use the correct case for function name and variable name when referencing CFML functions and variables from CFML as client-side CFML is case sensitive.

From **JavaScript**, when you are invoking **CFML**:

- Use the correct case for function name and variable name when referencing CFML functions and variables from JavaScript as client-side CFML is case sensitive.

Loading JavaScript files

You can load content of the JavaScript files in your ColdFusion code using the loadJSFile() function as shown in the

© 2014 Adobe Systems Incorporated. All rights reserved.

following example:

```
component client="true"
{
 function init ()
 {
  cfclient.loadJSFile("yourjsfile.js", function ()
  {
   alert("Script loaded");
  });
 }
}
```

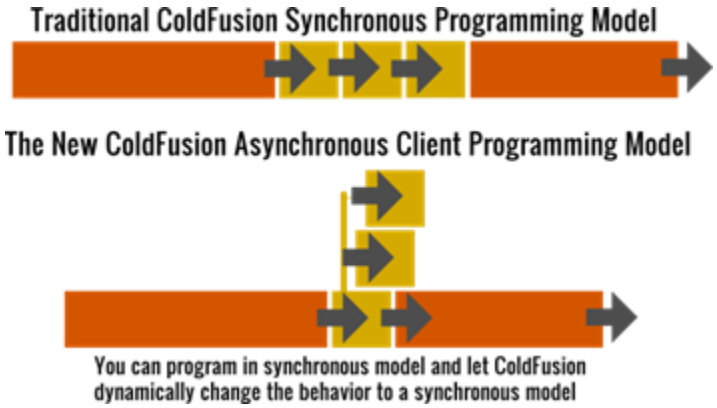
You can also use <cfinclude> to load a JavaScript file.

Synchronous and asynchronous function calls

ColdFusion automatically determines whether a function call is synchronous or asynchronous. However, if you need to invoke an asynchronous function in a synchronous mode, you can use the invokeInSyncMode function. The function call just needs to be wrapped around with the invokeInSyncMode function call. For instance, invokeInSyncMode (myAsyncFunc(arg1,arg2)). See InvokeCFClientFunction.

Asynchronous behavior

As a ColdFusion developer, you have always been using synchronous programming models. However, programming client applications using JavaScript needs to follow an asynchronous model. Fortunately, ColdFusion does most of the synchronous to asynchronous translation automatically.



For instance, see the following script:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cfscript>
  try
  {
    //Your code that throws an exception
  }
  catch (e)
  {
    // This statement will be reached unlike
    // in typical asynchronous model
  }
</cfscript>
```

The ability to use asynchronous functions in <cfclient> through the ‘known’ synchronous model provides a lot of convenience while building mobile applications using ColdFusion.

Since ColdFusion automatically translates synchronous code to asynchronous code, exception handling becomes easier in client code.

The behavior of certain tags has been modified to support the asynchronous behavior. In the process, functionalities of some tags may differ. For instance, <cfloop> does not support the following features when used along with <cfclient>:

- Looping over a COM collection
- Looping over a date or time range
- Looping over a directory

Support for ColdFusion Functions

You can start writing mobile applications in ColdFusion using existing data types and functions. The <cfclient> tag supports CFML data types and functions.

The following functions depict usage of data types and functions in your ColdFusion-based mobile projects.

Using CFML simple types

The following example shows the usage of simple data types:

```
<cfclient>
    <cfset myVar1 = 1>
    <cfset myVar2 = “hello”>
    <cfset myVar3 = true>
</cfclient>
```

Using CFML structures

© 2014 Adobe Systems Incorporated. All rights reserved.

The following example shows the usage of simple structures:

```
<cfclient>
    <cfset myStruct = structNew()>
    <cfset myStruct.key1= “hello”>
    <cfif structKeyExists(myStruct, “key1”)>
         ...
    </cfif>
</cfclient>
```

Using CFML arrays

The following example shows the usage of arrays:

```
<cfclient>
    <cfset myArray = arrayNew(1)>
    <cfset myArray[1] = “hello”>
</cfclient>
```

Using CFML functions

The following example shows the usage of functions:

```
<cfclient>
     <cfif arrayLen(myArray) gt 1 >
        ...
     </cfif>
     <!--- using the math function--->
     <cfset sum = arraySum(myArray) >


     <!--- using the date/time function--->
     <cfset currentDate = now() >


     <!--- using the locale function--->
     <cfset locale = getLocale() >
 </cfclient>
```

List of all supported functions

The following list shows all the **supported Array functions** in client-side CFML:

- arraySlice

© 2014 Adobe Systems Incorporated. All rights reserved.

- arrayAppend
- arrayIsDefined
- arrayAvg
- arrayIsEmpty
- arrayClear
- arrayLen
- arrayMax
- arrayNew
- arrayMin
- arraySort
- arrayDelete
- arrayToList
- arrayPrepend
- isArray
- arrayResize
- listToArray
- arraySet
- arrayFind
- arrayFindAll
- arraySum
- arraySwap
- arrayFindNoCase
- arrayFindAllNoCase

The following Array function is **NOT supported**:

- arrayFilter (closure function)

The following list shows all the **supported Structure functions** in client-side CFML:

- isStruct
- structDelete
- structAppend
- structInsert
- structClear
- structIsEmpty
- structCopy
- structKeyExists
- structCount
- structNew
- structFind
- structUpdate
- structFindKey
- structFindValue
- structGet
- structKeyArray
- structKeyList
- structSort

The following list shows all the **supported List functions** in client-side CFML:

- find
- findNoCase
- findOneOf
- formatBaseN

© 2014 Adobe Systems Incorporated. All rights reserved.

- lCase
- left
- len
- listAppend
- listChangeDelims
- listContains
- listContainsNoCase
- listDeleteAt
- listFind
- listFindNoCase
- listFirst
- listGetAt
- listInsertAt
- listLast
- listLen
- listPrepend
- listQualify
- listRest
- listSetAt
- listSort
- listToArray
- listValueCount
- listValueCountNoCase
- replaceList
- valueList

The following List function **is NOT supported**:

- getClientVariablesList

The following String functions **are NOT supported**:

- binaryEncode
- binaryDecode
- charsetEncode
- charsetDecode
- toBase64
- toBinary
- toString
- uRLDecode
- uRLEncodeFormat
- lSParseNumber
- lSParseCurrency
- lSParseEuroCurrency
- lSIsDate
- lSIsNumeric
- lSIsCurrency

The following **Regex functions are supported**:

- rEFind
- rEMatch
- rEFindNoCase
- rEMatchNoCase
- rEReplace

© 2014 Adobe Systems Incorporated. All rights reserved.

- rEReplaceNoCase

The following Math function **is NOT supported**:

- precisionEvaluate

The following Date functions **are NOT supported**:

- createODBCDate
- createODBCTime
- createODBCDateTime
- lSDateFormat
- lSIsDate
- lSParseDateTime
- lSTimeFormat

The following **utility functions are supported**:

- isBoolean
- isDefined
- decimalFormat
- isNumeric
- dollarFormat
- isNull
- htmlCodeFormat
- isSimpleValue
- htmlEditFormat
- isValid
- numberFormat
- createUUID

Support for custom tags

You have been using custom tags in ColdFusion for the past few releases of ColdFusion. Custom tags allowed you to extend CFML by adding your own tags to the ones shipped with ColdFusion. Custom tags can now be created in <cfclient> too. The following sections provide an overview of the supported features and restrictions while using custom tags for building mobile applications.

Note: Application and Server mappings are also supported in custom tags.

Paths for custom tags

Custom tags are detected when they are made available in the following locations:

- The custom tag available in the same directory as the calling page
- The custom tag available in the cfusion/CustomTags directory
- The custom tag available in sub-directories under the cfusion/CustomTags directory
- The custom tag available in server/application mapped folders

Invoking custom tags

The custom tags can be invoked in the following ways:

- Using the cf_<*tagname*>. For instance, by calling <cf_mytag>
- Using the <cfmodule> tag. For instance, <cfmodule template="../cf_mytag.cfm">
    - Also, <cfmodule name="tags.mytag">

© 2014 Adobe Systems Incorporated. All rights reserved.

- For the <cfimport> tag, we use the taglib to import the custom tags:
  - For instance, <cfimport prefix = “myTags” taglib = “/custom”>
  - The DOT(.) notation can be used to access custom tags available inside sub directories. For instance, use <cfmodule name = “tags.mytag”>

<cfimport> supports only path to custom tags and hence you cannot have JSP tag libraries.

Passing values

You can pass values to a custom tag using a name-value pair:

```
<cf_mytag myname=#myvalue#>
```

Also, multiple name-value pairs can be passed to a custom tag:

```
<cf_mytag myname1=#myvalue1# myname2=#myvalue2#>
```

To access the arguments passed to the custom tag, the custom tag CFM file can use the attributes scope as follows:

```
#attributes.myname1# and #attributes.myname2#
```

To send the data back to the calling page, the custom tag CFM file can use the Caller scope as follows:

```
<cfset caller.myname=#attributes.myname1# & " " & #attributes.myname2#>
```

You can also pass a struct to the custom tag:

```
<cfset args=structNew()>
<cfset args.x = "X">
<cfset args.y = "Y">
<cf_mytag arg1="value1" attributeCollection=#args# anotherarg="16">
```

Tag instance data

When a custom tag page executes, ColdFusion keeps data related to the tag instance in the **thisTag** structure. You can access the thisTag structure from within your custom tag to control processing of the tag.

To determine if an end tag is specified, use the hasEndTag as follows:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cfif thisTag.hasEndTag is 'false'>
  <! Abort the tag>
  <cfabort />
</cfif>
```

To determine the tag execution mode, use the executionMode attribute. Three modes are supported:

- Start mode – For processing the start tag
- End mode – For processing the end tag
- Inactive mode – For processing custom tags using nested tags

```
<cfif thisTag.executionMode is 'start'>
  <! Process start tag >
  <cfelseif thisTag.executionMode is 'end'>
  <! Process end tag >
</cfif>
```

You can access the body text within the custom tag using the thisTag.generatedContent variable. You can modify this text during processing of the tag. The contents of the thisTag.generatedContent variables are returned to the browser as part of the tag's output. The content includes all text and HTML code in the body, the results of evaluating ColdFusion variables, expressions, and functions, and the results generated by descendant tags.

See the following example:

```
<cfif thisTag.executionMode is 'end'>
  <cfset thisTag.generatedContent ='<!#thisTag.generatedContent#>'>
</cfif>
```

The nested sub tag can pass its attributes to the parent tag. A sub tag can use cfassociate to communicate its attributes to the base/ancestor tag.

```
<cfassociate baseTag="tagName" dataCollection="collectionName">
```

The following code shows how you can access the subtag attributes in the base tag:

```
<cfparam Name='thisTag.assocAttribs' default=#arrayNew(1)#>
```

You can also access the ancestral data in the sub tag using the getBaseTagList() helper method as follows:

```
<cfset ancestorlist = getBaseTagList()>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

The getBaseTagList() method returns a comma-delimited list of uppercase ancestor tag names, as a string. You can also use the getBaseTagData() method to return an object that contains all the variables of the *n*th ancestor.

Aborting custom tag processing

The <cfexit>/<cfabort> tag exits the page execution.

Deviation list for custom tags

The following list contains some known issues and deviations in behavior of the custom tags:

- In <cfclient>, variables scope is supported. But you have to explicitly scope it.

Server-side CFML	Client-side CFML

caller.cfm
`<cf_customtag value1="old_value">`
`<cfoutput>`
`#variables.old_value#`
`<cfoutput>`
`<cfoutput>`
`#old_value#`
`<cfoutput>`
 	
customtag.cfm
`<cfset caller[attributes.value1]="new_value"/>`
	caller.cfm
	`<cfclient>`
	`<cf_customtag value1="old_value" >`
	`<cfoutput>#variables.old_value#<cfoutput>`
	`</cfclient>`
	
	customtag.cfm
	`<cfset caller[attributes.value1]="new_value"/>`

- If you use the "#attributes.attributename#" syntax in the custom tag after an asynchronous call, you will see an unexpected behavior.

Server-side CFML	Client-side CFML

© 2014 Adobe Systems Incorporated. All rights reserved.

`<cfset divid = "#attributes.div_id#">`	`<cfset divid = "#attributes.div_id#">`
	
	The above code will not work.
	Use:
	
	`<cfset divid = #attributes.div_id#>`

- Numeric values passed to the attributes in caller are passed as a string to the custom tags:

```
<cf_custom attr1="1">
```

In the above example, attr1 is converted to a number, if you are accessing the attribute in a numeric operation. However, it does not work in this manner for client-side custom tags. You need to use:

```
<cf_custom attr1=1>
```

Or:

```
<cfset x = 1>
<cf_custom attr1=#x#>
```

Type conversion is not handled in client custom tags.

- Function declared in the caller CFM is accessible in the custom tag (CFM) using the caller.functionname() on the server-side. However, this is not the behavior on the client side.

Server-side CFML	Client-side CFML

`<cffunction name="func1">`
`<cfretrurn "Hello">`
`</cffunction>`
`<cf_custom>`
`<cfset caller.func1()>`
	The functions defined in the caller CFM are not available in the custom tags.

© 2014 Adobe Systems Incorporated. All rights reserved.

- Using variables to pass the path of the included file does not work inside <cfclient>.

Server-side CFML	Client-side CFML
`<cfset path="someCFM.cfm">` `<cfinclude template=#path#>`	This is not supported.

- Passing the template/name (with <cfmodule>) attribute as a variable does not work with <cfclient>.

Server-side CFML	Client-side CFML
`<cfset path="someCFM.cfm">` `<cfmodule template=#path#>`	This is not supported as we need to do the translation during the compile time itself.

- Exception thrown in the custom tag template will not be handled by the exception handler defined in the caller CFM. If the custom tag name is wrong or if the included CFM name is wrong, in client-side CFML, you will get an exception during the compilation time itself.

Note: Based on the location of the JavaScript file (specified in the <cfinclude> tag or using the <script> tag), the order of execution of statements differ.

Non-<cfclient> custom tags cannot be called from caller CFMs of <cfclient>. Also, a client-side custom tag cannot have server-side ColdFusion tags outside the <cfclient> tag. This is true for client-side included-CFMs too. For better debugging, do not add script blocks/link tags/style tags in the client-side custom tags. Always create a separate JavaScript file or a CSS file and add them using the <cfinclude> tag.

```
<cfinclude template="utils.js">
or
<cfinclude template="new.css">
```

This is applicable for client-side included CFMs too.

Support for CFC (Client-side and Server-side)

A client-side CFC can be written using the client=true attribute for a cfclient component. For instance, a client-side CFC can identify itself by having client=true along with other component attributes. See the following example:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
component client=true
{
    public function foo()
    {
        //some code here
    }
}
```

<cfclient> communicates with the ColdFusion Server quite seamlessly so much so that you can create objects of a server component and call functions on that object just like how you do ColdFusion Server programming. The ColdFusion Server facilitates all the various interactions required between a mobile client and the server without any restrictions on the language.

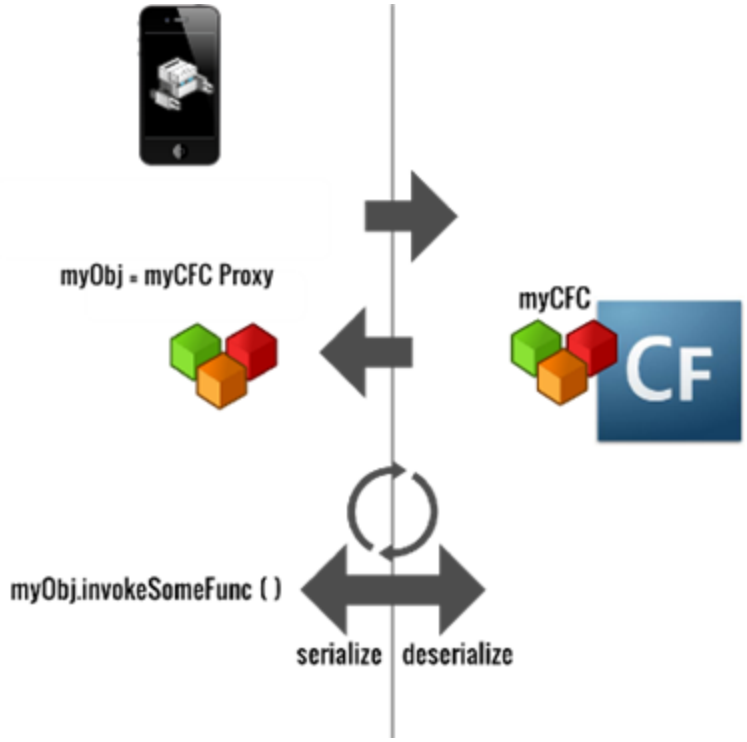
See the following example:

```
<cfclient>
   <!-- Create a JS proxy of server side component myCFC.cfc>
   <cfset proxy = new app1.controls.myCFC(id)>


   <!-- Update some data >
   <cfset proxy.setVar("myVar1")>
   <cfset proxy.setProperties(someStructVar)>
</cfclient>
```

In the above example, you are calling a function on a remote CFC within the <cfclient> tag. The new operator creates a proxy to the server CFC. You can use the instantiated CFC object to invoke remote functions on that CFC.

Note that a server CFC is similar to any other CFC without the client attribute being set to true.

© 2014 Adobe Systems Incorporated. All rights reserved.



The <cfclient> tag allows the usage of CFCs just like any other CFML constructs. There are multiple ways of using CFCs in the <cfclient> block.

The following example shows a very simple usage:

```
<cfclient>
  <cfset obj = new mycfc()>
  <cfset obj1 = createObject("component","mycfc")>
</cfclient>
```

In the above example, mycfc.cfc can be a client-side CFC or a server-side CFC. As you can infer, CFCs can be created using createObject, new keyword, and cfinvoke techniques.

You can also use a CFC that extends functionalities from another CFC:

```
<cfclient>
   <cfset obj = new mycfc()>
    <!--- mycfc extends mycfc1.cfc present in the same directory --->
   <cfset result = obj.getResult() >
    <!--- getResult() function present in mycfc1.cfc and can be
    accessed from the extending classes --->
</cfclient>
```

Note: Ensure that if mycfc.cfc is a client-side CFC, then mycfc1.CFC should also be a client-side CFC. This is applicable even for the server-side CFC if mycfc.cfc is a server-side CFC.

© 2014 Adobe Systems Incorporated. All rights reserved.

You can also use <cfimport> for importing mapped paths:

```
<cfimport path="com.*" />
<cfset obj = new com.mycfc() />
<!--- mycfc present in directory mapped to com --->


You can also use functions within a CFC:
<cfclient>
  <cfset obj = new mycfc() >
  <cfset obj.foo() >
  <!--- invoke function on cfc instance --->
</cfclient>
```

Note: Ensure that the function foo() is a remote function, if mycfc.cfc is a server-side CFC.

Support for database queries

You can start using the <cfquery> tag in client-side CFML just like how you are currently using it in server-side CFML code. Note that not all of the <cfquery> features are supported in this release. The support for database queries in client-side CFML is based on Web Database (Web SQL). So, this feature may not work on certain browsers. To check if your browser supports Web SQL, see this web page.

What is supported

The following list shows the extent of <cfquery> support available in client-side CFML:

- The <cfquery> tag supports ONLY the following attributes:
  - name = "query name"
  - dataSource = "data source name"
  - result = "resultVar"

The result variable will contain sql, recordCount, columnList, and sqlparameters.

- You can use the <cfloop> tag to iterate over the query. The <cfloop> tag will support query, startrow, and endrow attributes.
- You can use the <cfqueryparams> tag for parameterized query.
  - In the <cfqueryparam> tag, only the value attribute is supported.
  - The queryparam attribute value can be provided through position parameters in an array using the queryExecute function:

queryexecute("sql", queryparams, queryoptionsmap)

- Note that serialization and deserialization of queries from client to server and server to client will be seamless.
- You can use the isQuery function in client-side CFML to check if a variable is of query type.

© 2014 Adobe Systems Incorporated. All rights reserved.

What is NOT supported

The following features are not supported:

- Performing query of queries.
- In-memory query creation functions like QueryNew and QueryAddRow.

Usage example

The following example shows the basic usage of the <cfquery> tag in client-side CFML:

```
<div id="actual_1" class="async_actual">

    <cfclient>

        <cfquery datasource="cfds" >drop table if exists birthDates</cfquery>

        <cfquery datasource="cfds" >
   CREATE TABLE if not exists birthDates(
    serialNo INTEGER PRIMARY KEY AUTOINCREMENT,
    firstname  VARCHAR(20), lastname VARCHAR(20), dob TEXT)
  </cfquery>
         <!---Insert string. --->
        <cfquery datasource="cfds" name="q1">
        INSERT INTO birthDates(firstName, lastname,dob) VALUES('Jon', 'Doe', 'Mark')
        </cfquery>

  <cfset d1=createDate(1975, 12, 25)>

  <cfset a=dateFormat(d1,"yyyy-mm-dd")>

        <cfquery datasource="cfds" name="q2">
          INSERT INTO birthDates(firstName, lastname, dob) VALUES('Jon', 'Doe',
'#a#')
        </cfquery>

  <cfset d2=createDate(1980, 01, 01)>

  <cfset b=dateFormat(d2, "yyyy-mm-dd")>

        <cfquery datasource="cfds" name="q2">
   INSERT INTO birthDates(firstName, lastname, dob) VALUES('Jon', 'Doe','#b#')
        </cfquery>

  <cfset d3=createDate(1985, 12, 27)>

  <cfset c=dateFormat(d3, "yyyy-mm-dd")>

        <cfquery datasource="cfds" name="q3">
   INSERT INTO birthDates(firstName, lastname, dob) VALUES('Jon', 'Doe','#c#')
  </cfquery>


        <cfset startRow="2">
        <cfset endRow="4">

        <cfquery datasource="cfds" name="q4" result="test">
```

© 2014 Adobe Systems Incorporated. All rights reserved.

```
   SELECT * FROM birthDates where serialNo between <cfqueryparam value="#startRow#">
and <cfqueryparam value="#endRow#">
        </cfquery>

        <cfset write_to_div("actual_1", test.sql & "<br>" & test.recordCount  &
"<br>" & test.columnList)>

        <cfloop query="q4">
   <cfset write_to_div("actual_1", firstname & " " & lastname & ":" & dob &"<br>")>

  </cfloop>

    </cfclient>

    <script type="text/javascript">

        function write_to_div(div_id,data_to_write)
        {
   document.getElementById(div_id).innerHTML+=data_to_write;
        }
```

© 2014 Adobe Systems Incorporated. All rights reserved.

```
    </script>

</div>
```

See the server-side <cfquery> support.

© 2014 Adobe Systems Incorporated. All rights reserved.

Social Enhancements

ColdFusion has introduced the support for dynamically generating Like button, Tweet button, and Comment box for social media sites. The supported widgets are:

- Like button
- Tweet button
- Facebook comment box
- Google Plus button
- Facebook subscribe button
- Like box
- Activity feed
- Follow

Examples

1. Syntax for Facebook Like button:

```
<cf_socialplugin type = "like"
   url = ""
   layout = "standard|box_count|button_count"
   showfaces = "true|false"
   verb = "like|recommend"
   colorscheme = "light|dark"
   style = ""
   width = ""
   extraoptions = ""
>
```

2. Syntax for Facebook Likebox button:

```
<cf_socialplugin type = "likebox"
   url = "This refers to a Facebook Page."
   showfaces = "true|false"
   showstream = "true|false"
   showheader = "true|false"
   colorscheme = "light|dark"
   style = ""
   height = ""
   width = ""
   extraoptions = ""
>
```

3. Syntax for Activity feed for a Facebook application:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cf_socialplugin type = "activityfeed"
        appid= "facebook_app_id"
        width = ""
        height = ""
        colorscheme = "light|dark"
        showheader = "true|false"
        action = ""
        linktarget = "_blank|_top|_parent"
        recommendations = "true|false"
        style = ""
        extraoptions = ""
>
```

4. Syntax for a Twitter Follow button:

```
<cf_socialplugin type = "follow"
                showcount= "true|false"
                buttonsize= "medium|large"
                language = "en|fr|..........."
                showusername = "true|false"
                username = ""
                style = ""
                extraoptions = ""
>
```

5. Syntax for a Google Plus button:

```
<cf_socialplugin type = "plusone"
  url = "url to plus one"
  buttonsize= "small|medium|large|tall"
  language = "en|fr|..........."
  width = ""
  annotation = "none|inline|bubble"
  style = ""
  extraoptions = ""
>
```

6. Syntax for a Facebook Comment box:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cf_socialplugin type = "commentbox"
                 url = "url"
                 width= ""
                 colorscheme = "dark|light"
                 numberofposts = ""
                 style = ""
                 extraoptions = ""
>
```

7. Syntax for a Facebook Subscribe button:

```
<cf_socialplugin type = "subscribe"
                 url = "profile to subscribe"
                 width = ""
                 colorscheme = "dark|light"
                 showfaces = "true|false"
                 layout = "standard|button_count|box_count"
                 style = ""
                 extraoptions = ""
>
```

8. Syntax for Tweet button:

```
<cf_socialplugin type = "tweet"
                 url = "url to share"
                 tweettext = "default tweet text"
                 language = "en|fr|..........."
                 count = "none|horizontal"
                 hashtag = "Comma separated hash tags appended to
                            tweet text. Do not include the #. It is preprended to
each list item automatically."
                 buttonsize = "small|large"
                 via = ""
                 recommend = ""
                 style = ""
                 extraoptions = ""
>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

REST Enhancements in ColdFusion 11

- Site-level REST application support
- Support for pluggable serializer and deserializer

ColdFusion 11 now supports site-level REST applications and enables pluggable serializer and deserializer.

Site-level REST application support

See Site-level REST application support

Support for pluggable serializer and deserializer

See Support for pluggable serializer and deserializer

© 2014 Adobe Systems Incorporated. All rights reserved.

Authentication through OAuth

The <oauth> tag allows you to easily integrate third-party Oauth 2 authentication provider in your application. This tag currently supports Facebook and Google authentication. Also, this tag supports Oauth providers that support the Oauth 2 protocols. For instance, Microsoft and Github.

See cfoauth

© 2014 Adobe Systems Incorporated. All rights reserved.

Accessing and Using Data

Effective Database management and using the querying facility to search for data is explained in this section.

- Introduction to Databases and SQL
- Accessing and Retrieving Data
- Updating Your Database
- Using Query of Queries
- Managing LDAP Directories
- Solr search support

#back to top

© 2014 Adobe Systems Incorporated. All rights reserved.

Introduction to Databases and SQL

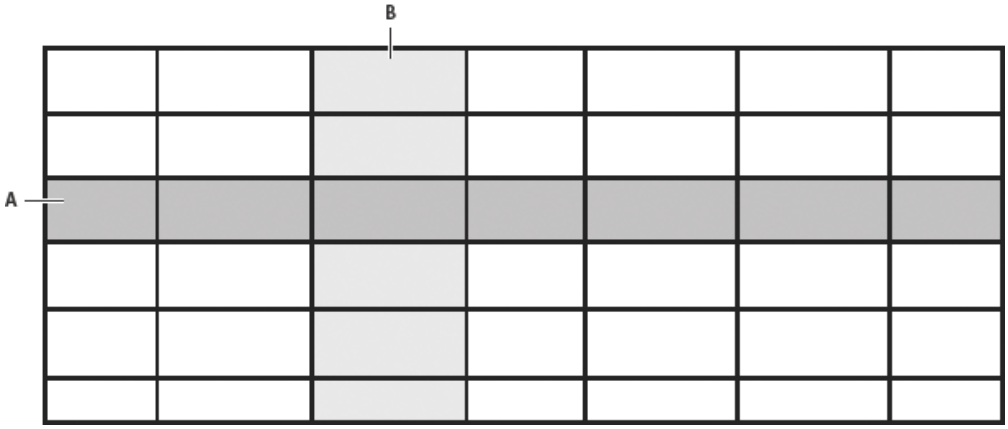
Adobe ColdFusion lets you create dynamic applications to access and modify data stored in a database. You do not require a thorough knowledge of databases to develop ColdFusion applications, but you need to know some basic database and SQL concepts and techniques.

Each database server (such as SQL Server, Oracle, or DB2) has unique capabilities and properties. For more information, see the documentation that ships with your database server.

© 2014 Adobe Systems Incorporated. All rights reserved.

What is a database?

A *database* defines a structure for storing information. Databases are typically organized into *tables*, which are collections of related items. You can think of a table as a grid of columns and rows. ColdFusion works primarily with relational databases, such as Oracle, DB2, and SQL Server.
The following image shows the basic layout of a database table:



A *column* defines one piece of data stored in all rows of the table. A *row* contains one item from each column in the table.
For example, a table contains the ID, name, title, and other information for individuals employed by a company. Each row, called a data *record*, corresponds to one employee. The value of a column within a record is referred to as a record *field*.
The following image shows an example table, named employees, containing information about company employees:

EmpID	LastName	FirstName	Title	DeptID	Email	Phone
4	Smith	John	Engineer	3	jsmith	x5833

The record for employee 4 contains the following field values:

- LastName field is "Smith"
- FirstName field is "John"
- Title field is "Engineer"
  This example uses the EmpID field as the table's *primary key* field. The primary key contains a unique identifier to maintain each record's unique identity. Primary keys field can include an employee ID, part number, or customer number. Typically, you specify which column contains the primary key when you create a database table.
  To access the table to read or modify table data, you use the SQL programming language. For example, the following SQL statement returns all rows from the table where the department ID is 3:

```
SELECT * FROM employees WHERE DEPTID=3
```

© 2014 Adobe Systems Incorporated. All rights reserved.

Note

The SQL keywords and syntax are represented here as uppercase letters. Table and column names use mixed uppercase and lowercase letters.

Using multiple database tables

In many database designs, information is distributed to multiple tables. The following image shows two tables, one for employee information and one for employee addresses:

EmpID	LastName	FirstName	Title	DeptID	Email	Phone
1	Jones	Joe	Engineer	3	jjones	x5844
2	Davis	Ken	Manager	4	kdavis	x5854
3	Baker	Mary	Engineer	3	mbaker	x5876
4	Smith	John	Engineer	3	jsmith	x5833
5	Morris	Jane	Manager	3	jmorris	x5833

EmpID	Street	City	State	Zip
1	4 Main St.	Newton	MA	02158
2	10 Oak Dr.	Newton	MA	02161
3	15 Main St.	Newton	MA	02158
4	56 Maple Ln.	Newton	MA	02160
5	25 Elm St.	Newton	MA	02160

In this example, each table contains a column named EmpID. This column associates a row of the employees table with a row in the addresses table.
For example, to obtain all information about an employee, you request a row from the employees table and the row from the addresses table with the same value for EmpID.
One advantage of using multiple tables is that you can add tables containing new information without modifying the structure of your existing tables. For example, to add payroll information, you add a new table to the database where the first column contains the employee's ID and the columns contain current salary, previous salary, bonus payment, and 401(k) percent.
Also, an access to a small table is more efficient than an access to a large table. Therefore, if you update the street address of an employee, you update only the addresses table, without having to access any other table in the database.

Database permissions

© 2014 Adobe Systems Incorporated. All rights reserved.

In many database environments, a database administrator defines the access privileges for users accessing the database, usually through user name and password. When a person attempts to connect to a database, the database ensures that the user name and password are valid and then imposes access requirements on the user. Privileges can restrict user access so that a user can do the following:

- Read data.
- Read data and add rows.
- Read data, add rows, modify existing tables.
  In ColdFusion, you use the ColdFusion Administrator to define database connections, called *data sources*. As part of defining these connections, you specify the user name and password used by ColdFusion to connect to the database. The database can then control access based on this user name and password.
  For more information on creating a data source, see *Configuring and Administering ColdFusion*.

Commits, rollbacks, and transactions

Before you access data stored in a database, it is important to understand several database concepts, including:

- Commit
- Rollback
- Transactions
  A database *commit* occurs when you make a permanent change to a database. For example, when you write a new row to a database, the write does not occur until the database commits the change.
  Rollback is the process of undoing a change to a database. For example, if you write a new row to a table, you can rollback the write up to the point where you commit the write. After the commit, you can no longer rollback the write.
  Most databases support transactions where a transaction consists of one or more SQL statements. Within a transaction, your SQL statements can read, modify, and write a database. You end a transaction by either committing all your changes within the transaction or rolling back all of them.
  Transactions can be useful when you have multiple writes to a database and want to make sure all writes occurred without error before committing them. In this case, you wrap all writes within a single transaction and check for errors after each write. If any write causes an error, rollback all of them. If all writes occur successfully, you commit the transaction.
  A bank might use a transaction to encapsulate a transfer from one account to another. For example, if you transfer money from your savings account to your checking account, you do not want the bank to debit the balance of your savings account unless it also credits your checking account. If the update to the checking account fails, the bank can rollback the debit of the savings account as part of the transaction.
  ColdFusion includes the `cftransaction` tag that lets you implement database transactions for controlling rollback and commit. For more information, see the *CFML Reference*.

Database design guidelines

From this basic description, the following database design rules emerge:

- Each record should contain a unique identifier as the primary key such as an employee ID, a part number, or a customer number. The primary key is typically the column used to maintain each record's unique identity among the tables in a relational database. Databases allow you to use multiple columns for the primary key.
- When you define a column, you define a SQL data type for the column, such as allowing only numeric values to be entered in the salary column.
- Assessing user needs and incorporating those needs in the database design is essential to a successful implementation. A well-designed database accommodates the changing data needs within an organization. The best way to familiarize yourself with the capabilities of your database product or database management system (DBMS) is to review the product documentation.

© 2014 Adobe Systems Incorporated. All rights reserved.

Using SQL

The following information introduces SQL, describes basic SQL syntax, and contains examples of SQL statements. so that you can begin to use ColdFusion. For complete SQL information, see the SQL reference that ships with your database.

A *query* is a request to a database. The query can ask for information from the database, write new data to the database, update existing information in the database, or delete records from the database.

Structured Query Language (SQL) is an ANSI/ISO standard programming language for writing database queries. All databases supported by ColdFusion support SQL, and all ColdFusion tags that access a database let you pass SQL statements to the tag.

SQL example

The most commonly used SQL statement in ColdFusion is the SELECT statement. The SELECT statement reads data from a database and returns it to ColdFusion. For example, the following SQL statement reads all the records from the employees table:

```
SELECT * FROM employees
```

You interpret this statement as "Select all rows from the table employees" where the wildcard symbol ⭐ correspond s to all columns.

> ✅ If you are using Dreamweaver MX 2004, Adobe Dreamweaver CS3, or HomeSite+, you can use the built-in query builder to build SQL statements graphically by selecting the tables and records to retrieve.

In many cases, you do not want all rows from a table, but only a subset of rows. The next example returns all rows from the employees table, where the value of the DeptID column for the row is 3:

```
SELECT * FROM employees WHERE DeptID=3
```

You interpret this statement as "Select all rows from the table employees where the DeptID is 3".

SQL also lets you specify the table columns to return. For example, instead of returning all columns in the table, you can return a subset of columns:

```
SELECT LastName, FirstName FROM employees WHERE DeptID=3
```

You interpret this statement as "Select the columns FirstName and LastName from the table employees where the DeptID is 3".

In addition to with reading data from a table, you can write data to a table using the SQL INSERT statement. The following statement adds a new row to the employees table:

```
INSERT INTO employees(EmpID, LastName, Firstname) VALUES(51, 'Doe', 'John')
```

© 2014 Adobe Systems Incorporated. All rights reserved.

Basic SQL syntax elements

The following tables briefly describe the main SQL command elements.

Statements

A SQL statement always begins with a SQL verb. The following keywords identify commonly used SQL verbs:

Keyword	Description
SELECT	Retrieves the specified records.
INSERT	Adds a new row.
UPDATE	Changes values in the specified rows.
DELETE	Removes the specified rows.

Statement clauses

Use the following keywords to refine SQL statements:

Keyword	Description
FROM	Names the data tables for the operation.
WHERE	Sets one or more conditions for the operation.
ORDER BY	Sorts the result set in the specified order.
GROUP BY	Groups the result set by the specified select list items.

Operators

The following basic operators specify conditions and perform logical and numeric functions:

Operator	Description
AND	Both conditions must be met
OR	At least one condition must be met
NOT	Exclude the condition following
LIKE	Matches with a pattern
IN	Matches with a list of values
BETWEEN	Matches with a range of values

© 2014 Adobe Systems Incorporated. All rights reserved.

=	Equal to
<>	Not equal to
<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to
+	Addition
-	Subtraction
/	Division
*	Multiplication

Case sensitivity with databases

ColdFusion is a case-insensitive programming environment. *Case insensitivity* means the following statements are equivalent:

```
<cfset foo="bar">
<CFSET FOO="BAR">
<CfSet FOO="bar">
```

However, many databases, especially UNIX databases, are case sensitive. *Case sensitivity* means that you must match exactly the case of all column and table names in SQL queries.
For example, the following queries are not equivalent in a case-sensitive database:

```
SELECT LastName FROM EMPLOYEES
SELECT LASTNAME FROM employees
```

In a case-sensitive database, employees and EMPLOYEES are two different tables.
For information on how your database handles case, see the product documentation.

SQL notes and considerations

When writing SQL in ColdFusion, keep in mind the following guidelines:

- If you use a ColdFusion variable in your SQL expression, and the variable value is a string that contains single quotes, place the variable in a PreserveSingleQuotes function to prevent ColdFusion from interpreting the quotation marks. The following example shows this use:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cfset List = "'Suisun', 'San Francisco', 'San Diego'">
<cfquery name = "GetCenters" datasource = "cfdocexamples">
 SELECT Name, Address1, Address2, City, Phone
 FROM Centers
 WHERE City IN (#PreserveSingleQuotes(List)#)
</cfquery>
```

- There is a lot more to SQL than what is covered here. It is a good idea to purchase one or several SQL guides for reference.
- To perform a successful query, the data source, columns, and tables that you reference must exist.
- Some DBMS vendors use nonstandard SQL syntax (known as a dialect) in their products. ColdFusion does not validate the SQL; it is passed on to the database for validation, so you are free to use any syntax that your database supports. Check your DBMS documentation for nonstandard SQL usage.

Reading data from a database

You use the SQL SELECT statement to read data from a database. The SQL statement has the following general syntax:

```
SELECT column_names
 FROM table_names
 [ WHERE search_condition ]
 [ GROUP BY group_expression ] [HAVING condition]
 [ ORDER BY order_condition [ ASC | DESC ] ]
```

The statements in brackets [] are optional.

> **Note**
>
> There are additional options to SELECT depending on your database. For a complete syntax description for SELECT, see the product documentation.

Results of a SELECT statement

When the database processes a SELECT statement, it returns a *record set* containing the requested data. The format of a record set is a table with rows and columns. For example, if you write the following query:

```
SELECT * FROM employees WHERE DeptID=3
```

The query returns a database table. Because the data returned to ColdFusion by a SELECT statement is in the form of a database table, ColdFusion lets you write a SQL query on the returned results. This functionality is called *query of queries*. For more information on query of queries, see Accessing and Retrieving Data.
The next example uses a SELECT statement to return only a specific set of columns from a table:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
SELECT LastName, FirstName FROM employees WHERE DeptID=3
```

Filtering results

The SELECT statement lets you filter the results of a query to return only those records that meet specific criteria. For example, if you want to access all database records for employees in department 3, you use the following query:

```
SELECT * FROM employees WHERE DeptID=3
```

You can combine multiple conditions using the WHERE clause. For example, the following example uses two conditions:

```
SELECT * FROM employees WHERE DeptID=3 AND Title='Engineer'
```

Sorting results

By default, a database does not sort the records returned from a SQL query. In fact, you cannot guarantee that the records returned from the same query are returned in the same order each time you run the query.
However, if you require records in a specific order, you can write your SQL statement to sort the records returned from the database. To do so, you include an ORDER BY clause in the SQL statement.
For example, the following SQL statement returns the records of the table ordered by the LastName column:

```
SELECT * FROM employees ORDER BY LastName
```

You can combine multiple fields in the ORDER BY clause to perform additional sorting:

```
SELECT * FROM employees ORDER BY DepartmentID, LastName
```

This statement returns row ordered by department, then by last name within the department.

Returning a subset of columns

You want only a subset of columns returned from a database table, as in the following example, which returns only the FirstName, LastName, and Phone columns. This example is useful if you are building a web page that shows the phone numbers for all employees.

```
SELECT FirstName, LastName, Phone FROM employees
```

However, this query does not to return the table rows in alphabetical order. You can include an ORDER clause in the SQL, as follows:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
SELECT the FirstName, LastName, Phone
 FROM employees
 ORDER BY LastName, FirstName
```

Using column aliases

You have column names that you do not want to retain in the results of your SQL statement. For example, your database is set up with a column that uses a reserved word in ColdFusion, such as EQ. In this case, you can rename the column as part of the query, as follows:

```
SELECT EmpID, LastName, EQ as MyEQ FROM employees
```

The results returned by this query contains columns named EmpID, LastName, and MyEQ.

Accessing multiple tables

In a database, you can have multiple tables containing related information. You can extract information from multiple tables as part of a query. In this case, you specify multiple table names in the SELECT statement, as follows:

```
SELECT LastName, FirstName, Street, City, State, Zip
 FROM employees, addresses
 WHERE employees.EmpID = addresses.EmpID
 ORDER BY LastName, FirstName
```

This SELECT statement uses the EmpID field to connect the two tables. This query prefixes the EmpID column with the table name. This is necessary because each table has a column named EmpID. Prefix a column name with its table name if the column name appears in multiple tables.
In this case, you extract LastName and FirstName information from the employees table and Street, City, State, and ZIP information from the addresses table. You can use output such as this is to generate mailing addresses for an employee newsletter.
The results of a SELECT statement that references multiple tables is a single result table containing a join of the information from corresponding rows. A *join* means information from two or more rows is combined to form a single row of the result. In this case, the resultant recordset has the following structure:

© 2014 Adobe Systems Incorporated. All rights reserved.

LastName	FirstName	Street	City	State	Zip

What is interesting in this result is that even though you used the EmpID field to combine information from the two tables, you did not include that field in the output.

Modifying a database

You can use SQL to modify a database in the following ways:

Inserting data into a database

You use SQL INSERT statement to write information to a database. A write adds a new row to a database table. The basic syntax of an INSERT statement is as follows:

```
INSERT INTO table_name(column_names) VALUES(value_list)
```

where:

- *column_names* specifies a comma-separated list of columns.
- *value_list* specifies a comma-separated list of values. The order of values has to correspond to the order that you specified column names.

> **Note**
>
> There are additional options to INSERT depending on your database. For a complete syntax description for INSERT, see the product documentation.

For example, the following SQL statement adds a new row to the employees table:

```
INSERT INTO employees(EmpID, LastName, Firstname) VALUES(51, 'Smith', 'John')
```

This statement creates a row in the employees table and sets the values of the EmpID, LastName, and FirstName fields of the row. The remaining fields in the row are set to Null. _Null_means that the field does not contain a value. When you, or your database administrator, creates a table, you can set properties on the table and the columns of the table. One of the properties you can set for a column is whether the field supports Null values. If a field supports Nulls, you can omit the field from the INSERT statement. The database automatically sets the field to Null when you insert a new row.
However, if the field does not support Nulls, specify a value for the field as part of the INSERT statement; otherwise,

© 2014 Adobe Systems Incorporated. All rights reserved.

the database issues an error.
The LastName and FirstName values in the query are contained within single-quotation marks. This is necessary because the table columns are defined to contain character strings. Numeric data does not require the quotation marks.

Updating data in a database

Use the UPDATE statement in SQL to update the values of a table row. Update lets you update the fields of a specific row or all rows in the table. The UPDATE statement has the following syntax:

```
UPDATE table_name
 SET column_name1=value1, ... , column_nameN=valueN
 [ WHERE search_condition ]
```

> **Note**
>
> There are additional options to UPDATE depending on your database. For a complete syntax description for UPDATE, see the product documentation.

Do not attempt to update a record's primary key field. Your database typically enforces this restriction.
The UPDATE statement uses the optional WHERE clause, much like the SELECT statement, to determine which table rows to modify. The following UPDATE statement updates the e-mail address of John Smith:

```
UPDATE employees SET Email='jsmith@mycompany.com' WHERE EmpID = 51
```

Be careful using UPDATE. If you omit the WHERE clause to execute the following statement:

```
UPDATE employees SET Email = 'jsmith@mycompany.com'
```

you update the Email field for all rows in the table.

Deleting data from a database

The DELETE statement removes rows from a table. The DELETE statement has the following syntax:

```
DELETE FROM table_name
 [ WHERE search_condition ]
```

> **Note**
>
> There are additional options to DELETE depending on your database. For a complete syntax description for DELETE, see the product documentation.

You can remove all rows from a table using a statement in the form:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
DELETE FROM employees
```

Typically, you specify a WHERE clause to the DELETE statement to delete specific rows of the table. For example, the following statement deletes John Smith from the table:

```
DELETE FROM employees WHERE EmpID=51
```

Updating multiple tables

The preceding examples describe how to modify a single database table. However, you might have a database that uses multiple tables to represent information.
One way to update multiple tables is to use one INSERT statement per table and to wrap all INSERT statements within a database transaction. A transaction contains one or more SQL statements that can be rolled back or committed as a unit. If any single statement in the transaction fails, you can roll back the entire transaction, canceling any previous writes that occurred within the transaction. You can use the same technique for selects, updates, and deletes. The following example uses the `cftransaction` tag to wrap multiple SQL statements:

```
<cftransaction>

<cfquery name="qInsEmp" datasource="cfdocexamples">
 INSERT INTO Employees (FirstName,LastName,EMail,Phone,Department)
 VALUES ('Simon', 'Horwith', 'SHORWITH','(202)-797-6570','Research and Development')

</cfquery>

<cfquery name="qGetID" datasource="cfdocexamples">
 SELECT MAX(Emp_ID) AS New_Employee
 FROM Employees
</cfquery>

</cftransaction>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

Accessing and Retrieving Data

Several ColdFusion tags provide a way to retrieve data from a database and work with query data. Use the `cfquery` tag to query a data source, the `cfoutput` tag to output the query results to a web page, and the `cfqueryparam` tag to help reduce security risks in your applications.

© 2014 Adobe Systems Incorporated. All rights reserved.

Working with dynamic data

A web application page is different from a static web page because it can publish data dynamically. This can involve querying databases, connecting to LDAP or mail servers, and leveraging COM, DCOM, CORBA, or Java objects to retrieve, update, insert, and delete data at run time as your users interact with pages in their browsers.
For ColdFusion developers, the term data source can refer to different types of structured content accessible locally or across a network. You can query websites, LDAP servers, POP mail servers, and documents in a variety of formats. Most commonly though, a database drives your applications, and for this discussion a *data source* means the entry point from ColdFusion to a database.
Here, you build a query to retrieve data from the cfdocexamples data source.
To query a database, use:

- ColdFusion data sources
- The `cfquery` tag
- SQL commands

© 2014 Adobe Systems Incorporated. All rights reserved.

Retrieving data

You can query databases to retrieve data at run time. The retrieved data, called the *recordset*, is stored on that page as a query object. A_ query object_ is a special entity that contains the recordset values, plus RecordCount, CurrentRow, ColumnList, SQL, Cached, and SQLParameter query variables. You specify the name of the query object in the `name` attribute of the `cfquery` tag. The query object is often called simply *the query*.
The following is a simple `cfquery` tag:

```
<cfquery name = "GetSals" datasource = "cfdocexamples">
 SELECT * FROM Employee
 ORDER BY LastName
</cfquery>
```

Note

The terms "recordset" and "query object" are often used synonymously when discussing recordsets for queries. For more information, see Using Query of Queries.

When retrieving data from a database, perform the following tasks:

- To tell ColdFusion how to connect to a database, use the `cfquery` tag on a page.
- To specify the data that you want to retrieve from the database, write SQL commands inside the `cfquery` bl ock.
- Reference the query object and use its data values in any tag that presents data, such as `cfoutput`, `cfgri d`, `cftable`, `cfgraph`, or `cftree`.

The cfquery tag

The `cfquery` tag is one of the most frequently used CFML tags. You use it to retrieve and reference the data returned from a query. When ColdFusion encounters a `cfquery` tag on a page, it does the following:

- Connects to the specified data source.
- Performs SQL commands that are enclosed within the block.
- Returns result set values to the page in a query object.

The cfquery tag syntax

The following code shows the syntax for the `cfquery` tag:

```
<cfquery name="EmpList" datasource="cfdocexamples">
 SQL code...
</cfquery>
```

In this example, the query code tells ColdFusion to do the following:

- Connect to the cfdocexamples data source (the cfdocexamples.mdb database).
- Execute SQL code that you specify.
- Store the retrieved data in the query object EmpList.

When creating queries to retrieve data, keep in mind the following guidelines:

© 2014 Adobe Systems Incorporated. All rights reserved.

- Use opening `<cfquery>` and ending `</cfquery>` tags, because the `cfquery` tag is a block tag.
- Enter the query `name` and `datasource` attributes within the opening `cfquery` tag.
- To tell the database what to process during the query, place SQL statements inside the `cfquery` block.
- When referencing text literals in SQL, use single-quotation marks (‘). For example, `SELECT * FROM mytable WHERE FirstName='Jacob'` selects every record from mytable in which the first name is Jacob.
- Surround attribute values with double quotation marks ("attrib_value").
- Make sure that a data source exists in the ColdFusion Administrator before you reference it in a `cfquery` tag .
- Columns and tables that you refer to in your SQL statement must exist, otherwise the query fails.
- Reference the query data by naming the query in one of the presentation tags, such as `cfoutput`, `cfgrid`, `cftable`, `cfgraph`, or `cftree`.
- When ColdFusion returns database columns, it removes table and owner prefixes. For example, if you query Employee.Emp_ID in the query, the Employee, is removed and returns as Emp_ID. You can use an alias to handle duplicate column names; for more information, see Using Query of Queries.
- You cannot use SQL reserved words, such as MIN, MAX, COUNT, in a SQL statement. Because reserved words are database-dependent, see the documentation of your database for a list of reserved words.
- If you use COMPUTE AVG() in your SQL, ColdFusion returns `avg()` as the column name. (Previous versions (ColdFusion 5 and ColdFusion MX 7) returned `ave()` as the column name.)
- To retrieve results returned by database triggers, add the following connection parameter in the connection string:

```
AlwaysReportTriggerResults=true
```

  This parameter determines how the driver reports results generated by database triggers (procedures that are stored in the database and executed, or fired, when a table is modified). For Microsoft SQL Server 2005, this includes triggers fired by Data Definition Language (DDL) events. If set to true, the driver returns all results, including results generated by triggers. Multiple trigger results are returned one at a time. Use the method Statement.getMoreResults to retrieve individual trigger results. Warnings and errors are reported in the results as they are encountered.

Building queries

As discussed earlier, you build queries by using the `cfquery` tag and SQL.

> **Note**
>
> This procedure and many subsequent procedures use the cfdocexamples data source that connects to the cfdocexamples.mdb database. This data source is installed by default. For information on adding or configuring a data source, see Configuring and Administering ColdFusion.

Query the table

1. Create a ColdFusion page with the following content:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<html>
<head>
<title>Employee List</title>
</head>
<body>
<h1>Employee List</h1>
<cfquery name="EmpList" datasource="cfdocexamples">
 SELECT FirstName, LastName, Salary, Contract
 FROM Employee
</cfquery>
</body>
</html>
```

Note

Adobe recommends that you create structured, reusable code by placing queries in ColdFusion_ components; however, for simplicity, the examples here include the query in the body of the_ ColdFusion_ page. For more information about using_ ColdFusion components, see Building and Using ColdFusion Components.

2. Save the page as emplist.cfm in the myapps directory under your *web_root* directory. For example, the default path on a Windows computer would be:C:\CFusion\wwwroot\myapps\
3. Enter the following URL in your web browser:http://localhost/myapps/emplist.cfmOnly the header appears.
4. View the source in the browser.ColdFusion creates the EmpList data set, but only HTML and text return to the browser. When you view the page's source, you see only HTML tags and the heading "Employee List." To display the data set on the page, use code tags and variables to output the data.

Reviewing the code

The query you just created retrieves data from the cfdocexamples database. The following table describes the highlighted code and its function:

Code	Description
`<cfquery name="EmpList" datasource="cfdocexamples">`	Queries the database specified in the cfdocexamples data source.
`SELECT FirstName, LastName, Salary, ContractFROM Employee`	Gets information from the FirstName, LastName, Salary, and Contract fields in the Employee table.
`</cfquery>`	Ends the `cfquery` block.

© 2014 Adobe Systems Incorporated. All rights reserved.

Outputting query data

After you define a query, you can use the `cfoutput` tag with the `query` attribute to output data from the recordset. When you use the `query` attribute, keep in mind the following:

- ColdFusion loops through all the code contained within the `cfoutput` block, once for each row in the recordset returned from the database.
- Reference specific column names within the `cfoutput` block to output the data to the page.
- You can place text, CFML tags, and HTML tags inside or surrounding the `cfoutput` block to format the data on the page.
- Although you do not have to specify the query name when you refer to a query column, use the query name as a prefix for best practices reasons. For example, if you specify the Emplist query in your `cfoutput` tag, you can refer to the Firstname column in the Emplist query as Firstname. However, using the query name as a prefix, Emplist.Firstname, is preferred, and is in the following procedure.
  The `cfoutput` tag accepts a variety of optional attributes but, ordinarily, you use the `query` attribute to define the name of an existing query.

1. Edit emplist.cfm so that it appears as follows:

```
<html>
<head>
<title>Employee List</title>
</head>
<body>
<h1>Employee List</h1>
<cfquery name="EmpList" datasource="cfdocexamples">
 SELECT FirstName, LastName, Salary, Contract
 FROM Employee
</cfquery>
<cfoutput query="EmpList">
#EmpList.FirstName#, #EmpList.LastName#, #EmpList.Salary#,
#EmpList.Contract#<br>
</cfoutput>
</body>
</html>
```

2. Save the file and view it in your web browser:A list of employees appears in the browser, with each line displaying one row of data.

> **Note**
>
> If necessary, refresh your browser to see your changes.

You created a ColdFusion application page that retrieves and displays data from a database. At present, the output is raw and needs formatting. For more information, see Introduction to Retrieving and Formatting Data.

Reviewing the code

The results of the query appear on the page. The following table describes the highlighted code and its function:

Code	Description

© 2014 Adobe Systems Incorporated. All rights reserved.

`<cfoutput query="EmpList">`	Displays information retrieved in the EmpList query.
`#EmpList.FirstName#, #EmpList.LastName#,#EmpList.Salary#, #EmpList.Contract#`	Displays the value of the `FirstName`, `LastName`, `Salary`, and `Contract` fields of each record, separated by commas and spaces.
` `	Inserts a line break (go to the next line) after each record.
`</cfoutput>`	Ends the `cfoutput` block.

Query output notes and considerations

When outputting query results, keep in mind the following guidelines:

- A `cfquery` must retrieve data before the `cfoutput` tag can display its results. Although you can include both on the same page, Adobe recommends that you place queries in ColdFusion components and output the results on a separate page. For more information, see Building and Using ColdFusion Components.
- To output data from all the records of a query, specify the query name by using the `query` attribute in the `cfoutput` tag.
- Columns must exist and be retrieved to the application to output their values.
- Inside a `cfoutput` block that uses a `cfquery` attribute, you can prefix the query variables with the name of the query; for example, `Emplist.FirstName`.
- As with other attributes, surround the `query` attribute value with double-quotation marks (").
- As with any variables that you reference for output, surround column names with number signs (#) to tell ColdFusion to output the current values of the column.
- Add a `<br>` tag to the end of the variable references so that ColdFusion starts a new line for each row that the query returns.

© 2014 Adobe Systems Incorporated. All rights reserved.

Getting information about query results

Each time you query a database with the `cfquery` tag, you get the data (the recordset) and the query variables; together they make up the query object. The following table describes the query variables, which are sometimes called *query properties*:

Variable	Description
`RecordCount`	The total number of records returned by the query.
`ColumnList`	A comma-delimited list of the query columns, in alphabetical order.
`SQL`	The SQL statement executed.
`Cached`	Whether the query was cached.
`SQLParameters`	Ordered array of `cfqueryparam` values.
`ExecutionTime`	Cumulative time required to process the query, in milliseconds.

In your CFML code, use these variables as if they are columns in a database table. Use the `result` attribute to specify the name of the structure that ColdFusion populates with these variables. You then use that structure name to refer to the query variables as the following example shows:

Output information about the query on your page

1. Edit emplist.cfm so that it appears as follows:

```
<cfset Emp_ID = 1>
<cfquery name="EmpList" datasource="cfdocexamples" result="tmpResult">
 SELECT FirstName, LastName, Salary, Contract
 FROM Employee
 WHERE Emp_ID = <cfqueryPARAM value = "#Emp_ID#"
      CFSQLType = "CF_SQL_INTEGER">
</cfquery>
<cfoutput query="EmpList">
 #EmpList.FirstName#, #EmpList.LastName#, #EmpList.Salary#,
#EmpList.Contract#<br>
</cfoutput> <br>
<cfoutput>
 The query returned #tmpResult.RecordCount# records.<br>
 The query columns are:#tmpResult.ColumnList#.<br>
 The SQL is #tmpResult.SQL#.<br>
 Whether the query was cached: #tmpResult.Cached#.<br>
 Query execution time: #tmpResult.ExecutionTime#.<br>
</cfoutput>
<cfdump var="#tmpResult.SQLParameters#">
```

2. Save the file and view it in your web browser:The number of employees now appears below the list of employees. If necessary, refresh your browser and scroll to see the RecordCount output.

© 2014 Adobe Systems Incorporated. All rights reserved.

Reviewing the code

You now display the number of records retrieved in the query. The following table describes the code and its function:

Code	Description
`<cfoutput>`	Displays what follows.
`The query returned`	Displays the text "The query returned".
`#EmpList.RecordCount#`	Displays the number of records retrieved in the EmpList query.
`records.`	Displays the text "records."
`</cfoutput>`	Ends the `cfoutput` block.

Query variable notes and considerations

When using query variables, keep in mind the following guidelines:

- Reference the query variable within a `cfoutput` block so that ColdFusion outputs the query variable value to the page.
- Surround the query variable reference with number signs (#) so that ColdFusion knows to replace the variable name with its current value.
- Do not use the `cfoutput` tag `query` attribute when you output the `RecordCount` or `ColumnList` property. If you do, you get one copy of the output for each row. Instead, prefix the variable with the name of the query.

© 2014 Adobe Systems Incorporated. All rights reserved.

Enhancing security with cfqueryparam

Some DBMSs let you send multiple SQL statements in a single query. However, hackers sometimes try to modify URL or form variables in a dynamic query by appending malicious SQL statements to existing parameters. Be aware of potential security risks when you pass parameters in a query string. These risks can exist in many development environments, including ColdFusion, ASP, and CGI. Using the `cfqueryparam` tag can reduce this risk.

About query string parameters

When you let a query string pass a parameter, ensure that only the expected information is passed. The following ColdFusion query contains a WHERE clause, which selects only database entries that match the last name specified in the LastName field of a form:

```
<cfquery name="GetEmployees" datasource="cfdocexamples">
 SELECT FirstName, LastName, Salary
 FROM Employee
 WHERE LastName='#Form.LastName#'
</cfquery>
```

Someone could call this page with the following malicious URL:
http://myserver/page.cfm?Emp_ID=7%20DELETE%20FROM%20Employee
The result is that ColdFusion tries to execute the following query:

```
<cfquery name="GetEmployees" datasource="cfdocexamples">
 SELECT * FROM Employee
 WHERE Emp_ID = 7 DELETE FROM Employee
</cfquery>
```

In addition to an expected integer for the Emp_ID column, this query also passes malicious string code in the form of a SQL statement. If this query successfully executes, it deletes all rows from the Employee table'something you definitely do not want to enable by this method. To prevent such actions, evaluate the contents of query string parameters.

Using cfqueryparam

You can use the `cfqueryparam` tag to evaluate query string parameters and pass a ColdFusion variable within a SQL statement. This tag evaluates variable values before they reach the database. You specify the data type of the corresponding database column in the `cfsqltype` attribute of the `cfqueryparam` tag. In the following example, because the Emp_ID column in the cfdocexamples data source is an integer, you specify a `cfsqltype` of `cf_sql_integer`:

```
<cfquery name="EmpList" datasource="cfdocexamples">
 SELECT * FROM Employee
 WHERE Emp_ID = <cfqueryparam value = "#Emp_ID#"
       cfsqltype = "cf_sql_integer">
</cfquery>
```

The `cfqueryparam` tag checks that the value of Emp_ID is an integer data type. If anything else in the query string

© 2014 Adobe Systems Incorporated. All rights reserved.

is not an integer, such as a SQL statement to delete a table, the `cfquery` tag does not execute. Instead, the `cfqueryparam` tag returns the following error message:
Invalid data '7 DELETE FROM Employee' for CFSQLTYPE 'CF_SQL_INTEGER'.

Using cfqueryparam with strings

When passing a variable that contains a string to a query, specify a `cfsqltype` value of `cf_sql_char`, and specify the `maxLength` attribute, as in the following example:

```
<cfquery name = "getFirst" dataSource = "cfdocexamples">
 SELECT * FROM employees
 WHERE LastName = <cfqueryparam value = "#LastName#"
        cfsqltype = "cf_sql_char" maxLength = "17">
</cfquery>
```

In this case, `cfqueryparam` performs the following checks:

- It ensures that LastName contains a string.
- It ensures that the string is 17 characters or less.
- It escapes the string with single-quotation marks so that it appears as a single value to the database. Even if a hacker passes a bad URL, it appears as follows:WHERE LastName = 'Smith DELETE FROM MyCustomerTable'.

Using cfSqlType

The following table lists the available SQL types against which you can evaluate the `value` attribute of the `cfqueryparam` tag:

BIGINT	BIT	CHAR	DATE
DECIMAL	DOUBLE	FLOAT	IDSTAMP
INTEGER	LONGVARCHAR	MONEY	MONEY4
NUMERIC	REAL	REFCURSOR	SMALLINT
TIME	TIMESTAMP	TINYINT	VARCHAR

Note

Specifying the `cfsqltype` attribute causes the DBMS to use bind variables, which can greatly enhance performance.

#back to top

© 2014 Adobe Systems Incorporated. All rights reserved.

Updating Your Database

Adobe ColdFusion lets you insert, update, and delete information in a database.

© 2014 Adobe Systems Incorporated. All rights reserved.

About updating your database

ColdFusion was originally developed as a way to readily interact with databases. You can quickly insert, update, and delete the contents of your database by using ColdFusion forms, which are typically a pair of pages. One page displays the form with which your end user enters values; the other page performs the action (insert, update, or delete).

Depending on the extent and type of data manipulation, you can use CFML with or without SQL commands. If you use SQL commands, ColdFusion requires a minimal amount of SQL knowledge.

© 2014 Adobe Systems Incorporated. All rights reserved.

Inserting data

You usually use two application pages to insert data into a database:

- An insert form
- An insert action page
  You can create an insert form with standard HTML form tags or with `cfform` tags (see Creating custom forms with the cfform tag). When the user submits the form, form variables are passed to a ColdFusion action page that performs an insert operation (and whatever else is called for) on the specified data source. The insert action page can contain either a `cfinsert` tag or a `cfquery` tag with a SQL INSERT statement. The insert action page should also contain a confirmation message for the end user.

Creating an HTML insert form

The following procedure creates a form using standard HTML tags.

1. Create a ColdFusion page with the following content:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<html>
<head>
<title>Insert Data Form</title>
</head>

<body>
<h2>Insert Data Form</h2>

<table>
<!--- begin html form;
put action page in the "action" attribute of the form tag. --->
<form action="insert_action.cfm" method="post">
<tr>
 <td>Employee ID:</td>
 <td><input type="text" name="Emp_ID" size="4" maxlength="4"></td>
</tr>
<tr>
 <td>First Name:</td>
 <td><input type="Text" name="FirstName" size="35" maxlength="50"></td>
</tr>
<tr>
 <td>Last Name:</td>
 <td><input type="Text" name="LastName" size="35" maxlength="50"></td>
</tr>
<tr>
 <td>Department Number:</td>
 <td><input type="Text" name="Dept_ID" size="4" maxlength="4"></td>
</tr>
<tr>
 <td>Start Date:</td>
 <td><input type="Text" name="StartDate" size="16" maxlength="16"></td>
</tr>
<tr>
 <td>Salary:</td>
 <td><input type="Text" name="Salary" size="10" maxlength="10"></td>
</tr>
<tr>
 <td>Contractor:</td>
 <td><input type="checkbox" name="Contract" value="Yes" checked>Yes</td>
</tr>
<tr>
 <td> </td>
 <td><input type="Submit" value="Submit"> <input type="Reset"
value="Clear Form"></td>
</tr>
</form>
<!--- end html form --->
</table>

</body>
</html>
```

2. Save the file as insert_form.cfm in the myapps directory under your *web_root* and view it in your web browser.

© 2014 Adobe Systems Incorporated. All rights reserved.

> **Note**
>
> The form does not work until you write an action page for it. For more information, see *Creating an action page to insert data* in Inserting data.

Data entry form notes and considerations

If you use the `cfinsert` tag in the action page to insert the data into the database, follow these rules for creating the form page:

- Create HTML form fields for only the database columns into which you insert data.
- By default, `cfinsert` inserts all of the form's fields into the database columns with the same names. For example, it places the Form.Emp_ID value in the database Emp_ID column. The tag ignores form fields that lack corresponding database column names.

> **Note**
>
> You can also use the `formfields` attribute of the `cfinsert` tag to specify which fields to insert; for example, `formfields="prod_ID,Emp_ID,status"`.

Creating an action page to insert data

You can use the `cfinsert` tag or the `cfquery` tag to create an action page that inserts data into a database.

Creating an insert action page with cfinsert

The `cfinsert` tag is the easiest way to handle simple inserts from either a `cfform` or an HTML form. This tag inserts data from all the form fields with names that match database field names.

1. Create a ColdFusion page with the following content:

```
<html>
<head> <title>Input form</title> </head>

<body>
<!--- If the Contractor check box is clear,
 set the value of the Form.Contract to "No" --->
<cfif not isdefined("Form.Contract")>
 <cfset Form.Contract = "N">
</cfif>

<!--- Insert the new record --->
<cfinsert datasource="cfdocexamples" tablename="EMPLOYEE">

<h1>Employee Added</h1>
<cfoutput> You have added #Form.FirstName# #Form.Lastname# to the employee
database.
</cfoutput>

</body>
</html>
```

2. Save the page as insert_action.cfm.

© 2014 Adobe Systems Incorporated. All rights reserved.

3. View insert_form.cfm in your web browser and enter values.

> **Note**
>
> You might want to compare views of the Employee table in the cfdocexamples data source before and after inserting values in the form.

4. Click Submit. ColdFusion inserts your values into the Employee table and displays a confirmation message.

Reviewing the code

The following table describes the code and its function:

Code	Description
`<cfif not isdefined("Form.Contract")> <cfset Form.Contract = "N"> </cfif>`	Sets the value of `Form.Contract` to `No` if it is not defined. If the Contractor check box is unchecked, no value is passed to the action page; however, the database field must have some value.
`<cfinsert datasource="cfdocexamples" tablename="EMPLOYEE">`	Creates a row in the Employee table of the cfdocexamples database. Inserts data from the form into the database fields with the same names as the form fields.
`<cfoutput>You have added #Form.FirstName# #Form.Lastname# to the employee database.</cfoutput>`	Informs the user that values were inserted into the database.

> **Note**
>
> If you use form variables in `cfinsert` or `cfupdate` tags, ColdFusion automatically validates any form data it sends to numeric, date, or time database columns. You can use the hidden field validation functions for these fields to display a custom error message. For more information, see Introduction to Retrieving and Formatting Data.

Creating an insert action page with cfquery

For more complex inserts from a form submittal, you can use a SQL INSERT statement in a `cfquery` tag instead of using a `cfinsert` tag. The SQL INSERT statement is more flexible because you can insert information selectively or use functions within the statement.
The following procedure assumes that you have created the insert_action.cfm page, as described in *Creating an insert action page with cfinsert* in Inserting data.

© 2014 Adobe Systems Incorporated. All rights reserved.

1. In insert_action.cfm, replace the `cfinsert` tag with the following highlighted `cfquery`code:

```
<html>
<head>
 <title>Input form</title>
</head>

<body>
<!--- If the Contractor check box is clear), set the value of the
Form.Contract
 to "No" --->
<cfif not isdefined("Form.Contract")>
 <cfset Form.Contract = "No">
</cfif>

<!--- Insert the new record --->
<cfquery name="AddEmployee" datasource="cfdocexamples">
 INSERT INTO Employee
 VALUES (#Form.Emp_ID#, '#Form.FirstName#',
 '#Form.LastName#', #Form.Dept_ID#,
 '#Form.StartDate#', #Form.Salary#, '#Form.Contract#')
</cfquery>

<h1>Employee Added</h1>
<cfoutput>You have added #Form.FirstName# #Form.Lastname# to the employee
database.
</cfoutput>

</body>
</html>
```

2. Save the page.
3. View insert_form.cfm in your web browser and enter values.
4. Click Submit. ColdFusion inserts your values into the Employee table and displays a confirmation message.

Reviewing the code

The following table describes the highlighted code and its function:

Code	Description

`<cfquery name="AddEmployee" datasource="cfdocexamples">`
`INSERT INTO Employee VALUES`
`(#Form.Emp_ID#,`
`'#Form.FirstName#',`
`'#Form.LastName#',`
`#Form.Dept_ID#,`
`'#Form.StartDate#',`
`#Form.Salary#, '#Form.Contract#')`
`</cfquery>`
	Inserts a new row into the Employee table of the cfdocexamples database. Specifies each form field to be added.Because you are inserting data into all database fields in the same left-to-right order as in the database, you do not have to specify the database field names in the query.Because `#From.Emp_ID#`, `#Form.Dept_ID#`, and `#Form.Salary#` are numeric, they do not need to be enclosed in quotation marks.

© 2014 Adobe Systems Incorporated. All rights reserved.

Inserting into specific fields

The preceding example inserts data into all the fields of a table (the Employee table has seven fields). There might be times when you do not want users to add data into all fields. To insert data into specific fields, the SQL statement in the `cfquery` must specify the field names following both INSERT INTO and VALUES. For example, the following `cfquery` omits salary and start date information from the update. Database values for these fields are 0 and NULL, respectively, according to the database's design.

```
<cfquery name="AddEmployee" datasource="cfdocexamples">
 INSERT INTO Employee
  (Emp_ID,FirstName,LastName,
  Dept_ID,Contract)
 VALUES
  (#Form.Emp_ID#,'#Form.FirstName#','#Form.LastName#',
  #Form.Dept_ID#,'#Form.Contract#')
</cfquery>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

Updating data

You usually use the following two application pages to update data in a database:

- An update form
- An update action page
  You can create an update form with `cfform` tags or HTML form tags. The update form calls an update action page, which can contain either a `cfupdate` tag or a `cfquery` tag with a SQL UPDATE statement. The update action page should also contain a confirmation message for the end user.

Creating an update form

The following are the key differences between an update form and an insert form:

- An update form contains a reference to the primary key of the record that is being updated. A *primary key* is a fields in a database table that uniquely identifies each record. For example, in a table of employee names and addresses, only the Emp_ID is unique to each record.
- An update form is populated with existing record data.
  The easiest way to designate the primary key in an update form is to include a hidden input field with the value of the primary key for the record you want to update. The hidden field indicates to ColdFusion which record to update.

1. Create a ColdFusion page with the following content:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<html>
<head>
<title>Update Form</title>
</head>

<body>
<cfquery name="GetRecordtoUpdate" datasource="cfdocexamples">
 SELECT * FROM Employee
 WHERE Emp_ID = #URL.Emp_ID#
</cfquery>


<cfoutput query="GetRecordtoUpdate">
<table>
<form action="update_action.cfm" method="Post">
 <input type="Hidden" name="Emp_ID" value="#Emp_ID#"><br>
<tr>
 <td>First Name:</td>
 <td><input type="text" name="FirstName" value="#FirstName#"></td>
</tr>
<tr>
 <td>Last Name:</td>
 <td><input type="text" name="LastName" value="#LastName#"></td>
</tr>
<tr>
 <td>Department Number:</td>
 <td><input type="text" name="Dept_ID" value="#Dept_ID#"></td>
</tr>
<tr>
 <td>Start Date:</td>
 <td><input type="text" name="StartDate" value="#StartDate#"></td>
</tr>
<tr>
 <td>Salary:</td>
 <td><input type="text" name="Salary" value="#Salary#"></td>
</tr>
<tr>
 <td>Contractor:</td>
 <td><cfif #Contract# IS "Yes">
  <input type="checkbox" name="Contract" checked>Yes
 <cfelse>
  <input type="checkbox" name="Contract">Yes
 </cfif></td>
</tr>
<tr>
 <td> </td>
 <td><input type="Submit" value="Update Information"></td>
</tr>
</form>
</table>
</cfoutput>

</body>
</html>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

2. Save the file as update_form.cfm.
3. View update_form.cfm in your web browser by specifying the page URL and an Employee ID; for example, enter the following: *http://localhost/myapps/update_form.cfm?Emp_ID=3*

> **Note**
>
> Although you can view an employee's information, code an action page before you can update the database. For more information, see *Creating an action page to update data* below.

Reviewing the code

The following table describes the code and its function:

Code	Description

`<cfquery name="GetRecordtoUpdate" datasource="cfdocexamples">`
`SELECT * FROM Employee`
`WHERE Emp_ID = #URL.Emp_ID#`
`</cfquery>`
	Queries the cfdocexamples data source and returns records in which the employee ID matches what was entered in the URL that called this page.

`<cfoutput query="GetRecordtoUpdate">`
`...`
`</cfoutput>`
	Makes available as variables the results of the GetRecordtoUpdate query in the form created in subsequent lines.

`<form action="update_action.cfm" method="Post">`
`...`
`</form>`
	Creates a form whose variables are processed on the update_action.cfm action page.

`<input type="Hidden" name="Emp_ID" value="#Emp_ID#"> `	Uses a hidden input field to pass the Emp_ID (primary key) value to the action page.

© 2014 Adobe Systems Incorporated. All rights reserved.

`First Name:` `<input type="text"` `name="FirstName"` `value="#FirstName#"> ` `Last Name:` `<input type="text" name="LastName"` `value="#LastName#"> ` `Department Number:` `<input type="text" name="Dept_ID"` `value="#Dept_ID#"> ` `Start Date:` `<input type="text"` `name="StartDate"` `value="#StartDate#"> ` `Salary:` `<input type="text" name="Salary"` `value="#Salary#"> `	Populates the fields of the update form. This example does not use ColdFusion formatting functions. As a result, start dates look like 1985-03-12 00:00:00 and salaries do not have dollar signs or commas. The user can replace the information in any field using any valid input format for the data.
`Contracto r:` `< cfif #Contract# IS "Yes">` `<input type="checkbox"  name="C` `ontract" checked>Yes ` `<cfelse>` `<input type="checkbox"` `name="Contract"> Yes  ` `</cfif>` ` ` `<input type="Submit" value="Update` `Information">` `</form>` `</cfoutput>`	The Contract field requires special treatment because a check box appears and sets its value. The cfif structure puts a check mark in the check box if the Contract field value is `Yes`, and leaves the box empty otherwise.

Creating an action page to update data

You can create an action page to update data with either the `cfupdate` tag or `cfquery` with the UPDATE statement.

Creating an update action page with cfupdate

The `cfupdate` tag is the easiest way to handle simple updates from a front-end form. The `cfupdate` tag has an almost identical syntax to the `cfinsert` tag.
To use the `cfupdate` tag, include the primary key fields in your form submittal. The `cfupdate` tag automatically detects the primary key fields in the table that you are updating and looks for them in the submitted form fields. ColdFusion uses the primary key fields to select the record to update (therefore, you cannot update the primary key value itself). It then uses the remaining form fields that you submit to update the corresponding fields in the record. Your form only needs to have fields for the database fields that you want to change.

1. Create a ColdFusion page with the following content:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<html>
<head>
 <title>Update Employee</title>
</head>
<body>
<cfif not isdefined("Form.Contract")>
 <cfset form.contract = "N">
<cfelse>
 <cfset form.contract = "Y">
</cfif>

<cfupdate datasource="cfdocexamples" tablename="EMPLOYEE">

<h1>Employee Updated</h1>
<cfoutput>
You have updated the information for #Form.FirstName# #Form.LastName# in the
employee
 database.
</cfoutput>

</body>
</html>
```

2. Save the page as update_action.cfm.
3. View update_form.cfm in your web browser by specifying the page URL and an Employee ID; for example, enter the following: *http://localhost/myapps/update_form.cfm?Emp_ID=3*
4. Enter new values in any of the fields, and click Update Information.ColdFusion updates the record in the Employee table with your new values and displays a confirmation message.

Reviewing the code

The following table describes the code and its function:

Code	Description

`<cfif not isdefined("Form.Contract")>`
`<cfset Form.contract = "N">`
`<cfelse>`
`<cfset form.contract = "Y">`
`</cfif>`
	Sets the value of `Form.Contract` to `No` if it is not defined, or to `Yes` if it is defined. If the Contractor check box is unchecked, no value is passed to the action page; however, the database field must have some value.

`<cfupdate datasource="cfdocexamples" tablename="EMPLOYEE">`	Updates the record in the database that matches the primary key on the form (Emp_ID). Updates all fields in the record with names that match the names of form controls.

© 2014 Adobe Systems Incorporated. All rights reserved.

`<cfoutput>`
`You have updated the information`
`for #Form.FirstName#`
`#Form.LastName# in the employee`
`database.`
`</cfoutput>`
	Informs the user that the change was made successfully.

Creating an update action page with cfquery

For more complicated updates, you can use a SQL UPDATE statement in a `cfquery` tag instead of a `cfupdate` tag. The SQL UPDATE statement is more flexible for complicated updates.
The following procedure assumes that you have created the update_action.cfm page as described in Creating an update action page with cfupdate.

1. In update_action.cfm, replace the `cfupdate` tag with the following highlighted `cfquery`code:

```
<html>
<head>
 <title>Update Employee</title>
</head>
<body>
<cfif not isdefined("Form.Contract")>
 <cfset form.contract = "No">
<cfelse>
 <cfset form.contract = "Yes">
</cfif>

<!--- cfquery requires date formatting when retrieving from
Access. Use the left function when setting StartDate to trim
the ".0" from the date when it first appears from the
Access database --->
 <cfquery name="UpdateEmployee" datasource="cfdocexamples">
 UPDATE Employee
 SET FirstName = '#Form.Firstname#',
  LastName = '#Form.LastName#',
  Dept_ID = #Form.Dept_ID#,
  StartDate = '#left(Form.StartDate,19)#',
  Salary = #Form.Salary#
 WHERE Emp_ID = #Form.Emp_ID#
</cfquery>

<h1>Employee Updated</h1>
<cfoutput>
You have updated the information for
#Form.FirstName# #Form.LastName#
in the employee database.
</cfoutput>
</body>
</html>
```

2. Save the page.
3. View update_form.cfm in your web browser by specifying the page URL and an Employee ID; for example,

© 2014 Adobe Systems Incorporated. All rights reserved.

enter the following: *http://localhost/myapps/update_form.cfm?Emp_ID=3*

4. Enter new values in any of the fields, and click Update Information.ColdFusion updates the record in the Employee table with your new values and displays a confirmation message.
   When the `cfquery` tag retrieves date information from a Microsoft Access database, it displays the date and time with tenths of seconds, as follows:

© 2014 Adobe Systems Incorporated. All rights reserved.

Deleting data

You use a `cfquery` tag with a SQL DELETE statement to delete data from a database. ColdFusion has no `cfdelete` tag.

Deleting a single record

To delete a single record, use the table's primary key in the WHERE condition of a SQL DELETE statement. In the following procedure, Emp_ID is the primary key, so the SQL Delete statement is as follows:

```
DELETE FROM Employee WHERE Emp_ID = #Form.Emp_ID#
```

You often want to see the data before you delete it. The following procedure displays the data to be deleted by reusing the form page used to insert and update data. Any data that you enter in the form before submitting it is not used, so you can use a table to display the record to be deleted instead.

1. In update_form.cfm, change the title to "Delete Form" and the text on the submit button to "Delete Record".
2. Change the `form`tag so that it appears as follows:

```
<form action="delete_action.cfm" method="Post">
```

3. Save the modified file as delete_form.cfm.
4. Create a ColdFusion page with the following content:

```
<html>
<head>
<title>Delete Employee Record</title>
</head>
<body>

<cfquery name="DeleteEmployee"
 datasource="cfdocexamples">
  DELETE FROM Employee
  WHERE Emp_ID = #Form.Emp_ID#
</cfquery>

<h1>The employee record has been deleted.</h1>
<cfoutput>
You have deleted #Form.FirstName# #Form.LastName# from the employee database.
</cfoutput>
</body>
</html>
```

5. Save the page as delete_action.cfm.
6. View delete_form.cfm in your web browser by specifying the page URL and an Employee ID; for example, enter the following: **http://localhost/myapps/delete_form.cfm?Emp_ID=3.Click** **Delete Record** ColdFusion deletes the record in the Employee table and displays a confirmation message.

Reviewing the code

© 2014 Adobe Systems Incorporated. All rights reserved.

The following table describes the code and its function:

Code	Description

`<cfquery name="DeleteEmployee" datasource="cfdocexamples">`
`DELETE FROM Employee WHERE Emp_ID`
`= #Form.Emp_ID#`
`</cfquery>`
	Deletes the record in the database whose Emp_ID column matches the Emp_ID (hidden) field on the form. Since the Emp_ID is the table's primary key, only one record is deleted.

`<cfoutput>`
`You have deleted #Form.FirstName#`
`#Form.LastName# from the employee`
`database.`
`</cfoutput>`
	Informs the user that the record was deleted.

Deleting multiple records

You can use a SQL condition to delete several records. The following example deletes the records for everyone in the Sales department (which has Dept_ID number 4) from the Employee table:

```
DELETE FROM Employee WHERE Dept_ID = 4
```

To delete all the records from the Employee table, use the following code:

```
DELETE FROM Employee
```

Deleting records from a database is not reversible. Use DELETE statements carefully.

© 2014 Adobe Systems Incorporated. All rights reserved.

Database-related enhancements in ColdFusion 10

- Track client information
- Support for new SQL types
- Miscellaneous enhancements that include enhancements to connection validation and exception handling

Tracking client information

To enable auditing on database, while performing a database operation, you can track certain client information, such as application name or client ID. The supported client information varies from database to database. For example, Oracle supports client information in a table named `$v_session`. Other databases store it in local cache or registers.
You can send the client information using the `ClientInfo` attribute in the following ColdFusion tags: `cfquery`, `cfupdate`, `cfinsert`, and `cfstoredproc`. The information sent is set before executing the query.
The following are the values supported by the `ClientInfo` attribute:

- AccountingInfo
- Action: The action performed by the query.
- ApplicationName: Application name.
- ClientHostName: The host from where the query is executed.
- ClientID: The client ID.
- ClientUser: The user ID.
- ProgramID: The program ID.
- Module: The module name.
  Use the `cfdbinfo` tag to find the supported `clientInfo` properties.
  Example

```
<cfscript>
 clientInfo = structNew();
 clientInfo.AccountingInfo = "MyAccount_cfquery";
 clientInfo.Action = "cfstoredproc_cfquery";
 clientInfo.ApplicationName = "testApp_cfquery";
 clientInfo.ClientHostName = "Testserver_cfquery";
 clientInfo.ClientID = "testID_cfquery";
 clientInfo.ClientUser = "cfadmin_cfquery";
 clientInfo.ProgramID = 1234;
 clientInfo.Module = "test_query";
</cfscript>
<cfquery name="qName"  datasource="#regdatasource#" clientInfo="#clientInfo#">
Select * from employees
</cfquery>
```

Accessing client information metadata

The `cfdbinfo` tag supports the `ClientInfo` value for the `type` attribute. If this value is specified, the metadata supported for the specified data source is returned.

```
<CFDBINFO type="clientinfo" datasource="#regdatasource#" name="result">
<cfdump var="#result#">
```

Accessing client information

© 2014 Adobe Systems Incorporated. All rights reserved.

When you perform a database operation using the `fetchclientinfo` attribute, you can access the database-specific client information. The following ColdFusion tags support the `fetchclientinfo` attribute: `cfquery`, `cfinsert`, and `cfstoredproc`. If set to `true`, the attribute returns a struct with the key-value pair passed by the last query.Example

```
<cfquery fetchclientinfo=true result="resultQ" datasource="#regdatasource#">
select * from employees
</cfquery>
```

Passing client information using ColdFusion administrator

Using the ColdFusion administrator's Data Sources Advanced Settings page, you can set the database-specific information. If specified, the following information is sent to the database before executing the query.

- Client Hostname: The host name from where the query is executed.
- Client Username: The user name if the user is logged in using the `<cflogin>` tag.
- Application Name: The application name specified in the `application.cfc`.
- Prefix: If specified, the value is prefixed with the application name specified in `application.cfc.`

> **Note**
>
> If the same client info properties are specified in the query tag, it takes the precedence over the server-level settings.

New data type support for CFSQLType

The `cfqueryparam` and `cfprocparam` tags support the following SQL types:

- CF_SQL_NCHAR
- CF_SQL_NVARCHAR
- CF_SQL_LONGNVARCHAR
- CF_SQL_NCLOB
- CF_SQL_SQLXML

Miscellaneous enhancements

Following are the miscellaneous enhancements:

Validating connection before executing a query

In the ColdFusion administrator's Data Sources Advanced Settings page, you can set the option to validate the connection. If this option is set, ColdFusion validates the database connection before executing a query. Setting this option can have an impact on the performance of the application.You can optionally specify a query to validate the connection. If specified, the connection is validated based on it. If not specified, the default mechanism is used to validate the connection.

Improved exception handling

The `<cfcatch>` tag's `type=database` property is improved for better exception handling.The `#CFCATCH.exceptions#` provides details in a list of structs. If multiple exceptions are thrown, it provides multiple elements. Each element provides information in the following categories: class, messages, and list of causes (if any).Example

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cftry>
 <cfquery datasource="badmysql" timeout="2">
    Select * from employees
 </cfquery>
<cfcatch type="database">
 <cfdump var="#cfcatch#">
</cfcatch>
</cftry>
```

#back to top

© 2014 Adobe Systems Incorporated. All rights reserved.

Using Query of Queries

A query that retrieves data from a recordset is called a *Query of Queries*. After you generate a recordset, you can interact with its results as if they are database tables by using Query of Queries.

© 2014 Adobe Systems Incorporated. All rights reserved.

About recordsets

Query of Queries is based on manipulating the recordset, which you can create using the cfquery tag and other ways.
When you execute a database query, Adobe ColdFusion retrieves the data in a *recordset*. In addition to presenting recordset data to the user, you can manipulate this recordset to improve the performance of your application. Because a recordset contains rows (records) and columns (fields), you can think of it as a virtual database table, or as a spreadsheet. For example, the cfpop tag retrieves a recordset in which each row is a message and each column is a message component, such as To, From, and Subject.

Creating a recordset

You can perform a Query of Queries on any ColdFusion tag or function that generates a recordset, including the following:

- cfcollection
- cfdirectory
- cfftp
- cfhttp
- cfindex
- cfldap
- cfmail
- cfpop
- cfprocresult
- cfquery(against a database or against another Query of Queries)
- cfsearch
- cfstoredproc
- cfwddx
- The QueryNew function

Creating a recordset with the QueryNew() function

In addition to creating a recordset by using a cfquery or other CFML tags, you can create it with the QueryNew fun ction.

1. Create a ColdFusion page with the following content:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<html>
<head>
<title>The queryNew function</title>
</head>
<body>
<h2>QueryNew Example</h2>

<!--- Create a query, specify data types for each column. --->
<cfset qInstruments = queryNew("name, instrument, years_playing",
 "CF_SQL_VARCHAR, CF_SQL_VARCHAR, CF_SQL_INTEGER")>

<!--- Add rows. --->
<cfset newrow = queryaddrow(qInstruments, 3)>

<!--- Set values in cells. --->
<cfset temp = querysetcell(qInstruments, "name", "Thor", 1)>
<cfset temp = querysetcell(qInstruments, "instrument", "hammer", 1)>
<cfset temp = querysetcell(qInstruments, "years_playing", "1000", 1)>

<cfset temp = querysetcell(qInstruments, "name", "Bjorn", 2)>
<cfset temp = querysetcell(qInstruments, "instrument", "sitar", 2)>
<cfset temp = querysetcell(qInstruments, "years_playing", "24", 2)>

<cfset temp = querysetcell(qInstruments, "name", "Raoul", 3)>
<cfset temp = querysetcell(qInstruments, "instrument", "flute", 3)>
<cfset temp = querysetcell(qInstruments, "years_playing", "12", 3)>

<!--- Output the query. --->
<cfoutput query="qInstruments">
 <pre>#name# #instrument# #years_playing#</pre>
</cfoutput>

<h3>Individual record retrieval:</h3>
<cfoutput>
<p>#qInstruments.name[2]# has played #qInstruments.instrument[2]# for
 #qInstruments.years_playing[2]# years.</p>
</cfoutput>

</body>
</html>
```

2. Save the page as queryNew.cfm in the myapps directory under the *web_root* directory.
3. Display queryNew.cfm in your browser

© 2014 Adobe Systems Incorporated. All rights reserved.

About Query of Queries

After you have created a recordset with a tag or function, you can retrieve data from the recordset in one or more dependent queries. A query that retrieves data from a recordset is called a Query of Queries. A typical use of a Query of Queries is to retrieve an entire table into memory with one query, and then access the table data (the recordset) with subsequent sorting or filtering queries. In essence, you query the recordset as if it is a database table.

> **Note**
>
> Because you can generate a recordset in ways other than using the `cfquery` tag, the term In Memory Query is sometimes used instead of Query of Queries.

Benefits of Query of Queries

Performing a Query of Queries has many benefits, including the following:

1. When you have to access the same tables multiple times, you greatly reduce access time, because the data is already in memory (in the recordset). A Query of Queries is ideal for tables of 5,000 to 50,000 rows, and is limited only by the memory of the ColdFusion host computer.
2. You can perform joins and union operations on results from different data sources. For example, you can perform a union operation on queries from different databases to eliminate duplicates for a mailing list.
3. You can efficiently manipulate cached query results in different ways. You can query a database once, and then use the results to generate several different summary tables.For example, to summarize the total salary by department, by skill, and by job, you can make one query to the database and use its results in three separate queries to generate the summaries.
4. You can obtain drill-down, master-detail information for which you do not access the database for the details.For example, you can select information about departments and employees in a query, and cache the results. You can then display the names of the employees. When users select an employee, the application displays the details of the employees by selecting information from the cached query, without accessing the database.
5. You can use a Query of Queries in report definitions to generate subreport data. For more information, see *U sing subreports* in Common reporting tasks and techniques.

Performing a Query of Queries

Perform a Query of Queries as follows:

1. Generate a recordset through a *master query*.You can write a master query using a tag or function that creates a recordset. For more information, see *Creating a recordset* in About recordsets.
2. Write a *detail query,* a `cfquery` tag that specifies `dbtype="query"`.
3. In the detail query, write a SQL statement that retrieves the relevant records. Specify the names of one or more existing queries as the table names in your SQL code. Do not specify a `datasource` attribute.
4. If the database content does not change rapidly, use the `cachedwithin` attribute of the master query to cache the query results between page requests. This way, ColdFusion accesses the database on the first page request, and does not query the database again until the specified time expires. Use the `CreateTimeSpan` function to specify the `cachedwithin` attribute value (in days, hours, minutes, seconds format).
   The detail query generates a new query result set, identified by the value of the `name` attribute of the detail query. The following example illustrates the use of a master query and a single detail query that extracts information from the master.

Use the results of a query in a query

1. Create a ColdFusion page with the following content:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<h1>Employee List</h1>
<!--- LastNameSearch (normally generated interactively) --->
<cfset LastNameSearch="Doe">

<!--- Master Query --->
<cfquery datasource="cfdocexamples" name="master"
 cachedwithin=#CreateTimeSpan(0,1,0,0)#>
 SELECT * from Employee
</cfquery>

<!--- Detail Query (dbtype=query, no data source) --->
<cfquery dbtype="query" name="detail">
 SELECT Emp_ID, FirstName, LastName
 FROM master
 WHERE LastName=<cfqueryparam value="#LastNameSearch#"
cfsqltype="cf_sql_char" maxLength="20"></cfquery>

<!--- output the detail query results --->
<p>Output using a query of query:</p>
<cfoutput query=detail>
 #Emp_ID#: #FirstName# #LastName#<br>
</cfoutput>

<p>Columns in the master query:</p>
<cfoutput>
 #master.columnlist#<br>
</cfoutput>

<p>Columns in the detail query:</p>
<cfoutput>
 #detail.columnlist#<br>
</cfoutput>
```

2. Save the page as query_of_query.cfm in the myapps directory under the *web_root*.
3. Display query_of_query.cfm in your browser

Reviewing the code

The master query retrieves the entire Employee table from the cfdocexamples data source. The detail query selects only the three columns to display for employees with the specified last name. The following table describes the code and its function:

Code	Description
`cfset LastNameSearch="Doe"`	Sets the last name to use in the detail query. In a complete application, this information comes from user interaction.

© 2014 Adobe Systems Incorporated. All rights reserved.

`<cfquery datasource="cfdocexamples" name="master" cachedwithin=#CreateTimeSpan(0,1,0,0)#> SELECT * from Employee </cfquery>`	Queries the cfdocexamples data source and selects all data in the Employees table. Caches the query data between requests to this page, and does not query the database if the cached data is less than an hour old.
`<cfquery dbtype="query" name="detail"> SELECT Emp_ID, FirstName, LastName FROM master WHERE LastName=<cfqueryparam value="#LastNameSearch#" cfsqltype="cf_sql_char" maxLength="20">  </cfquery>`	Uses the master query as the source of the data in a new query, named detail. This new query selects only entries that match the last name specified by the `LastNameSearch` variable. The query also selects only three columns of data: employee ID, first name, and last name. The query uses the cfqueryparam tag to prevent passing erroneous or harmful code.
`<cfoutput query=detail> #Emp_ID#: #FirstName# #LastName#   </cfoutput>`	Uses the detail query to display the list of employee IDs, first names, and last names.
`<cfoutput> #master.columnlist#  </cfoutput>`	Lists all the columns returned by the master query.
`<cfoutput> #detail.columnlist#  </cfoutput>`	Lists all the columns returned by the detail query.

Displaying recordset data incrementally

If your database is large, you can limit the number of rows displayed at one time. The following example shows how to do this using the `currentRow` query variable of a Query of Queries. For more information on query variables, see Getting information about query results.

1. Create a ColdFusion page with the following content:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<html>
<head>
<title>QoQ with incremental row return</title>
</head>

<body>
<h3>QoQ with incremental row return</h3>
<!--- define startrow and maxrows to facilitate 'next N' style browsing --->
<cfparam name = "MaxRows" default = "5">
<cfparam name = "StartRow" default = "1">

<!--- master query: retrieve all info from Employee table --->
<cfquery name = "GetSals" datasource = "cfdocexamples">
 SELECT * FROM Employee
 ORDER BY LastName
</cfquery>

<!--- detail query: select 3 fields from the master query --->
<cfquery name = "GetSals2" dbtype = "query">
 SELECT FirstName, LastName, Salary
 FROM GetSals
 ORDER BY LastName
</cfquery>

<!--- build table to display output --->
<table cellpadding = 1 cellspacing = 1>
 <tr>
 <td bgcolor = f0f0f0>
 <b><i> </i></b>
 </td>

 <td bgcolor = f0f0f0>
 <b><i>FirstName</i></b>
 </td>

 <td bgcolor = f0f0f0>
 <b><i>LastName</i></b>
 </td>

 <td bgcolor = f0f0f0>
 <b><i>Salary</i></b>
 </td>
 </tr>

<!--- Output the query and define the startrow and maxrows
  parameters. Use the query variable currentRow to
  keep track of the row you are displaying. --->
 <cfoutput query = "GetSals2" startrow = "#StartRow#" maxrows = "#MaxRows#">
 <tr>
 <td valign = top bgcolor = ffffed>
  <b>#GetSals2.currentRow#</b>
 </td>

 <td valign = top>
  <font size = "-1">#FirstName#</font>
 </td>

 <td valign = top>
  <font size = "-1">#LastName#</font>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

```
 </td>

 <td valign = top>
  <font size = "-1">#LSCurrencyFormat(Salary)#</font>
 </td>
 </tr>
 </cfoutput>
<!--- If the total number of records is less than or equal to
 the total number of rows, provide a link to the same page, with the
 StartRow value incremented by MaxRows (5, in this example) --->
 <tr>
 <td colspan = 4>
 <cfif (startrow + maxrows) lte getsals2.recordcount>
 <a href="qoq_next_row.cfm?startrow=<cfoutput>#Evaluate(StartRow +
  MaxRows)#</cfoutput>">See next <cfoutput>#MaxRows#</cfoutput>
  rows</a>
 </cfif>
 </td>
 </tr>
</table>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

```
</body>
</html>
```

2. Save the page as qoq_next_row.cfm in the myapps directory under the *web_root*.
3. Display qoq_next_row.cfm in your browser.

Using the cfdump tag with query results

As you debug your CFML code, you can use the `cfdump` tag to quickly display the contents of your query. This tag has the following format:

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

For more information on the `cfdump` tag, see the *CFML Reference*.

Using Query of Queries with non-SQL recordsets

A Query of Queries can operate on any CFML tag or function that returns a recordset; you are not limited to operating on `cfquery` results. You can perform queries on non-SQL recordsets, such as a `cfdirectory` tag, a `cfsearch` tag, a `cfldap` tag, and so on.

© 2014 Adobe Systems Incorporated. All rights reserved.

Query of Queries user guide

If you know SQL or have interacted with databases, you might be familiar with some of the Query of Queries functionality.

Using dot notation

Query of Queries supports using dot notation in table names.

Example

If a structure named A contains a field named B, which contains a table named Products, you can refer to the table with dot notation, as follows:

```
SELECT tape_ID, length
FROM A.B.Products;
```

Using joins

A join operation uses a single SELECT statement to return a result set from multiple, related tables, typically those tables with a primary key - foreign key relationship. The two SQL clauses that perform joins are:

- **WHERE clause**: Query of Queries supports joins through a WHERE clause.
- **INNER JOIN and OUTER JOIN**: Query of Queries does not support joins through INNER JOIN or OUTER JOIN clauses.

> **Note**
>
> Query of Queries supports joins between two tables only.

Using unions

The UNION operator lets you combine the results of two or more SELECT expressions into a single recordset. The original tables must have the same number of columns, and corresponding columns must be UNION-compatible data types. Columns are UNION-compatible data types if they meet one of the following conditions:

- The same data type; for example, both Tinyint
- Both Numeric; for example, Tinyint, Smallint, Integer, Bigint, Double, Float, Real, Decimal, or Numeric
- Both Characters; for example, Char, Varchar, or LongVarchar
- Both Dates; for example, Time, TimeStamp, or Date

> **Note**
>
> Query Of Queries does not support ODBC-formatted dates and times.

Syntax

```
select_expression = select_expression UNION [ALL] select_expression
```

Example

© 2014 Adobe Systems Incorporated. All rights reserved.

This example uses the following tables:

Table1	
Type(int)	Name(varchar)
1	Tennis
2	Baseball
3	Football

Table2	
ID(int)	Sport(varchar)
3	Football
4	Volleyball
5	PingPong

To combine Table1 and Table2, use a UNION statement, as follows:

```
SELECT * FROM Table1
UNION
SELECT * FROM Table2
```

The UNION statement produces the following result (UNION) table:

Result table	
Type(int)	Name(varchar)
1	Tennis
2	Baseball
3	Football
4	Volleyball
5	PingPong

Using aliases for column names

© 2014 Adobe Systems Incorporated. All rights reserved.

The column names of a UNION table are the column names in the result set of the first SELECT statement in the UNION operation; Query of Queries ignores the column names in the other SELECT statement. To change the column names of the result table, you can use an alias, as follows:

```
Select Type as SportType, Name as SportName from Table1
UNION
Select * from Table2
```

Duplicate rows and multiple tables

By default, the UNION operator removes duplicate rows from the result table. If you use the keyword ALL, then duplicates are included.
You can combine an unlimited number of tables using the UNION operator, for example:

```
Select * from Table1
UNION
Select * from Table2
UNION
Select * from Table3
...
```

Parentheses and evaluation order

By default, the Query of Queries SQL engine evaluates a statement containing UNION operators from left to right. You can use parentheses to change the order of evaluation. For example, the following two statements are different:

```
/* First statement. */
SELECT * FROM TableA
UNION ALL
(SELECT * FROM TableB
 UNION
 SELECT * FROM TableC
)

/* Second statement. */
(SELECT * FROM TableA
 UNION ALL
 SELECT * FROM TableB
)
UNION
SELECT * FROM TableC
```

In the first statement, there are no duplicates in the union between TableB and TableC. Then, in the union between that set and TableA, the ALL keyword includes the duplicates. In the second statement, duplicates are included in the union between TableA and TableB but are eliminated in the subsequent union with TableC. The ALL keyword has no effect on the final result of this expression.

Using other keywords with UNION

© 2014 Adobe Systems Incorporated. All rights reserved.

When you perform a UNION, the individual SELECT statements cannot have their own ORDER BY or COMPUTE clauses. You can only have one ORDER BY or COMPUTE clause after the last SELECT statement; this clause is applied to the final, combined result set. You can only specify GROUP BY and HAVING expressions in the individual SELECT statements.

Using conditional operators

Query of Queries lets you use the following conditional operators in your SQL statements:

Test conditional

This conditional tests whether a Boolean expression is True, False, or Unknown.

Syntax

```
cond_test ::= expression [IS [NOT] {TRUE | FALSE | UNKNOWN} ]
```

Example

```
SELECT _isValid FROM Chemicals
WHERE _isValid IS true;
```

Null conditional

This conditional tests whether an expression is null.

Syntax

```
null_cond ::= expression IS [NOT] NULL
```

Example

```
SELECT bloodVal FROM Standards
WHERE bloodVal IS NOT null;
```

Comparison conditional

This conditional lets you compare an expression against another expression of the same data type (Numeric, String, Date, or Boolean). You can use it to selectively retrieve only the relevant rows of a recordset.

Syntax

© 2014 Adobe Systems Incorporated. All rights reserved.

```
comparison_cond ::= expression [= | > | >= | <> | != | < | <=] expression
```

Example

The following example uses a comparison conditional to retrieve only those dogs whose IQ is at least 150:

```
SELECT dog_name, dog_IQ
FROM Dogs
WHERE dog_IQ >= 150;
```

BETWEEN conditional

This conditional lets you compare an expression against another expression. You can use it to selectively retrieve only the relevant rows of a recordset. Like the comparison conditional, the BETWEEN conditional also compares; however, the BETWEEN conditional compares against a range of values. Therefore, its syntax requires two values, which are inclusive, a minimum and a maximum. Separate these values with the AND keyword.

Syntax

```
between_cond ::= expression [NOT] BETWEEN expression AND expression
```

Example

The following example uses a BETWEEN conditional to retrieve only those dogs whose IQ is between 150 and 165, inclusive:

```
SELECT dog_name, dog_IQ
FROM Dogs
WHERE dog_IQ BETWEEN 150 AND 165;
```

IN conditional

This conditional lets you specify a comma-delimited list of conditions to match. It is similar in function to the OR conditional. In addition to being more legible when working with long lists, the IN conditional can contain another SELECT statement.

Syntax

```
in_cond ::= expression [NOT] IN (expression_list)
```

Example

© 2014 Adobe Systems Incorporated. All rights reserved.

The following example uses the IN conditional to retrieve only those dogs who were born at either Ken's Kennels or Barb's Breeders:

```
SELECT dog_name, dog_IQ, Kennel_ID
FROM Dogs
WHERE kennel_ID IN ('Kens','Barbs');
```

LIKE conditional

This conditional lets you perform wildcard searches, in which you compare your data to search patterns. This strategy differs from other conditionals, such as BETWEEN or IN, because the LIKE conditional compares your data to a value that is partially unknown.

Syntax

```
like_cond ::= left_string_exp [NOT] LIKE right_string_exp [ESCAPE escape_char]
```

The left_string_exp can be either a constant string, or a column reference to a string column. The right_string_exp can be either a column reference to a string column, or a search pattern. A *search pattern* is a search condition that consists of literal text and at least one wildcard character. A *wildcard character* is a special character that represents an unknown part of a search pattern, and is interpreted as follows:

- The underscore (_) represents any single character.
- The percent sign (%) represents zero or more characters.
- Brackets [ ] represents any character in the range.
- Brackets with a caret Query of Queries user guide represent any character not in the range.
- All other characters represent themselves.

> **Note**
>
> Earlier versions of ColdFusion do not support bracketed ranges.

Examples

The following example uses the LIKE conditional to retrieve only those dogs of the breed Terrier, whether the dog is a Boston Terrier, Jack Russell Terrier, Scottish Terrier, and so on:

```
SELECT dog_name, dog_IQ, breed
FROM Dogs
WHERE breed LIKE '%Terrier';
```

The following examples are select statements that use bracketed ranges:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
SELECT lname FROM Suspects WHERE lname LIKE 'A[^c]%';
SELECT lname FROM Suspects WHERE lname LIKE '[a-m]%';
SELECT lname FROM Suspects WHERE lname LIKE '%[]';
SELECT lname FROM Suspects WHERE lname LIKE 'A[%]%';
SELECT lname FROM Suspects WHERE lname LIKE 'A[^c-f]%';
```

Case sensitivity

Unlike the rest of ColdFusion, Query of Queries is case-sensitive. However, Query of Queries supports two string functions, `UPPER()` and `LOWER()`, which you can use to achieve case-insensitive matching.

Examples

The following example matches only 'Sylvester':

```
SELECT dog_name
FROM Dogs
WHERE dog_name LIKE 'Sylvester';
```

The following example is not case sensitive; it uses the `LOWER()` function to treat 'Sylvester', 'sylvester', 'SYLVESTER', and so on, as all lowercase, and matches them with the all lowercase string, sylvester:

```
SELECT dog_name
FROM Dogs
WHERE LOWER(dog_name) LIKE 'sylvester';
```

If you use a variable on the right side of the LIKE conditional and want to ensure that the comparison is not case-sensitive, use the `LCase` or `UCase` function to force the variable text to be all of one case, as in the following example:

```
WHERE LOWER(dog_name) LIKE '#LCase(FORM.SearchString)#';
```

Escaping wildcards

You can specify your own escape character by using the conditional ESCAPE clause.

Example

The following example uses the ESCAPE clause to enable a search for a literal percent sign (%), which ColdFusion normally interprets as a wildcard character:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
SELECT emp_discount
FROM Benefits
WHERE emp_discount LIKE '10\%'
ESCAPE '\';
```

Managing data types for columns

A Query of Queries requires that every column has metadata that defines the data type of the column. All queries that ColdFusion creates have metadata. However, a query created with `QueryNew` function that omits the second parameter does not contain metadata. You use this optional second parameter to define the data type of each column in the query.

Specify column data types in the QueryNew function

1. Type a `QueryNew`function, specifying the column names in the first parameter and the data types in the second parameter, as the following example shows:

```
<cfset qInstruments = queryNew("name, instrument, years_playing",
"CF_SQL_VARCHAR, CF_SQL_VARCHAR, CF_SQL_INTEGER")>
```

> **Note**
>
> To see the metadata for a Query of Queries, use the GetMetaData function.

Specify the column data types in the QueryAddColumn function

1. Create a query by specifying the QueryNew function with no parameters.

```
<!--- Make a query. --->
<cfset myQuery = QueryNew("")>
```

2. Add and populate a column with the QueryAddColumnfunction, specifying the data type in the third parameter:

```
<!--- Create an array. --->
<cfset FastFoodArray = ArrayNew(1)>
<cfset FastFoodArray[1] = "French Fries">
<cfset FastFoodArray[2] = "Hot Dogs">
<cfset FastFoodArray[3] = "Fried Clams">
<cfset FastFoodArray[4] = "Thick Shakes">
<!--- Use the array to add a column to the query. --->
<cfset nColumnNumber = QueryAddColumn(myQuery, "FastFood", "CF_SQL_VARCHAR",
 FastFoodArray)>
```

If you do not specify the data type, ColdFusion examines the first 50 rows of each column to determine the data type when performing conditional expressions.

© 2014 Adobe Systems Incorporated. All rights reserved.

In some cases, ColdFusion can guess a data type that is inappropriate for your application. In particular, if you use columns in a WHERE clause or other conditional expression, the data types must be compatible. If they are not compatible, use the CAST function to recast one of the columns to a compatible data type. For more information on casting, see *Using the CAST function* below. For more information on data type compatibility, see *Understanding Query of Queries processing* belowUnderstanding Query of Queries processingUnderstanding Query of Queries processing.

> **Note**
>
> Specifying the data type in the `QueryNew` function helps you avoid compatibility issues.

Using the CAST function

In some cases, the data type of a column is not compatible with the processing you want to do. For example, query columns returned by the `cfhttp` tag are all of type CF_SQL_VARCHAR, even if the contents are numeric. In this case, use the Query of Queries CAST function to convert a column value into an expression of the correct data type. The syntax for the CAST function is as follows:

```
CAST ( expression AS castType )
```

Where *castType* is one of the following:

- BINARY
- BIGINIT
- BIT
- DATE
- DECIMAL
- DOUBLE
- INTEGER
- TIME
- TIMESTAMP
- VARCHAR

  For example:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cfhttp
url="http://quote.yahoo.com/download/quotes.csv?Symbols=csco,jnpr&format=sc1l1&ext=.
csv"
 method="GET"
 name="qStockItems"
 columns="Symbol,Change,LastTradedPrice"
 textqualifier="""
 delimiter=","
 firstrowasheaders="no">
<cfoutput>
 <cfdump var="#qStockItems#">
 <cfdump var="#qStockItems.getColumnNames()#">
</cfoutput>

<cfoutput>
<cfloop index="i" from="1" to="#arrayLen(qStockItems.getColumnNames())#">
 #qStockItems.getMetaData().getColumnTypeName(javaCast("int",i))#<br/>
</cfloop>
</cfoutput>
<cftry>
 <cfquery name="hello" dbtype="query">
  SELECT SUM(CAST(qStockItems.LastTradedPrice as INTEGER))
  AS SUMNOW from qStockItems
 </cfquery>
 <cfcatch>Error in Query of Queries</cfcatch>
</cftry>

<cfoutput>
 <cfdump var="#hello#">
</cfoutput>
```

Using aggregate functions

Aggregate functions operate on a set of data and return a single value. Use these functions for retrieving summary information from a table, as opposed to retrieving an entire table and then operating on the recordset of the entire table.
Consider using aggregate functions to perform the following operations:

- To display the average of a column
- To count the number of rows for a column
- To find the earliest date in a column
  Since not every relational database management system (RDBMS) supports all aggregate functions, refer to the documentation of your database. The following table lists the aggregate functions that Query of Queries supports:

Function	Description
AVG()	Returns the average (mean) for a column.
COUNT()	Returns the number of rows in a column.
MAX()	Returns the largest value of a column.

© 2014 Adobe Systems Incorporated. All rights reserved.

MIN()	Returns the lowest value of a column.
SUM()	Returns the sum of values of a column.

Syntax

```
aggregate_func ::= <COUNT>(* | column_name) | AVG | SUM | MIN | MAX)
([ALL | DISTINCT] numeric_exp)
```

Example

The following example uses the `AVG( )` function to retrieve the average IQ of all terriers:

```
SELECT dog_name, AVG(dog_IQ) AS avg_IQ
FROM Dogs
WHERE breed LIKE '%Terrier';
```

Arbitrary expressions in aggregate functions

Query of Queries supports aggregate functions of any arbitrary expression, as follows:

```
SELECT lorange, count(lorange+hirange)
FROM roysched
GROUP BY lorange;
```

Aggregate functions in arbitrary expressions

Query of Queries supports mathematical expressions that include aggregate functions, as follows:

```
SELECT MIN(lorange) + MAX(hirange)
FROM roysched
GROUP BY lorange;
```

Using group by and having expressions

Query of Queries supports the use of any arbitrary arithmetic expression, as long as it is referenced by an alias.

Examples

The following code is correct:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
SELECT (lorange + hirange)/2 AS midrange,
COUNT(*)
FROM roysched
GROUP BY midrange;
```

The following code is correct:

```
SELECT (lorange+hirange)/2 AS x,
COUNT(*)
FROM roysched GROUP BY x
HAVING x > 10000;
```

The following code is not supported in Query of Queries:

```
SELECT (lorange + hirange)/2 AS midrange,
COUNT(*)
FROM roysched
GROUP BY (lorange + hirange)/2;
```

Using ORDER BY clauses

Query of Queries supports the ORDER BY clause to sort. Make sure that it is the last clause in your SELECT statement. You can sort by multiple columns, by relative column position, by nonselected columns. You can specify a descending sort direction with the DESC keyword (by default, most RDBMS sorts are ascending, which makes the ASC keyword unnecessary).

Syntax

```
order_by_column ::= ( <IDENTIFIER> | <INTEGER_LITERAL> ) [<ASC> | <DESC>]
```

Example

The following example shows a simple sort using an ORDER BY clause:

```
SELECT acetylcholine_levels, dopamine_levels
FROM results
ORDER BY dopamine_levels
```

The following example shows a more complex sort; results are first sorted by ascending levels of dopamine, then by descending levels of acetylcholine. The ASC keyword is unnecessary, and is used only for legibility.

© 2014 Adobe Systems Incorporated. All rights reserved.

```
SELECT acetylcholine_levels, dopamine_levels
FROM results
ORDER BY 2 ASC, 1 DESC
```

Using aliases

Query of Queries supports the use of database column aliases. An *alias* is an alternate name for a database field or value. Query of Queries lets you reuse an alias in the same SQL statement.
One way to create an alias is to concatenate (append) two or more columns to generate a value. For example, you can concatenate a first name and a last name to create the value fullname. Because the new value does not exist in a database, you refer to it by its alias. The AS keyword assigns the alias in the SELECT statement.

Examples

Query of Queries supports alias substitutions in the ORDER BY, GROUP BY, and HAVING clauses.

> **Note**
>
> Query of Queries does not support aliases for table names.

```
SELECT FirstName + ' ' + LastName AS fullname
from Employee;
```

The following examples rely on these two master queries:

```
<cfquery name="employee" datasource="2pubs">
 SELECT * FROM employee
</cfquery>

<cfquery name="roysched" datasource="2pubs">
 SELECT * FROM roysched
</cfquery>
```

ORDER BY example

```
<cfquery name="order_by" dbtype="query">
 SELECT (job_id || job_lvl)/2 AS job_value
 FROM employee
 ORDER BY job_value
</cfquery>
```

GROUP BY example

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cfquery name="group_by" dbtype="query">
 SELECT lorange || hirange AS x, count(hirange)
 FROM roysched
 GROUP BY x
</cfquery>
```

HAVING example

```
<cfquery name="having" dbtype="query">
 SELECT (lorange || hirange)/2 AS x,
 COUNT(*)
 FROM roysched GROUP BY x
 HAVING x > 10000
</cfquery>
```

Handling null values

Query of Queries uses Boolean logic to handle conditional expressions. Proper handling of NULL values requires the use of ternary logic. The `IS NOT NULL` clause works correctly in Query of Queries. However the following expressions do not work properly when the column breed is NULL:

```
WHERE (breed > 'A')
WHERE NOT (breed > 'A')
```

The correct behavior should not include NULL breed columns in the result set of either expression. To avoid this limitation, add an explicit rule to the conditionals and rewrite them in the following forms:

```
WHERE breed IS NOT NULL AND (breed > 'A')
WHERE breed IS NOT NULL AND not (breed > 'A')
```

Concatenating strings

Query of Queries support two string concatenation operators: + and ||, as the following examples show:

```
LASTNAME + ', ' + FIRSTNAME
LASTNAME || ', ' || FIRSTNAME
```

Escaping reserved keywords

ColdFusion has a list of reserved keywords, which are typically part of the SQL language and are not normally used for names of columns or tables. To escape a reserved keyword for a column name or table name, enclose it in brackets.

© 2014 Adobe Systems Incorporated. All rights reserved.

Earlier versions of ColdFusion let you use some reserved keywords without escaping them.

Examples

Query of Queries supports the following SELECT statement examples:

```
SELECT [from] FROM parts;
SELECT [group].firstname FROM [group];
SELECT [group].[from] FROM [group];
```

Query of Queries does not support nested escapes, such as in the following example:

```
SELECT [[from]] FROM T;
```

The following table lists ColdFusion reserved keywords:{{}}

ABSOLUTE	ACTION	ADD	ALL	ALLOCATE
ALTER	AND	ANY	ARE	AS
ASC	ASSERTION	AT	AUTHORIZATION	AVG
BEGIN	BETWEEN	BIT	BIT_LENGTH	BOTH
BY	CASCADE	CASCADED	CASE	CAST
CATALOG	CHAR	CHARACTER	CHARACTER_LENGTH	CHAR_LENGTH
CHECK	CLOSE	COALESCE	COLLATE	COLLATION
COLUMN	COMMIT	CONNECT	CONNECTION	CONSTRAINT
CONSTRAINTS	CONTINUE	CONVERT	CORRESPONDING	COUNT
CREATE	CROSS	CURRENT	CURRENT_DATE	CURRENT_TIME
CURRENT_TIMESTAMP	CURRENT_USER	CURSOR	DATE	DAY
DEALLOCATE	DEC	DECIMAL	DECLARE	DEFAULT
DEFERRABLE	DEFERRED	DELETE	DESC	DESCRIBE

© 2014 Adobe Systems Incorporated. All rights reserved.

DESCRIPTOR	DIAGNOSTICS	DISCONNECT	DISTINCT	DOMAIN
DOUBLE	DROP	ELSE	END	END-EXEC
ESCAPE	EXCEPT	EXCEPTION	EXEC	EXECUTE
EXISTS	EXTERNAL	EXTRACT	FALSE	FETCH
FIRST	FLOAT	FOR	FOREIGN	FOUND
FROM	FULL	GET	GLOBAL	GO
GOTO	GRANT	GROUP	HAVING	HOUR
IDENTITY	IMMEDIATE	IN	INDICATOR	INITIALLY
INNER	INPUT	INSENSITIVE	INSERT	INT
INTEGER	INTERSECT	INTERVAL	INTO	IS
ISOLATION	JOIN	KEY	LANGUAGE	LAST
LEADING	LEFT	LEVEL	LIKE	LOCAL
LOWER	MATCH	MAX	MIN	MINUTE
MODULE	MONTH	NAMES	NATIONAL	NATURAL
NCHAR	NEXT	NO	NOT	NULL
NULLIF	NUMERIC	OCTET_LENGTH	OF	ON
ONLY	OPEN	OPTION	OR	ORDER
OUTER	OUTPUT	OVERLAPS	PAD	PARTIAL
POSITION	PRECISION	PREPARE	PRESERVE	PRIMARY
PRIOR	PRIVILEGES	PROCEDURE	PUBLIC	READ
REAL	REFERENCES	RELATIVE	RESTRICT	REVOKE
RIGHT	ROLLBACK	ROWS	SCHEMA	SCROLL
SECOND	SECTION	SELECT	SESSION	SESSION_USER
SET	SMALLINT	SOME	SPACE	

© 2014 Adobe Systems Incorporated. All rights reserved.

SQL	SQLCODE	SQLERROR	SQLSTATE	SUBSTRING
SUM	SYSTEM_USER	TABLE	TEMPORARY	THEN
TIME	TIMESTAMP	TIMEZONE_HOUR	TIMEZONE_MINUTE	TO
TRAILING	TRANSACTION	TRANSLATE	TRANSLATION	TRIM
TRUE	UNION	UNIQUE	UNKNOWN	UPDATE
UPPER	USAGE	USER	USING	VALUE
VALUES	VARCHAR	VARYING	VIEW	WHEN
WHENEVER	WHERE	WITH	WORK	WRITE
YEAR	ZONE			

Using Queries of Queries with dates

If you create a query object with the `QueryNew` function and populate a column with date constants, ColdFusion stores the dates as a string inside the query object until a Query of Queries is applied to the query object. When ColdFusion applies a Query of Queries to the query object, it converts the string representations into date objects. Query of Queries supports date constants in SQL and ODBC format, as follows:

- **SQL format**: Dates, times, or timestamps in one of the following format:
  - **Date string**: *yyyy-mm-dd*, for example, 1955-06-13.
  - **Time string**: *hh:mm:ss[.nnn]*, for example, 14:34:30.75.
  - **Timestamp string**: *yyyy-mm-dd hh:mm:ss[.nnn]*, for example, 1924-01-14 12:00:00.000.
- **ODBC format**: Dates, times, or timestamps in one of the following format:
  - **Date string**: *{d 'value'}*,
    for example, {d '2004-07-06'}.
  - **Time string**: *{t 'value'}*, for example, {t '13:45:30'}.
  - **Timestamp string**: *{ts 'value'}*, for example,
    {ts '2004-07-06 13:45:30'}.

> ✅ If you want to convert the date to its original format, use the `DateFormat_` function and apply the "mm/dd/yy" mask._

Understanding Query of Queries performance

Query of Queries performs well on single-table query objects that were accessed directly from a database. This happens because ColdFusion stores meta information for a query object accessed from a database.
When working with a query resulting in a SQL join, Query of Queries performs as follows:

1. Query of Queries is efficient for simple joins in which only one equality exists between two column references or constants, for example:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
SELECT T1.a, b, c, d FROM T1, T2 WHERE T1.a = T2.a
```

2. Query of Queries is less efficient for joins in which the predicate contains multiple expressions, for example:

```
SELECT T1.a, b, c, d FROM T1, T2
WHERE T1.a = T2.a AND T1.b + T1.c = T2.b + T2.c
```

Understanding Query of Queries processing

Query of Queries can process the following:

- Column comparisons
- Queries passed by reference
- Complex objects

Comparing columns with different data types

Starting with ColdFusion MX 7, ColdFusion includes enhancements that allow you to compare columns with different data types.
If one of the operands has a known column type (only constants have an unknown column type), Query of Queries tries to coerce the constant with an unknown type to the type of the operand with metadata. The pairs of allowed coercions are as follows:

- Binary, string
- Dates, string
- Numeric, bigdecimal
- Boolean, numeric
  That is, ColdFusion can coerce between binary and string, but not between date and string.
  If both operands have known data types, the types must be the same. The only exception is that ColdFusion can coerce among integer, float, and double.
  If both operands are constants, ColdFusion tries to coerce the values, first to the most restrictive type, then to the least restrictive type.

- First to binary then to string.
- First to date then to string.
- First to boolean then to numeric.

Passing queries by reference

A Query of Queries is copied by reference from its related query; which means that ColdFusion does not create a query when you create a Query of Queries. It also means that changes to a Query of Queries, such as ordering, modifying, and deleting data, are also applied to the base query object.
If you do not want the original query to change, use the Duplicate function to create a copy and create the Query of Queries using the copied query.

Managing complex objects

You cannot use Query Of Queries on a recordset that contains complex objects, such as arrays and structures.

© 2014 Adobe Systems Incorporated. All rights reserved.

Note

You can store a recordset in a complex object.

#back to top

© 2014 Adobe Systems Incorporated. All rights reserved.

Managing LDAP Directories

CFML applications use the `cfldap` tag to access and manage LDAP (Lightweight Directory Access Protocol) directories.

The following information teaches you to query and update an LDAP database. It is not assumed that you are familiar with LDAP, and hence an introduction to LDAP directories and the LDAP protocol is provided. However, it is assumed that you have information on the structure and attributes of your LDAP database. Hence, procedures to create an LDAP directory or manage a directory server are not provided. To learn more about LDAP and LDAP servers, see your LDAP server documentation and published books on LDAP.

The examples here use the Airius sample LDAP database that is supplied with the Netscape and iPlanet Directory Servers.

© 2014 Adobe Systems Incorporated. All rights reserved.

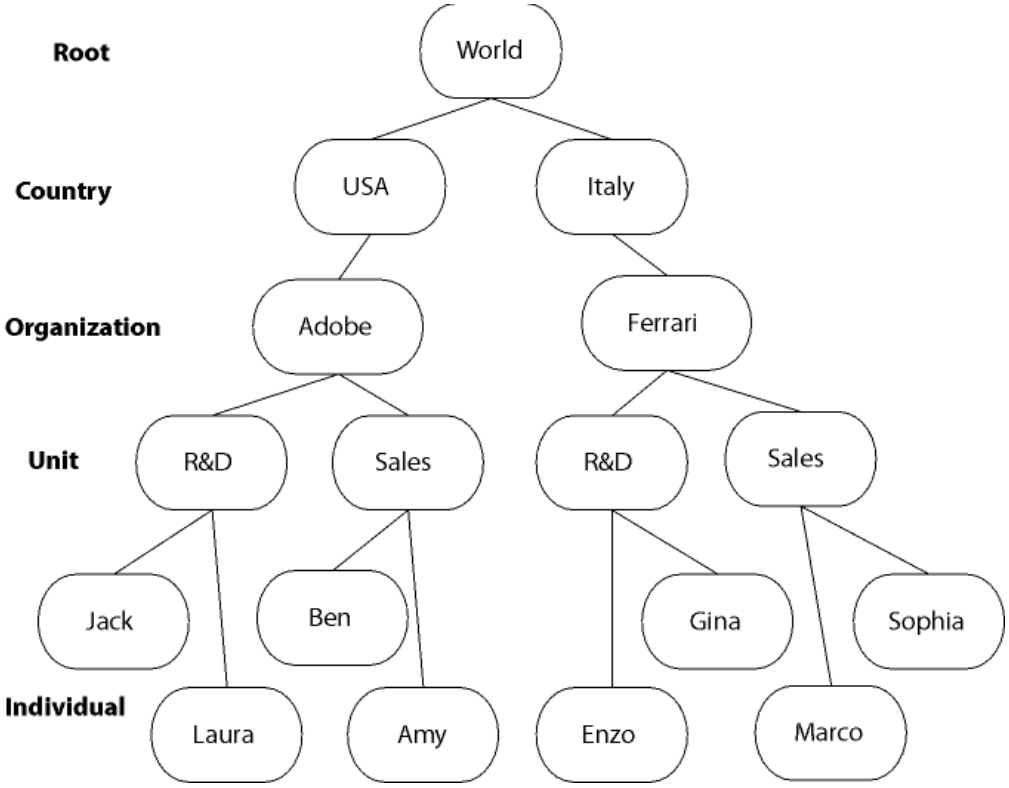
About LDAP

The LDAP protocol enables organizations to arrange and access directory information in a hierarchy. In this context, *directory* refers to a collection of information, such as a telephone directory, not a collection of files in a folder on a disk drive.

LDAP originated in the mid-1990s as a response to the need to access ISO X.500 directories from computers that had limited processing power. Since then, products such as iPlanet Server have been developed that are native LDAP directory servers. Several companies now provide LDAP access to their directory servers, including Novell NDS, Microsoft Active Directory Services (ADS), Lotus Domino, and Oracle.

An LDAP directory is typically a hierarchically structured database. Each layer in the hierarchy typically corresponds to a level of organizational structure.

The following image shows a simple directory structure:

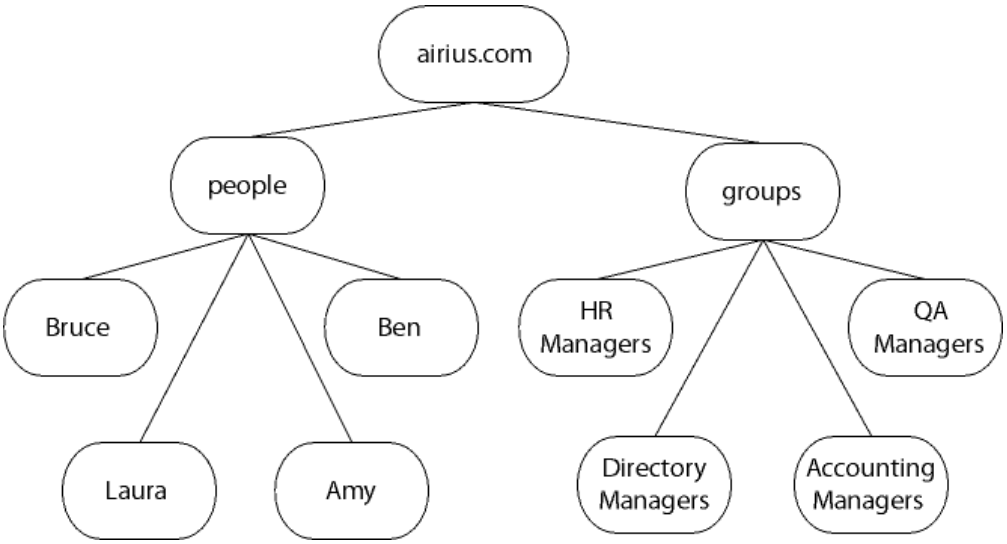


This example is fully symmetrical: all the entries at each layer are of the same type.

You can also structure an LDAP directory so that the layers under one entry contain different information from the layers under another entry.

The following image shows such an asymmetric directory:

© 2014 Adobe Systems Incorporated. All rights reserved.



In this directory structure, the second level of the tree divides the directory into two organizational units: people and groups. The third level contains entries with information that is specific to the organizational unit. Each person's entry includes a name, e-mail address, and telephone number. Each group's entry includes the names of group members.

This complexity and flexibility is a key to the usefulness. of LDAP. With it, you can represent any organizational structure.

LDAP offers performance advantages over conventional databases for accessing hierarchical, directory-like information that is read frequently, and changed infrequently.

Although LDAP is often used for e-mail, address, telephone, or other organizational directories, it is not limited to these types of applications. For example, you can store ColdFusion Advanced Security information in an LDAP database.

© 2014 Adobe Systems Incorporated. All rights reserved.

The LDAP information structure

Several LDAP concepts are the basis of the LDAP information structure:

- Entry
- Attribute
- Distinguished name (DN)
- Schema, including the object class and attribute type

Entry

The basic information object of LDAP is the *entry*. An entry is composed of one or more *attributes*. Entries are subject to content rules defined by the directory *schema* (see *Schema* below).
Each node, not just the terminal nodes, of an LDAP directory is an entry. In the preceding images, each item is an entry. For example, in the first diagram, both USA and Ferrari are entries. The USA entry's attributes could include a Language attribute, and the Ferrari entry could include an entry for the chief executive officer.

Attribute

An LDAP directory entry consists of one or more attributes. Attributes have *types* and *values*. The type determines the information that the values can contain. The type also specifies how the value is processed. For example, the type determines whether an attribute can have multiple values. The mail attribute type, which contains an e-mail address, is multivalued so you can store multiple e-mail addresses for one person.
Some commonly used attribute types have short keyword type names. Often these short keyword type names correspond to longer type names, and the two names can be used interchangeably. The following table lists common attribute type keywords used in LDAP directories:

Keyword	Long name	Comment
c	CountryName	
st	stateOrProvinceName	
l	LocalityName	Typically, city, but can be any geographical unit
street	StreetAddress	
o	OrganizationName	
ou	OrganizationalUnitName	
cn	CommonName	Typically, first and last name
sn	SurName	
dc	domaincomponent	
mail	mail	E-mail address

For more information, see *Attribute type* below.

© 2014 Adobe Systems Incorporated. All rights reserved.

Distinguished name (DN)

An entry's *distinguished name* uniquely identifies it in the directory. A DN is made up of *relative distinguished names* (RDNs). An RDN identifies the entry among the children of its parent entry. For example, in the first image in About LDAP, the RDN for the Ferrari entry is "o=Ferrari".
An entry's DN consists of an entry's RDN followed by the DN of its parent. In other words, it consists of the RDNs for the entry and each of the entry's parent entries, up to the root of the directory tree. Commas and optional spaces separate the RDNs. For example, in the first image, the DN for the Ferrari entry is "o=Ferrari, c=Italy".
As with file system paths and URLs, entering the correct LDAP name format is essential to successful search operations.

> **Note**
>
> The RDN is an attribute of a directory entry. The full DN is not. However, you can output the full DN by specifying "dn" in a query's `attributes` list. For more information, see `cfldap` in CFML Reference. Adobe ColdFusion always returns DNs with spaces after the commas.

A *multivalued RDN* is made up of more than one attribute-value pair. In multivalued RDNs, a plus sign separates the attribute-value pairs. In the sample directories, individuals could have complex RDNs consisting of their common name and their e-mail address, for example, "cn=Robert Boyd + mail=rjboyd@adobe.com".

Schema

The concepts of schemas and object classes are central to a thorough understanding of LDAP. The information provided here does not have detailed descriptions but is enough to use the cfldap tag effectively.
A directory *schema* is a set of rules that determines what can be stored in a directory. It defines, at a minimum, the following two basic directory characteristics:

- The object classes to which entries can belong
- The directory attribute types

Object class

Object classes enable LDAP to group related information. Frequently, an object class corresponds to a real object or concept, such as a country, person, room, or domain, which are all standard object type names. Each entry in an LDAP directory must belong to one or more object classes.
The following characteristics define an object class:

- The class name
- A unique object ID that identifies the class
- The attribute types that entries of the class must contain
- The attribute types that entries of the class can optionally contain
- (Optional) A *superior* class from which the class is derived
  If an entry belongs to a class that derives from another class, the entry's objectclass attribute lists the lowest-level class and all the superior classes from which the lowest-level class derives.
  When you add, modify, or delete a directory entry, you must treat the entry's object class as a possibly multivalued attribute. For example, when you add a new entry, you specify the object class in the cfldap tag a `ttributes` attribute. To retrieve an entry's object class names, specify "objectclass" in the list of query attributes. To retrieve entries that provide a specific type of information, you can use the object class name in the `cfldap` tag `filter` attribute.

Attribute type

The attribute type specification of a schema defines the following properties:

© 2014 Adobe Systems Incorporated. All rights reserved.

- The attribute type name
- A unique object ID that identifies the attribute type
- (Optional) An indication of whether the type is single-valued or multivalued (the default is multivalued)
- The attribute syntax and matching rules (such as case sensitivity)
  The attribute type definition can also determine limits on the range or size of values that the type represents, or provide an application-specific usage indicator. For standard attributes, a registered numeric ID specifies the syntax and matching rule information. For more information on attribute syntaxes, see ETF RFC 2252 at http://www.ietf.org/rfc/rfc2252.txt.
  Operational attributes, such as creatorsName or modifyTimeStamp, are managed by the directory service and cannot be changed by user applications.

© 2014 Adobe Systems Incorporated. All rights reserved.

Using LDAP with ColdFusion

The cfldap tag extends the ColdFusion query capabilities to LDAP network directory services. The `cfldap` tag lets you use LDAP in many ways, such as the following:

- Create Internet White Pages so users can locate people and resources and get information about them.
- Provide a front end to manage and update directory entries.
- Build applications that incorporate data from directory queries in their processes.
- Integrate applications with existing organizational or corporate directory services.
  The `cfldap` tag `action` attribute supports the following operations on LDAP directories:

Action	Description
`query`	Returns attribute values from a directory.
`add`	Adds an entry to a directory.
`delete`	Deletes an entry from a directory.
`modify`	Adds, deletes, or changes the value of an attribute in a directory entry.
`modifyDN`	Renames a directory entry (changes its distinguished name).

The following table lists the attributes that are required and optional for each action. For more information on each attribute, see the cfldap tag in the *CFML Reference*.

Action	Required attributes	Optional attributes
`query`	`server`, `name`, `start`, `attributes`	`port`, `username`, `password`, `timeout`, `secure`, `rebind`, `referral`, `scope`, `filter`, `sort`, `sortControl`, `startRow`, `maxRows`, `separator`, `delimiter`
`add`	`server`, `dn`, `attributes`	`port`, `username`, `password`, `timeout`, `secure`, `rebind`, `referral`, `separator`, `delimiter`
`delete`	`server`, `dn`	`port`, `username`, `password`, `timeout`, `secure`, `rebind`, `referral`
`modify`	`server`, `dn`, `attributes`	`port`, `username`, `password`, `timeout`, `secure`, `rebind`, `referral`, `modifyType`, `separator`, `delimiter`

© 2014 Adobe Systems Incorporated. All rights reserved.

modifyDN	server, dn, attributes	port, username, password, timeout, secure, rebind, referral

© 2014 Adobe Systems Incorporated. All rights reserved.

Querying an LDAP directory

The cfldap tag lets you search an LDAP directory. The tag returns a ColdFusion query object with the results, which you can use as you would any query result. When you query an LDAP directory, you specify the directory entry where the search starts and the attributes whose values to return. You can specify the search scope and attribute content filtering rules and use other attributes to further control the search.

Scope

The search *scope* sets the limits of a search. The default scope is the level below the distinguished name specified in the `start` attribute. This scope does not include the entry identified by the `start` attribute. For example, if the `start` attribute is "ou=support, o=adobe" the level below support is searched. You can restrict a query to the level of the `start` entry, or extend it to the entire subtree below the `start` entry.

Search filter

The search filter syntax has the form *attribute operator value.* The default filter, objectclass=*, returns all entries in the scope.
The following table lists the filter operators:

Operator	Example	Matches
=*	(mail=*)	All entries that contain a mail attribute.
=	(o=adobe)	Entries in which the organization name is adobe.
~=	(sn~=Hansen)	Entries with a surname that approximates Hansen. The matching rules for approximate matches vary among directory vendors, but anything that "sounds like" the search string should be matched. In this example, the directory server might return entries with the surnames Hansen and Hanson.
>=	(st>=ma)	The name "ma" and names appearing after "ma" in an alphabetical state attribute list.
<=	(st<=ma)	The name "ma" and names appearing before "ma" in an alphabetical state attribute list.
*	(o=macro*)	Organization names that start with "macro".

© 2014 Adobe Systems Incorporated. All rights reserved.

	(o=*media)	Organization names that end with "media".
	(o=mac*ia)	Organization names that start with "mac" and end with "ia". You can use more than one * operator in a string; for example, m*ro*dia.
	(o=*med*)	Organization names that contain the string "med", including the exact string match "med".
&	(&(o=adobe)(co=usa))	Entries in which the organization name is "adobe" and the country is "usa".
|	(|(o=adobe)(sn=adobe)(cn=adobe))	Entries in which the organization name is "adobe" or the surname is "adobe", or the common name is "adobe".
!	(!(STREET=*))	Entries that do not contain a StreetAddress attribute.

The Boolean operators & and | can operate on more than two attributes and precede all of the attributes on which they operate. You surround a filter with parentheses and use parentheses to group conditions.
If the pattern that you are matching contains an asterisk, left parenthesis, right parenthesis, backslash, or NUL character, use the following three-character escape sequence in place of the character:

Character	Escape sequence
*	\2A
(	\28
)	\29
|\5C	
NUL	\00

For example, to match the common name St*r Industries, use the filter:
(cn=St\2Ar Industries).
LDAP v3 supports an extensible match filter that permits server-specific matching rules. For more information on using extensible match filters, see your LDAP server documentation.

Searching and sorting notes

- To search for multiple values of a multivalued attribute type, use the & operator to combine expressions for each attribute value. For example, to search for an entry in which cn=Robert Jones and cn=Bobby Jones,

© 2014 Adobe Systems Incorporated. All rights reserved.

specify the following filter:

```
filter="(&(cn=Robert Jones)(cn=Bobby Jones))"
```

- You can use object classes as search filter attributes; for example, you can use the following search filter:

```
filter="(objectclass=inetorgperson)"
```

- To specify how query results are sorted, use the `sort` field to identify the attribute(s) to sort. By default, ColdFusion returns sorted results in case-sensitive ascending order. To specify descending order, case-insensitive sorting, or both, use the `sortControl` attribute.
- ColdFusion requests the LDAP server to do the sorting. Doing so can have the following effects:
  - The sort order can differ between ColdFusion MX and previous versions.
  - If you specify sorting and the LDAP server does not support sorting, ColdFusion generates an error. To sort results from servers that do not support sorting, use a query of queries on the results.
- Using filter operators to construct sophisticated search criteria can degrade performance if the LDAP server is slow to process the synchronous search routines that `cfldap` supports. Use the `cfldap` tag `timeout` and `maxRows` attributes to control the apparent performance of pages that perform queries, by limiting the number of entries and by exiting the query if the server does not respond in a specified time.

Getting all the attributes of an entry

Typically, you do not use a query that gets all the attributes in an entry. Such a query would return attributes that only the directory server uses. However, you can get all the attributes by specifying attributes="*" in your query. If you do this, ColdFusion returns the results in a structure in which each element contains a single attribute name-value pair. The tag does *not* return a query object. ColdFusion does this because LDAP directory entries, unlike the rows in a relational table, vary depending on their object class.
For example, the following code retrieves the contents of the Airius directory:

```
<cfldap name="GetList"
 server=#myServer#
 action="query"
 attributes="*"
 scope="subtree"
 start="o=airius.com"
 sort="sn,cn">
```

This tag returns entries for all the people in the organization and entries for all the groups. The group entries have a different object class, and therefore different attributes from the person entries. If ColdFusion returned both types of entries in one query object, some rows would have only the group-specific attribute values and the other rows would have only person-specific attribute values. Instead, ColdFusion returns a structure in which each attribute is an entry.

Example: querying an LDAP directory

The following example uses the `cfldap` tag to get information about the people in the Airius corporation's Santa Clara office. Users can enter all or part of a person's name and get a list of matching names with their departments, e-mail addresses, and telephone numbers.

© 2014 Adobe Systems Incorporated. All rights reserved.

This example uses the sample Airius corporate directory that is distributed with the Netscape Directory Server. If you do not have access to this directory, modify the code to work with your LDAP directory.

1. Create a file that looks like the following:

```
<!--- This example shows the use of CFLDAP --->
<html>
<head> <title>cfldap Query Example</title> </head>

<h3>cfldap Query Example</h3>

<body>
<p>This tool queries the Airius.com database to locate all people in the
company's
 Santa Clara office whose common names contain the text entered in the
form.</p>

<p>Enter a full name, first name, last name, or name fragment.</p>

<form action="cfldap.cfm" method="POST">
 <input type="text" name="name"><br><br>
 <input type="submit" value="Search">
</form>
<!--- make the LDAP query --->
<!-- Note that some search text is required.
 A search filter of cn=** would cause an error -->
<cfif (isdefined("form.name") AND (form.name IS NOT ""))>
 <cfldap
  server="ldap.airius.com"
  action="query"
  name="results"
  start="ou=People, o=Airius.com"
  scope="onelevel"
  filter="(&(cn=*#form.Name#*)(l=Santa Clara))"
  attributes="cn,sn,ou,mail,telephonenumber"
  sort="ou,sn"
  maxrows=100
  timeout=20000
 >

<!--- Display results --->
 <table border=0 cellspacing=2 cellpadding=2>
  <tr>
  <th colspan=4><cfoutput>#results.
  RecordCount# matches found</cfoutput>
  </th>
  </tr>
  <tr>
  <th>Name</th>
  <th>Department</th>
  <th>E-Mail</th>
  <th>Phone</th>
  </tr>
 <cfoutput query="results">
  <tr>
  <td>#cn#</td>
  <td>#listFirst(ou)#</td>
  <td><a href="mailto:#mail#">#mail#</a></td>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

```
    <td>#telephonenumber#</td>
   </tr>
   </cfoutput>
   </table>
  </cfif>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

```
</body>
</html>
```

2. Change the `server` attribute from ldap.airius.com to the name of your installation of the Airius database.
3. Save the page as cfldap.cfm and run it in your browser.

Reviewing the code

The following table describes the code:

Code	Description
`<form action="cfldap.cfm" method="POST">` `  <input type="text" name="name">  ` `  <input type="submit" value="Search">` `</form>`	Uses a form to get the name or name fragment to search for.
`<cfif (isdefined("form.name") AND (form.name IS NOT ""))>`	Ensures that the user has submitted the form. This is necessary because the form page is also the action page. Ensures that the user entered search text.

`<cfldap`
`  server="ldap.airius.com"`
`  action="query"`
`  name="results"`
`  start="ou=People, o=Airius.com"`
 	
`  scope="onelevel"`
 	
`filter="(&(cn=*#form.Name#*)(l=Santa Clara))"`
 	
`attributes="cn,sn,ou,mail,telephonenumber"`
`  sort="ou,sn"`
`  maxrows=100`
`  timeout=20000`
`  >`
	Connects anonymously to LDAP server ldap.airius.com, query the directory, and return the results to a query object named results. Starts the query at the directory entry that has the distinguished name ou=People, o=Airius.com, and searches the directory level immediately below this entry.Requests records for entries that contain the location (l) attribute value "Santa Clara" and the entered text in the common name attribute.Gets the common name, surname, organizational unit, e-mail address, and telephone number for each entry.Sorts the results first by organization name, then by surname. Sorts in the default sorting order.Limit the request to 100 entries. If the server does not return the data in 20 seconds, generates an error indicating that LDAP timed out.

© 2014 Adobe Systems Incorporated. All rights reserved.

`<table border=0 cellspacing=2 cellpadding=2>`
`<tr>`
`<th colspan=4><cfoutput>#results.RecordCount# matches found</cfoutput>`
`</th>`
`</tr>`
`<tr>`
`<th>Name</th>`
`<th>Department</th>`
`<th>E-Mail</th>`
`<th>Phone</th>`
`</tr>`
`<cfoutput query="results">`
`<tr>`
`<td>#cn#</td>`
`<td>#listFirst(ou)#</td>`
`<td><a href="mailto:#mail#">#mail#</a></td>`
`<td>#telephonenumber#</td>`
`</tr>`
`</cfoutput>`
`</table>`
`</cfif>`
	Starts a table to display the output. Displays the number of records returned. Displays the common name, department, e-mail address, and telephone number of each entry.Displays only the first entry in the list of organizational unit values. (For more information, see the description that follows this table.)

This search shows the use of a logical AND statement in a filter. It returns one attribute, the surname, that is used only for sorting the results.In this query, the `ou` attribute value consists of two values in a comma-delimited list. One is the department name. The other is People. This is because the Airius database uses the `ou` attribute type twice:

- In the distinguished names, at the second level of the directory tree, where it differentiates between organizational units such as people, groups, and directory servers
- As the department identifier in each person's entry
  Because the attribute values are returned in order from the person entry to the directory tree root, the `ListFirst` function extracts the person's department name.

© 2014 Adobe Systems Incorporated. All rights reserved.

Updating an LDAP directory

The `cfldap` tag lets you perform the following actions on LDAP directory entries:

- Add
- Delete
- Add attributes
- Delete attributes
- Replace attributes
- Change the DN (rename the entry)
  These actions let you manage LDAP directory contents remotely.
  You build a ColdFusion page that lets you manage an LDAP directory. The form displays directory entries in a table and includes a button that lets you populate the form fields based on the unique user ID.
  The example ColdFusion page does not add or delete entry attributes or change the DN. For information on these operations, see *Adding and deleting attributes of a directory entry* below and *Changing a directory entry's DN* below.
  To keep the code short, this example has limitations that are not appropriate in a production application. In particular, it has the following limitations:

- If you enter an invalid user ID and click either the Update or the Delete button, ColdFusion generates a "No such object" error, because there is no directory entry to update or delete. Your application should verify that the ID exists in the directory before it tries to change or delete its entry.
- If you enter a valid user ID in an empty form and click Update, the application deletes all the attributes for the User. The application should ensure that all required attribute fields contain valid entries before updating the directory.

Adding a directory entry

When you add an entry to an LDAP directory, you specify the DN, all the required attributes, including the entry's object class, and any optional attributes. The following example builds a form that adds an entry to an LDAP directory.

1. Create a file that looks like the following:

```
<!--- Set the LDAP server ID, user name, and password as variables
 here so they can be changed in only one place. --->
<cfset myServer="ldap.myco.com">
<cfset myUserName="cn=Directory Manager">
<cfset myPassword="password">

<!--- Initialize the values used in form fields to empty strings. --->
<cfparam name="fullNameValue" default="">
<cfparam name="surnameValue" default="">
<cfparam name="emailValue" default="">
<cfparam name="phoneValue" default="">
<cfparam name="uidValue" default="">

<!---When the form is submitted, add the LDAP entry. --->
<cfif isdefined("Form.action") AND Trim(Form.uid) IS NOT "">
 <cfif Form.action is "add">
  <cfif Trim(Form.fullName) is "" OR Trim(Form.surname) is ""
   OR Trim(Form.email) is "" OR Trim(Form.phone) is "">
    <h2>You must enter a value in every field.</h2>
    <cfset fullNameValue=Form.fullName>
    <cfset surnameValue=Form.surname>
    <cfset emailValue=Form.email>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

```
    <cfset phoneValue=Form.phone>
    <cfset uidValue=Form.uid>
  <cfelse>
   <cfset attributelist="objectclass=top, person,
     organizationalperson, inetOrgPerson;
    cn=#Trim(Form.fullName)#; sn=#Trim(Form.surname)#;
    mail=#Trim(Form.email)#;
    telephonenumber=#Trim(Form.phone)#;
    ou=Human Resources;
    uid=#Trim(Form.uid)#">
   <cfldap action="add"
    attributes="#attributeList#"
    dn="uid=#Trim(Form.uid)#, ou=People, o=Airius.com"
    server=#myServer#
    username=#myUserName#
    password=#myPassword#>
   <cfoutput><h3>Entry for User ID #Form.uid# has been added</h3>
   </cfoutput>
  </cfif>
 </cfif>
</cfif>

<html>
<head>
 <title>Update LDAP Form</title>
</head>
<body>
<h2>Manage LDAP Entries</h2>

<cfform action="update_ldap.cfm" method="post">
 <table>
  <tr><td>Full Name:</td>
   <td><cfinput type="Text"
     name="fullName"
     value=#fullNameValue#
     size="20"
     maxlength="30"
     tabindex="1"></td>
  </tr>
  <tr><td>Surname:</td>
   <td><cfinput type="Text"
    name="surname"
    Value= "#surnameValue#"
    size="20"
    maxlength="20"
    tabindex="2"></td>
  </tr>
  <tr>
   <td>E-mail Address:</td>
   <td><cfinput type="Text"
    name="email"
    value="#emailValue#"
    size="20"
    maxlength="20"
    tabindex="3"></td>
  </tr>
  <tr>
   <td>Telephone Number:</td>
   <td><cfinput type="Text"
```

© 2014 Adobe Systems Incorporated. All rights reserved.

```
     name="phone"
     value="#phoneValue#"
     size="20"
     maxlength="20"
     tabindex="4"></td>
   </tr>
   <tr>
    <td>User ID:</td>
    <td><cfinput type="Text"
     name="uid"
     value="#uidValue#"
     size="20"
     maxlength="20"
     tabindex="5"></td>
   </tr>
   <tr>
    <td colspan="2">
     <input type="Submit"
      name="action"
      value="Add"
      tabindex="8"></td>
   </tr>
  </table>
  <br>
  *All fields are required for Add<br>
 </cfform>

 <!---Output the user list. --->
 <h2>User List for the Human Resources Department</h2>
 <cfldap name="GetList"
  server=#myServer#
  action="query"
  attributes="cn,sn,mail,telephonenumber,uid"
  start="o=Airius.com"
  scope="subtree"
  filter="ou=Human Resources"
  sort="sn,cn"
  sortControl="asc, nocase">

 <table border="1">
  <tr>
   <th>Full Name</th>
   <th>Surname</th>
   <th>Mail</th>
   <th>Phone</th>
   <th>UID</th>
  </tr>
  <cfoutput query="GetList">
  <tr>
   <td>#GetList.cn#</td>
   <td>#GetList.sn#</td>
   <td>#GetList.mail#</td>
   <td>#GetList.telephonenumber#</td>
   <td>#GetList.uid#</td>
  </tr>
  </cfoutput>
 </table>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

```
</body>
</html>
```

2. At the top of the file, change the `myServer`, `myUserName`, and `myPassword` variable assignments to values that are valid for your LDAP server.
3. Save the page as update_ldap.cfm and run it in your browser.

Reviewing the code

The following table describes the code:

Code	Description

`<cfset myServer="ldap.myco.com">`
`<cfset myUserName="cn=Directory Manager">`
`<cfset myPassword="password">`
	Initializes the LDAP connection information variables. Uses variables for all connection information so that any changes have to be made in only one place.

`<cfparam name="fullNameValue" default="">`
`<cfparam name="surnameValue" default="">`
`<cfparam name="emailValue" default="">`
`<cfparam name="phoneValue" default="">`
`<cfparam name="uidValue" default="">`
	Sets the default values of empty strings for the form field value variables. The data entry form uses cfinput fi elds with `value` attributes so that the form can be prefilled and so that, if the user submits an incomplete form, ColdFusion can retain any entered values in the form when it redisplays the page.

`<cfif isdefined("Form.action") AND Trim(Form.uid) IS NOT "">`	Ensures that the user entered a User ID in the form.
`<cfif Form.action is "add">`	If the user clicks Add, processes the code that follows.

© 2014 Adobe Systems Incorporated. All rights reserved.

`<cfif Trim(Form.fullName) is "" OR Trim(Form.surname) is "" OR Trim(Form.email) is "" OR Trim(Form.phone) is "">`
`<h2>You must enter a value in every field.</h2>`
`<cfset fullNameValue=Form.fullName>`
`<cfset surnameValue=Form.surname>`
 	
`<cfset emailValue=Form.email>`
`<cfset phoneValue=Form.phone>`
`<cfset uidValue=Form.uid>`
	If any field in the submitted form is blank, display a message and set the other form fields to display data that the user submitted.

`<cfelse>` `<cfset attributelist="objectclass=top, person, organizationalperson, inetOrgPerson; cn=#Trim(Form.fullName)#; sn=#Trim(Form.surname)#; mail=#Trim(Form.email)#; telephonenumber=#Trim(Form.phone)#; ou=Human Resources; uid=#Trim(Form.uid)#">`	If the user entered data in all fields, sets the attributelist variable to specify the entry's attributes, including the object class and the organizational unit (in this case, Human Resources). The Trim function removes leading or trailing spaces from the user data.

`<cfldap action="add" attributes="#attributeList#" dn="uid=#Trim(Form.uid)#, ou=People, o=Airius.com" server=#myServer# username=#myUserName# password=#myPassword#> <cfoutput>`
`<h3>Entry for User ID #Form.uid# has been added</h3>`
`</cfoutput>`
`</cfif>`
`</cfif>`
`</cfif>`
	Adds the new entry to the directory.

© 2014 Adobe Systems Incorporated. All rights reserved.

<cffo rm
action="update_ldap.cfm"
method="post">\
  <table>\
  <tr>\
  <td>Full Name:</td>\
  <td><cfinput type="Text"\
  name="fullName"\
  value=#fullNameValue#\
  size="20"\
  maxlength="30"\
  tabindex="1"></td>\
  </tr>\
 	
\.\.\.\
 	
<tr>\
  <td colspan="2">\
  <input type="Submit"\
  name="action"\
  value="Add"\
  tabindex="8"></td>\
  </tr>\
</table>\
 	
*All fields are required for
Add
</cfform>
	Outputs the data entry form, formatted as a table. Each `cfinput` field always has a value, set by the `value` attribute when the page is called. The `value` attribute lets ColdFusion update the form contents when the form is redisplayed after the user clicks Add. The code that handles cases in which the user fails to enter all the required data uses this feature.

<cfldap name="GetList" server=#myServer# action="query" attributes="cn,sn,mail,telephonenumber,uid" start="o=Airius.com" scope="subtree" filter="ou=Human Resources" sort="sn,cn" sortControl="asc, nocase">	Queries the directory and gets the common name, surname, e-mail address, telephone number, and user ID from the matching entries. Searches the subtree from the entry with the DN of o=Airius.com, and selects all entries in which the organizational unit is Human Resources. Sorts the results by surname and then common name (to sort by last name, then first). Sorts in default ascending order that is not case sensitive.

© 2014 Adobe Systems Incorporated. All rights reserved.

`<table border="1">`
`  <tr>`
`  <th>Full Name</th>`
`  <th>Surname</th>`
`  <th>Mail</th>`
`  <th>Phone</th>`
`  <th>UID</th>`
`  </tr>`
`  <cfoutput query="GetList">`
`  <tr>`
`  <td>#GetList.cn#</td>`
`  <td>#GetList.sn#</td>`
`  <td>#GetList.mail#</td>`
 	
`<td>#GetList.telephonenumber#</td>`
 	
`  <td>#GetList.uid#</td>`
`  </tr>`
`  </cfoutput>`
`  </table>`
`  </body>`
`  </html>`
	Displays the query results in a table.

Deleting a directory entry

To delete a directory entry,specify the entry DN.
The following example builds on the code that adds an entry. It adds Retrieve and Delete buttons. The Retrieve button lets you view a user's information in the form before you delete it.

1. Open update_ldap.cfm, which you created in Adding a directory entry.
2. Between the first and second `}}{{`cfiftags, add the following code:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cfelseif Form.action is "Retrieve">
 <cfldap name="GetEntry"
  server=#myServer#
  action="query"
  attributes="cn,sn,mail,telephonenumber,uid"
  scope="subtree"
  filter="uid=#Trim(Form.UID)#"
  start="o=Airius.com">
 <cfset fullNameValue = GetEntry.cn[1]>
 <cfset surnameValue = GetEntry.sn[1]>
 <cfset emailValue = GetEntry.mail[1]>
 <cfset phoneValue = GetEntry.telephonenumber[1]>
 <cfset uidValue = GetEntry.uid[1]>
<cfelseif Form.action is "Delete">
 <cfldap action="delete"
  dn="uid=#Trim(Form.UID)#, ou=People, o=Airius.com"
  server=#myServer#
  username=#myUserName#
  password=#myPassword#>
 <cfoutput><h3>Entry for User ID #Form.UID# has been deleted
 </h3></cfoutput>
```

3. At the end of the code for the Add button (the `input` tag with `Value=Add` at the bottom of the form{{}}, delete the `</td>` end tag.
4. After the end of the Add button `input`tag, add the following code:

```
 
 <input type="Submit"
   name="action"
   value="Retrieve"
   tabindex="7">
  
 <input type="Submit"
   name="action"
   value="Delete"
   tabindex="8"></td>
```

5. Save the file and run it in your browser.

Reviewing the code

The following table describes the code:

Code	Description

© 2014 Adobe Systems Incorporated. All rights reserved.

`<cfelseif Form.action is "Retrieve">`
`  <cfldap name="GetEntry"`
`  server=#myServer#`
`  action="query"`
 	
`attributes="cn,sn,mail,telephonenumber,uid"`
`  scope="subtree"`
`  filter="uid=#Trim(Form.UID)#"`
`  start="o=Airius.com">`
`  <cfset fullNameValue = GetEntry.cn[1]>`
`  <cfset surnameValue = GetEntry.sn[1]>`
`  <cfset emailValue = GetEntry.mail[1]>`
`  <cfset phoneValue = GetEntry.telephonenumber[1]>`
`  <cfset uidValue = GetEntry.uid[1]>`
	If the user clicks Retrieve, queries the directory and gets the information for the specified User ID.Sets the form field's `Value` attribute to the corresponding query column. This example uses the array index 1 to identify the first row of the GetEntry query object. Because the query always returns only one row, the index can be omitted.

`<cfelseif Form.action is "Delete">`
 	
`  <cfldap action="delete"`
`  dn="uid=#Trim(Form.UID)#, ou=People, o=Airius.com"`
`  server=#myServer#`
`  username=#myUserName#`
`  password=#myPassword#>`
`  <cfoutput><h3>Entry for User ID #Form.UID# has been deleted`
`  </h3>`
	The user clicks delete, deletes the entry with the specified User ID, and informs the user that the entry was deleted.

` `
`<input type="Submit"`
`  name="action"`
`  value="Retrieve"`
`  tabindex="7">`
` `
`<input type="Submit"`
`  name="action"`
`  value="Delete"`
`  tabindex="8"></td>`
	Displays submit buttons for the Retrieve and Delete actions.

Updating a directory entry

© 2014 Adobe Systems Incorporated. All rights reserved.

The cfldap tag lets you change the values of entry attributes. To do so, you specify the entry DN in the `dn` attribute, and list the attributes to change and their new values in the `attributes` attribute.
The following example builds on the code that adds and deletes an entry. It can update one or more of an entry's attributes. Because the UID is part of the DN, you cannot change it.

1. Open update_ldap.cfm, which you created in Adding a directory entry.
2. Between the `cfelseif` block and the `</cfif`tag, add the following code:

```
<cfelseif Form.action is "Update">
<cfset attributelist="cn=#Trim(form.FullName)#;
sn=#Trim(Form.surname)#; mail=#Trim(Form.email)#;
telephonenumber=#Trim(Form.phone)#">
<cfldap action="modify"
 modifytype="replace"
 attributes="#attributeList#"
 dn="uid=#Trim(Form.UID)#, ou=People, o=Airius.com"
 server=#myServer#
 username=#myUserName#
 password=#myPassword#>
<cfoutput><h3>Entry for User ID #Form.UID# has been updated</h3>
</cfoutput>
```

3. At the end of the code for the Delete button (the `input` tag with `Value=Delete` at the bottom of the form{{}}, delete the `</td>` mark.
4. After the end of the Delete button `input`tag, add the following code:

```
 
<input type="Submit"
 name="action"
 value="Update"
 tabindex="9"></td>
```

5. Save the file and run it in your browser.

Reviewing the code

The following table describes the code:

Code	Description

© 2014 Adobe Systems Incorporated. All rights reserved.

`<cfelseif Form.action is "Update">`
 	
`<cfset attributelist="cn=#Trim(form.FullName)#; sn=#Trim(Form.surname)#; mail=#Trim(Form.email)#;`
 	
`telephonenumber=#Trim(Form.phone)#">`
`<cfldap action="modify"`
`modifytype="replace"`
`attributes="#attributeList#"`
`dn="uid=#Trim(Form.UID)#, ou=People, o=Airius.com"`
`server=#myServer#`
`username=#myUserName#`
`password=#myPassword#>`
`<cfoutput><h3>Entry for User ID #Form.UID# has been updated</h3>`
`</cfoutput>`
	If the user clicks Update, sets the attribute list to the form field values and replaces the attributes for the entry with the specified UID.Displays a message to indicate that the entry was updated.This code replaces all of the attributes in a form, without checking whether they are blank. A more complete example would check for blank fields and either require entered data or not include the corresponding attribute in the `attributes` string.

` `
`<input type="Submit"`
`name="action"`
`value="Update"`
`tabindex="9"></td>`
	Defines the Submit button for the update action.

Adding and deleting attributes of a directory entry

The following table lists the cfldap tag attributes that you must specify to add and delete LDAP attributes in an entry:

Action	cfldap syntax

Add attribute to entry	`dn = "entry dn"`
	`action = "modify"`
	`modifyType = "add"`
	`attributes = "attribname=attribValue[;...]"`

Delete attribute from entry	`dn = "entry dn"`
	`action = "modify"`
	`modifyType = "delete"`
	`attributes = "attribName[;...]"`

© 2014 Adobe Systems Incorporated. All rights reserved.

You can add or delete multiple attributes in one statement. To do this, use semicolons to separate the attributes in the attribute string.
The following example specifies the description and seealso LDAP attributes:

```
attributes="description=Senior Technical Writer;seealso=writers"
```

You can change the character that you use to separate values of multivalued attributes in an attribute string. You can also change the character that separates attributes when a string contains multiple attributes. For more information, see *Specifying an attribute that includes a comma or semicolon* in Advanced topics.
You can add or delete attributes only if the directory schema defines them as optional for the entry's object class.

Changing a directory entry's DN

To change the DN of an entry, provide the following information in the cfldap tag:

```
dn="original DN"
action="modifyDN"
attributes="dn=new DN"
```

For example:

```
<cfldap action="modifyDN"
 dn="#old_UID#, ou=People, o=Airius.com"
 attributes="uid=#newUID#"
 server=#myServer#
 username=#myUserName#
 password=#myPassword#>
```

The new DN and the entry attributes must conform to the directory schema; therefore, you cannot move entries arbitrarily in a directory tree. You can only modify a leaf only. For example, you cannot modify the group name if the group has children.

Note

LDAP v2 does not let you change entry DNs.

© 2014 Adobe Systems Incorporated. All rights reserved.

Advanced topics

Some more advanced techniques enable you to use LDAP directories more effectively.

Specifying an attribute that includes a comma or semicolon

LDAP attribute values can contain commas. The cfldap tag normally uses commas to separate attribute values in a value list. Similarly, an attribute can contain a semicolon, which `cfldap` normally uses to delimit (separate) attributes in an attribute list. To override the default separator and delimiter characters, you use the `cfldap` tag `separator` and `delimiter` attributes.
For example, assume that you want to add the following attributes to an LDAP entry:

```
cn=Proctor, Goodman, and Jones
description=Friends of the company; Rationalists
```

Use the `cfldap` tag in the following way:

```
<cfldap action="modify"
 modifyType="add"
 attributes="cn=Proctor, Goodman, and Jones: description=Friends of the company;
  Rationalists"
 dn="uid=goodco, ou=People, o=Airius.com"
 separator="&"
 delimiter=":"
 server=#myServer#
 username=#myUserName#
 password=#myPassword#>
```

Using cfldap output

You can create a searchable collection from LDAP data.
The ability to generate queries from other queries is useful when cfldap queries return complex data. For more information on querying queries, see Using Query of Queries.

Viewing a directory schema

LDAP v3 exposes a directory's schema information in a special entry in the root DN. You use the directory root `subschemaSubentry` attribute to access this information.
The following ColdFusion query shows how to get and display the directory schema. It displays information from the schema's object class and attribute type definitions. For object classes, it displays the class name, superior class, required attribute types, and optional attribute types. For attribute types, it displays the type name, type description, and whether the type is single- or multivalued.
The example does not display all the information in the schema. For example, it does not display the matching rules. It also does not display the object class IDs, attribute type IDs, attribute type syntax IDs, or the object class descriptions. (The object class description values are all "Standard Object Class.")

> **Note**
>
> To be able to view the schema for an LDAP server, the server must support LDAP v3

© 2014 Adobe Systems Incorporated. All rights reserved.

This example does not work on iPlanet Directory Server 5.0. It does work on a 4.x server.

View the schema for an LDAP directory

1. Create a file that looks like the following:

```
<html>
<head>
 <title>LDAP Schema</title>
</head>

<body>
<!--- Start at Root DSE to get the subschemaSubentry attribute. --->
<cfldap
 name="EntryList"
 server="ldap.mycorp.com"
 action="query"
 attributes="subschemaSubentry"
 scope="base"
 start="">

<!--- Use the DN from the subschemaSubEntry attribute to get the schema. --->
<cfldap
 name="EntryList2"
 server="ldap.mycorp.com"
 action="query"
 attributes="objectclasses, attributetypes"
 scope="base"
 filter="objectclass=*"
 start=#entryList.subschemaSubentry#>

<!--- Only one record is returned, so query loop is not required. --->
<h2>Object Classes</h2>
<table border="1">
 <tr>
  <th>Name</th>
  <th>Superior class</th>
  <th>Must have</th>
  <th>May have</th>
 </tr>
 <cfloop index = "thisElement" list = #Entrylist2.objectclasses#>
  <cfscript>
   thiselement = Trim(thisElement);
   nameloc = Find("NAME", thisElement);
   descloc = Find("DESC", thisElement);
   suploc = Find("SUP", thisElement);
   mustloc = Find("MUST", thisElement);
   mayloc = Find("MAY", thisElement);
   endloc = Len(thisElement);
  </cfscript>
  <tr>
   <td><cfoutput>#Mid(thisElement, nameloc+6, descloc-nameloc-8)#
    </cfoutput></td>
   <cfif #suploc# NEQ 0>
    <td><cfoutput>#Mid(thisElement,
    suploc+5, mustloc-suploc-7)#
    </cfoutput></td>
   <cfelse>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

```
     <td>NONE</td>
    </cfif>
    <cfif #mayloc# NEQ 0>
     <td><cfoutput>#Replace(Mid(thisElement, mustloc+6,
      mayloc-mustloc-9), " $ ", ", ", "all")
      #</cfoutput></td>
     <td><cfoutput>#Replace(Mid(thisElement, mayloc+5,
     endloc-mayloc-8),
      " $ ", ", ", "all")#</cfoutput></td>
    <cfelse>
     <td><cfoutput>#Replace(Mid(thisElement, mustloc+6,
      endloc-mustloc-9), " $ ", ", ", "all")
      #</cfoutput></td>
     <td>NONE</td>
    </cfif>
   </tr>
  </cfloop>
 </table>
 <br><br>

  <h2>Attribute Types</h2>
 <table border="1" >
  <tr>
   <th>Name</th>
   <th>Description</th>
   <th>multivalued?</th>
  </tr>
  <cfloop index = "thisElement"
   list = #ReplaceNoCase(EntryList2.attributeTypes, ", alias", "<br> Alias",
   "all")# delimiters = ",">
   <cfscript>
    thiselement = Trim(thisElement);
    nameloc = Find("NAME", thisElement);
    descloc = Find("DESC", thisElement);
    syntaxloc = Find("SYNTAX", thisElement);
    singleloc = Find("SINGLE", thisElement);
    endloc = Len(thisElement);
   </cfscript>
   <tr>
    <td><cfoutput>#Mid(thisElement, nameloc+6, descloc-nameloc-8)#
     </cfoutput></td>
    <td><cfoutput>#Mid(thisElement, descloc+6, syntaxloc-descloc-8)#
     </cfoutput></td>
    <cfif #singleloc# EQ 0>
      <td><cfoutput>Yes</cfoutput></td>
    <cfelse>
      <td><cfoutput>No</cfoutput></td>
    </cfif>
   </tr>
  </cfloop>
 </table>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

```
</body>
</html>
```

2. Change the server from `ldap.mycorp.com` to your LDAP server. You might also need to specify a user ID and password in the cfldap tag.
3. Save the template as `ldapschema.cfm` in `myapps` under your web root directory and view it in your browser.

Reviewing the code

The following table describes the code and its function:

Code	Description
`<cfldap name="EntryList" server="ldap.mycorp.com" action="query" attributes="subschemaSubentry" scope="base" start="">`	Gets the value of the subschemaSubentry attribute from the root of the directory server. The value is the DN of the schema.
`<cfldap name="EntryList2" server="ldap.mycorp.com" action="query" attributes="objectclasses, attributetypes" scope="base" filter="objectclass=*" start=#entryList.subschemaSubentry #>`	Uses the schema DN to get the objectclasses and attributetypes attributes from the schema.

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<h2>Object Classes</h2>
 <table border="1">
  <tr>
   <th>Name</th>
   <th>Superior class</th>
   <th>Must have</th>
   <th>May have</th>
  </tr>
  <cfloop index = "thisElement"
list = #Entrylist2.objectclasses#>

   <cfscript>
    thiselement =
Trim(thisElement);
    nameloc = Find("NAME",
thisElement);
    descloc = Find("DESC",
thisElement);
    suploc = Find("SUP",
thisElement);
    mustloc = Find("MUST",
thisElement);
    mayloc = Find("MAY",
thisElement);
    endloc = Len(thisElement);
   </cfscript>
```

Displays the object class name, superior class, required attributes, and optional attributes for each object class in a table.The schema contains the definitions of all object classes in a comma delimited list, so the code uses a list type cfloop tag.The thisElement variable contains the object class definition. Trim off any leading or trailing spaces, then use the class definition field keywords in Find functions to get the starting locations of the required fields, including the Object class ID. (The ID is not displayed.) Gets the length of the `thisElement` string for use in later calculations.

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<tr>
 <td><cfoutput>#Mid(thisElement,
nameloc+6, descloc-nameloc-8)#
   </cfoutput></td>
  <cfif #suploc# NEQ 0>

<td><cfoutput>#Mid(thisElement,
suploc+5, mustloc-suploc-7)#
     </cfoutput></td>
    <cfelse>
     <td>NONE</td>
    </cfif>
    <cfif #mayloc# NEQ 0>

<td><cfoutput>#Replace(Mid(thisEle
ment, mustloc+6,
      mayloc-mustloc-9), " $ ", ",
", "all")#</cfoutput></td>

<td><cfoutput>#Replace(Mid(thisEle
ment, mayloc+5, endloc-mayloc-8),
      " $ ", ", ",
"all")#</cfoutput></td>
    <cfelse>

<td><cfoutput>#Replace(Mid(thisEle
ment, mustloc+6,
      endloc-mustloc-9), " $ ", ",
", "all")#</cfoutput></td>
     <td>NONE</td>
    </cfif>
   </tr>
  </cfloop>
 </table>
```

Displays the field values. Uses the Mid function to extract individual field values from the thisElement string.The top object class does not have a superior class entry. Handles this special case by testing the suploc location variable. If the value is not 0, handles normally, otherwise, output "NONE". There might not be any optional attributes. Handles this case similarly to the superior class. The calculation of the location of required attributes uses the location of the optional attributes if the field exists; otherwise, uses the end of the object class definition string.

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<h2>Attribute Types</h2>
 <table border="1" >
  <tr>
   <th>Name</th>
   <th>Description</th>
   <th>multivalued?</th>
  </tr>
  <cfloop index = "thisElement"
   list =
#ReplaceNoCase(EntryList2.attribut
eTypes, ", alias", "<br> Alias",
   "all")# delimiters = ",">
   <cfscript>
    thiselement =
Trim(thisElement);
    nameloc = Find("NAME",
thisElement);
    descloc = Find("DESC",
thisElement);
    syntaxloc = Find("SYNTAX",
thisElement);
    singleloc = Find("SINGLE",
thisElement);
    endloc = Len(thisElement);
   </cfscript>
   <tr>

<td><cfoutput>#Mid(thisElement,
nameloc+6, descloc-nameloc-8)#
     </cfoutput></td>

<td><cfoutput>#Mid(thisElement,
descloc+6, syntaxloc-descloc-8)#
     </cfoutput></td>
    <cfif #singleloc# EQ 0>

<td><cfoutput>Yes</cfoutput></td>
    <cfelse>

<td><cfoutput>No</cfoutput></td>
    </cfif>
   </tr>
  </cfloop>
```

Does the same types of calculations for the attribute types as for the object classes. The attribute type field can contain the text ", alias for....". This text includes a comma, which also delimits attribute entries. Use the REReplaceNoCase function to replace any comma that precedes the word "alias" with an HTML `<br>` tag. The attribute definition includes a numeric syntax identifier, which the code does not display, but uses its location in calculating the locations of the other fields.

Referrals

An LDAP database can be distributed over multiple servers. If the requested information is not on the current server, the LDAP v3 standard provides a mechanism for the server to return a referral to the client that informs the client of an alternate server. (This feature is also included in some LDAP v2-compliant servers.)
ColdFusion can handle referrals automatically. If you specify a nonzero `referral` attribute in the cfldap tag, ColdFusion sends the request to the server specified in the referral.
The `referral` attribute value specifies the number of referrals allowed for the request. For example, if the `referral` attribute is 1, and server A sends a referral to server B, which then sends a referral to server C, ColdFusion

© 2014 Adobe Systems Incorporated. All rights reserved.

returns an error. If the `referral` attribute is 2, and server C has the information, the LDAP request succeeds. The value to use depends on the topology of the distributed LDAP directory, the importance of response speed, and the value of response completeness.
When ColdFusion follows a referral, the `rebind` attribute specifies whether ColdFusion uses the `cfldap` tag login information in the request to the new server. The default, No, sends an anonymous login to the server.

Managing LDAP security

When you consider how to implement LDAP security, consider server security and application security.

Server security

The cfldap tag supports secure socket layer (SSL) v2 security. This security provides certificate-based validation of the LDAP server. It also encrypts data transferred between the ColdFusion server and the LDAP server, including the user password, and ensures the integrity of data passed between the servers. To specify SSL v2 security, set the `cfladap` tag `secure="cfssl_basic"` attribute.

About LDAP Server Security

ColdFusion uses Java Native Directory Interface (JNDI), the LDAP provider, and an SSL package to create the client side of an SSL communication. The LDAP server provides the server side. The LDAP server that the `cfldap` tag connects to using SSL holds an SSL server certificate, a certificate that is securely "signed" by a trusted authority and identifies (authenticates) the sender. During the initial SSL connection, the LDAP server presents its server certificate to the client. If the client trusts this certificate, the SSL connection is established and secure LDAP communication can begin.
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. The ColdFusion 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
```

The keytool utility initial keypass password is "change it". For more information on using the keytool utility, see the Sun JDK documentation.
Once ColdFusion establishes secure communication with the server, it must provide the server with login credentials. You specify the login credentials in the `cfldap` tag `username` and `password` attributes. When the server determines that the login credentials are valid, ColdFusion can access the directory.

Using LDAP security

To use security, first ensure that the LDAP server supports SSL v2 security.
Specify the `cfldap` tag `secure` attribute as follows:

```
secure = "cfssl_basic"
```

For example:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cfldap action="modify"
 modifyType="add"
 atributes="cn=Lizzie"
 dn="uid=lborden, ou=People, o=Airius.com"
 server=#myServer#
 username=#myUserName#
 password=#myPassword#
 secure="cfssl_basic"
 port=636>
```

The `port` attribute specifies the server port used for secure LDAP communications, which is 636 by default. If you do not specify a port, ColdFusion attempts to connect to the default, nonsecure, LDAP port 389.

Application security

To ensure application security, prevent outsiders from gaining access to the passwords that you use in cfldap tags. The best way to do this is to use variables for your `username` and `password` attributes. You can set these variables on one encrypted application page. For more information on securing applications, see Securing Applications.

#back to top

© 2014 Adobe Systems Incorporated. All rights reserved.

Solr search support

The Solr search service is an open source enterprise search server based on the Lucene Java search library.

© 2014 Adobe Systems Incorporated. All rights reserved.

CFML tags used to configure Solr

- **cfcollection** To create a collection using this tag, specify `engine="solr"}}with {{action="create"/"list"/"map"`. For a Solr collection, you do not need to specify the language. Language can be specified in the <cfindex> tag. All other attributes remain the same.
- **cfindex** ColdFusion detects the search engine based on the collection name.

© 2014 Adobe Systems Incorporated. All rights reserved.

Solr search service

Solr is a full-text search engine, based on Lucene, that contains the following features:

- XML/HTTP Interfaces
- Loose schema to define types and fields
- Web Administration Interface
- Extensive Caching
- Index Replication
- Extensible Open Architecture
- Written in Java, deployable as a WAR
- Support for stemming
- Support for MS Office 2007 file formats
  The ColdFusion installer automatically creates the ColdFusion 11 Add-on Service, which contains the Solr web application. For UNIX and Linux, you need to start and stop the Solr shell script.

© 2014 Adobe Systems Incorporated. All rights reserved.

Language support

Solr supports the following languages:

- Danish
- Dutch
- Finnish
- French
- German
- Italian
- Norwegian
- Spanish
- Portugese
- Russian
- Swedish
- Chinese
- Japanese
- Korean
- Czech
- Greek
- Thai

  Solr supports documents in any language. If the document has a language (for example, Arabic) not listed above, it can still index the content, but stemming is not available. In this case, do not specify a language attribute in the cfindex tag.

  Solr supports stemming. That is, it considers the root form (stem) of the word for search. This applies only if you provide the attribute `language`.

© 2014 Adobe Systems Incorporated. All rights reserved.

Solr search examples

The following examples show the search syntax used while performing search operations using Solr:

- Single word search: The following example shows how to search for a single word in a collection:

```
<cfsearch name="qsearch1" collection="solr_complex" criteria="Graphics">
```

- Multiple word search: The following example shows how to search a document or query having words "ColdFusion" and "Green" in it:

```
<cfsearch name="qsearch1"
collection="solr_complex"
criteria="+Green +Coldfusion">
```

- Search with at least one word: The following example shows how search for at least "Coldfusion" OR (Green OR Blue):

```
<cfsearch name="qsearch1"
collection="solr_complex"
criteria=" +Coldfusion Green Blue">
```

- Search for one word, but not the other: The following example shows how to search for "Green" but NOT "Coldfusion":

```
<cfsearch name="qsearch1"
collection="solr_complex"
criteria=" -Coldfusion +Green">
```

- Fuzzy search: The following example shows how to search words like roam, roams, foam, foams:

```
<cfsearch name="qsearch1"
collection="solr_complex"
criteria=" roam~">
```

  Alternate way to perform a fuzzy search for "roam":

```
<cfsearch name="qsearch1"
collection="solr_complex"
criteria="roam~">
```

  Searching for higher similarity with roam:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cfsearch name="qsearch1"
collection="solr_complex"
criteria=" roam~0.8" >
```

- Wildcard search: The following syntax searches for 'test', 'text', 'teat', and so on:

```
<cfsearch name="qsearch1"
collection="solr_complex"
criteria=" te?t">
```

This example searches for 'test', 'text', 'teeeeeext', and 'texyzt':

```
<cfsearch name="qsearch1"
collection="solr_complex"
criteria=" te*t">
```

Note

You cannot use a * or question mark (?) symbol as the first character of a search.

- Proximity search: To search for "apache" and "jakarta" within five words of each other in a document, use the following search:

```
<cfsearch name="qsearch1"
collection="solr_complex"
criteria="jakarta apache" ~10>
```

- Range Search: Following searches all documents with title between 'fuzzy1.txt' to 'text1.txt':

```
<cfsearch name="qsearch"
collection="solr_srch"
criteria="title:fuzzy1.txt TO text1.txt">
```

To search a document whose modification date is between a given range:

```
<cfsearch name="qsearch"
collection="solr_srch"
criteia="modified:20080101 TO 20500101">
```

These

© 2014 Adobe Systems Incorporated. All rights reserved.

ranges are inclusive of start and end terms. To exclude them, use curly brackets{} instead.

- Field search: To search any document whose title contains "fuzzy1.txt"

```
<cfsearch name="qsearch"
collection="solr_srch"
criteria="title:fuzzy1.txt">
```

Searching for document that contains title as 'fuzzy1.txt' OR 'fuzzy2.txt':

```
<cfsearch name="qsearch"
collection="solr_srch"
criteria="title:fuzzy?.txt">
```

The following syntax can be used to perform the same search:

```
<cfsearch name="qsearch"
collection="solr_srch"
criteria= ""title:fuzzy1.txt"" OR ""title:fuzzy2.txt"">
```

Alternatively, you can search using the following syntax:

```
<cfsearch name="qsearch"
collection="solr_srch"
criteria="title:(test* +fuzzy1*)">
```

- String search:

```
<cfsearch name="qsearch1"
collection="solr_complex"
criteria="Cold Fusiongava" OR "Internet Tools">
```

- Searching synonyms: There are two ways to search documents that have synonymous words like 'MB', 'megabyte', 'gig', and so on:
  1. If collection is not yet created, go to: `<cf_home>/solr/multicore/template/conf/synonyms.txt` This file contains some default mappings such as `'GB, gig, gigabyte, gigabytes'`. Define your synonym mappings in the next row.
  2. If you want to add a synonym mapping for a collection that is already created, go to `<collection_location>/conf/synonyms.txt` and define your mapping.Restart the Solr server after defining mapping.

© 2014 Adobe Systems Incorporated. All rights reserved.

Term highlighting

By default, Solr highlights searched terms in the summary content as described in the following snippet:

```
<cfsearch
collection="syn1"
criteria="Services solr"
name="results"
status="r"
suggestions="always"
contextPassages="1">
```

To highlight contents in the entire document, modify the solrconfig.xml and schema.xml files. These files are available in the following locations:

- <Collection Directory>/conf: Modify files in this location to apply the changes to all future Solr collections.
- <Solr Home>/multicore/template/conf: Modify files in this location to apply the changes only to a particular collection.

1. Stop ColdFusion 11 Add-on service.
2. Replace the following section in the solrconfig.xml

```
<requestHandler name="standard" class="solr.StandardRequestHandler"
default="true">
    <!-- default values for query parameters -->
     <lst name="defaults">
       <str name="echoParams">explicit</str>
       <str name="hl.fl">summary title </str>
       <!-- omp = Only More Popular -->
       <str name="spellcheck.onlyMorePopular">false</str>
       <!-- exr = Extended Results -->
       <str name="spellcheck.extendedResults">false</str>
       <!--  The number of suggestions to return -->
       <str name="spellcheck.count">1</str>

     </lst>
     <arr name="last-components">
      <str>spellcheck</str>
     </arr>
  </requestHandler>
```

with

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<requestHandler name="standard" class="solr.StandardRequestHandler"
default="true">
    <!-- default values for query parameters -->
     <lst name="defaults">
       <str name="echoParams">explicit</str>
       <str name="hl.fl">contents title </str>
       <!-- omp = Only More Popular -->
       <str name="spellcheck.onlyMorePopular">false</str>
       <!-- exr = Extended Results -->
       <str name="spellcheck.extendedResults">false</str>
       <!--  The number of suggestions to return -->
       <str name="spellcheck.count">1</str>

     </lst>
     <arr name="last-components">
      <str>spellcheck</str>
     </arr>
  </requestHandler>
```

3. Replace the following section in the schema.xml

```
<field name="contents" type="text" indexed="true"  stored="false"
required="false" multiValued="true" omitNorms="true"/>
```

with

```
<field name="contents" type="text" indexed="true" stored="true"
required="false" multiValued="true" omitNorms="true"/>
```

4. Restart Solr.
5. Reindex the collection.

Note

The modifications to solrconfig.xml and schema.xml will increase the index size.

© 2014 Adobe Systems Incorporated. All rights reserved.

Configuring Solr memory

Perform the following steps to increase the memory for Solr.

Non-Windows platforms

1. Stop Solr using the following command:`sudo ./cfsolr stop`
2. In the cfsolr script, modify the line that starts with `VMARGS=` by changing the default value `-Xmx512m` with an appropriate value. For example, `-Xmx1024m.`
3. Start Solr using the following command:`sudo ./cfsolr start`

Windows platform

1. Stop the Solr service `ColdFusion 11 Add-on Service`.
2. In the C:\ColdFusion11\cfusion\jetty\jetty.lax, modify the line that starts with `lax.nl.java.option.additional=` by changing the default value `-Xmx512m` with an appropriate value. For example, `-Xmx1024m.`
3. Start Solr service.

© 2014 Adobe Systems Incorporated. All rights reserved.

Solr enhancements in ColdFusion 9.0.1

Apart from overall improvement in the accuracy of indexing, the following are the enhancements:

- Displays correct MIME types for all documents
- Enhanced support for indexing of metadata for binary files such as mp3 and JPEG
- Support for the attribute `previousCriteria` (in the tag `cfsearch`)
- Both the tags `cfindex` and `cfsearch` support the attribute `categoryTree.`
- A new section to enable/disable term highlighting for entire document has been added to the ColdFusion Administrator. This applies only if Solr is running on local machine.

> **Note**
>
> The following steps apply only if Solr is running on local machine. If Solr is on a remote machine, follow the steps provided in the section Term highlighting in *Developing ColdFusion Applications*.

1. In the ColdFusion Administrator, go to ColdFusion Collections and then click a Solr collection.
2. Use the Enable/Disable button to enable/disable term highlighting.
3. Reindex the collection.

> **Note**
>
> Enabling term highlighting increases the size of the Solr collection. So ensure that you allocate adequate memory for Solr if you are enabling term highlighting.

Upgrading Solr

Solr is upgraded as part of updater.
Therefore, to use Solr with ColdFusion 9.0.1, upgrade Solr.
For local installation, Solr is auto-upgraded when you run the ColdFusion Updater.
For remote installation, manually upgrade Solr using the following steps:

1. Stop Solr.
2. Back up solr.xml available in *Solr_Home*/multicore.
3. Uninstall Solr.
4. Reinstall the standalone version of Solr available on Adobe download location.
5. Stop Solr (if it has started automatically).
6. Bring back the backed up copy of solr.xml to *Solr_Home*/multicore.

> **Note**
>
> After you upgrade, ensure that you reindex the entire Solr collection before you use the search service.

© 2014 Adobe Systems Incorporated. All rights reserved.

Solr enhancements in ColdFusion 10

Enhancements in this release let you

- Use Data Import Handler for database indexing
- Index and search based on dynamic custom fields
- Reload individual collections
- Add languages for search
- Secure your search system using ColdFusion Administrator (Data & Services > Solr Server > Show Advanced Settings > Use HTTPs connection).
- Autocommit indexed documents
- Boost specific fields or entire document for improved search results.

Modifications to file location and filenames

- In the case of standalone installation, all Solr files reside in the jetty folder ColdFusion 10\cfusion\jetty (previously ColdFusion 9\cfusion\solr).
- On Windows, the Solr service has been renamed as ColdFusion 10 Jetty Service (previously ColdFusion 9 Solr Service, and as of ColdFusion 11, Add-on Service).
- On Windows, the executable file has been renamed as jetty.exe (previously solr.exe)

Using Data Import Handler

In ColdFusion 9, indexing database was a two step process (of querying database using the tag `cfquery` and indexing the query using the tag `cfindex`). In ColdFusion 10, you need not use `cfquery` to get data; rather Solr directly communicates with the database and fetches data using Data Import Handler helping you improve indexing performance.
You can perform a full or partial indexing depending on your requirement. For example, when you index the database for the first time, you may do a full indexing. For any updates in the database, you can perform partial indexing to update your collection.

Indexing using Data Import Handler

The following steps help you configure Data Import Handler for indexing databases:

1. Do the following:

- **For full import:**Create the following dataconfig.xml to define mapping of database table columns to Solr:

```
<dataconfig>
 <datasource driver="org.hsqldb.jdbcDriver" url="jdbc:mysql:/temp/example/ex"
user="user"
             password="user"/>
 <document name="products">
  <entity name="item" query="select * from item">
   <field column="ID" name="id"/>
   <field column="NAME" name="name"/>
  </entity>
 </document>
</dataconfig>
```

- **For delta import:**Create the following dataconfig.xml:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<dataconfig>
 <dataSource
 driver="com.mysql.jdbc.Driver" "jdbc:mysql:/temp/example/ex" user="user"
password="password" />
 <document name="rrr">
 <entity name="item" pk="ID" query="select
ID,NAME,PRICE,WEIGHT,last_modified  from item"
         deltaimportquery="select ID,NAME,PRICE,WEIGHT,last_modified from item
where ID='${dataimporter.delta.id}'"
         deltaquery="select id from item where last_modified >
'${dataimporter.last index_time}'">
  <field column="ID" name="uid"/>
  <field column="NAME" name="name_t"/>
  <field column="PRICE" name="price_f"/>
  <field column="WEIGHT" name="weight_d"/>
  <entity name="feature" pk="ITEM_ID"
          query="select description as   features from feature where
item_id='${item.ID}'">
   <field name="features_t" column="features"/>
  </entity>
  <entity name="item_category" pk="ITEM_ID, CATEGORY_ID"
          query="select CATEGORY_ID from item_category where
ITEM_ID='${item.ID}'">
   <entity name="category" pk="ID"
           query="select description as cat from category where id =
'${item_category.CATEGORY_ID}'">
    <field column="cat" n a m
           e="cat t"/>
   </entity>
  </entity>
 </entity>
</dataconfig>
```

For details of the attributes, see *Schema for the data config* in the section Configuration in data-config.xml at the URL http://wiki.apache.org/solr/DataImportHandler.

- Ensure that `last_modified` is the column name of the table that you index and the column has time stamp.
- Unless you have this column mapped, partial import fails.
- The latest timestamp is created in the dataimport.properties available in the collection location.

1. Save the file in the conf directory of the collection that you have created.
2. In the solrconfig.xml (in the conf directory), uncomment the following section.

```
<!--
  <requestHandler name="/dataimport"
class="org.apache.solr.handler.dataimport.DataImportHandler">
    <lst name="defaults">
      <str name="config">data-config.xml</str>
    </lst>
  </requestHandler>
  -->
```

© 2014 Adobe Systems Incorporated. All rights reserved.

This enables Data Import Handler.

3. Reload the collection.
4. Use one of the following `cfindex` actions: `fullImport`, `deltaImport`, `status`, or `abort`.

Modifications to the tag cfindex

New value for the attribute type

To use Data Import Handler, specify `type=dih`.

New actions

The following new actions have been added to the tag `cfindex` to help Solr directly fetch data from the database.

- `fullimport:` To index full database. For instance, when you index the database for the first time.For example,

```
<cfindex collection="dih1" type="DIH" action="fullimport" status="st">
<cfsearch collection="dih1" criteria="damaged" name="s" orderby="price_f desc"
status="stat">
```

- `deltaimport:` For partial indexing. For instance, for any updates in the database, instead of a full import, you can perform delta import to update your collection.For example,

```
<cfindex collection="dih1" type="DIH" action="deltaimport" status="st">
<cfsearch collection="dih1" criteria="damaged" name="s" orderby="price_f desc"
status="stat">
```

- `status:` Provides the status of indexing, such as the total number of documents processed and status such as idle or running.For example,

```
<cfindex collection="bt" type="DIH" action="status" status="s">
<cfoutput>
 Rows Indexed : #s.TOTAL_DOCUMENTS_PROCESSED#
 <br>
</cfoutput>
<cfoutput>
 Status of Solr Server : #s.status#
 <br>
</cfoutput>
```

- `abort:` Aborts an ongoing indexing task.

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cfindex collection="bt" type="DIH" action="abort" status="s">
<cfoutput>
 Status of Solr Server : #s.status#
 <br>
</cfoutput>
```

Storing your custom data

In addition to indexing, you can store custom information using custom fields that are dynamically defined.
For example, while indexing a PDF, you can store information such as author and date of publication as shown in the following example:

```
<cfindex collection="CodeColl"
 action="refresh"
    type="file"
    key="C:\learning_resources\wwwroot\vw_files\gettingstarted.pdf"
    urlpath="http://localhost:8500/vw_files/"
    language="English"
    title="Cfindex Reference page"
    status="info"
 blurb_s=information
 publisher_s=adobe/>
```

To specify custom fields, use the following syntax:

```
<cfindex ...
    datefield_dt=#date1#
 column_i=#secondaryColumn#
 body=#primaryColumn#
 ...../>
```

Note

Custom fields can contain only lower case characters.

In the code, `_i` stands for integer custom data whose value is stored and indexed. Any field name that ends with `_i` is treated as a Solr integer.
Similarly, `_s` stands for string custom data.
All the supported datatypes are listed in the schema.xml:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<dynamicfield name="*_i" type="sint" indexed="true" stored="true"/>
<dynamicfield name="*_s" type="string" indexed="true" stored="true"/>
<dynamicfield name="*_l" type="slong" indexed="true" stored="true"/>
<dynamicfield name="*_t" type="text" indexed="true" stored="true"/>
<dynamicfield name="*_b" type="boolean" indexed="true" stored="true"/>
<dynamicfield name="*_f" type="sfloat" indexed="true" stored="true"/>
<dynamicfield name="*_d" type="sdouble" indexed="true" stored="true"/>
<dynamicfield name="*_dt" type="date" indexed="true" stored="true"/>
<dynamicfield name="random*" type="random"/>
```

Note

`_dt` supports only the date formats supported by ColdFusion.

Example

```
<cfindex collection="custom1" type="file" action="update"
key="#datadir#/text/text1.txt"
         status="s" date_dt=NOW>
<cfsearch collection="custom1" criteria="" name="n" orderby="title">
```

New attribute orderBy in cfsearch

A new attribute `orderBy` has been added to `cfsearch`. It sorts the custom field column rank order. This is an optional attribute and by default, it sorts in ascending order.

```
<cfsearch
collection="someCollection"
criteria="someCriteria"
orderby="field1 desc,field2,field3 asc"  ...../>
```

Autocommit indexed documents

Automatically commit the changes to the search server by setting the attribute `autoCommit` to `true` in `cfindex` as shown in the following example:

```
<cfindex collection="autocommit_check" action="update" type="file"
key="#Expandpath(".")#/_boost1.txt" first_t="fieldboost" second_t="secondfield"
fieldboost="first_t:1,second_t:2" docboost="6" autocommit="true">
```

If `false`, indexed documents are not committed unless you specifically commit using cfindex `action="commit"`. By default, the value is set to `true`.

Improving search result rankings

© 2014 Adobe Systems Incorporated. All rights reserved.

The following attributes in `cfindex` help you improve the search result rankings:

- `fieldBoost`: Boost specific fields while indexing.`fieldBoost` enhances the score of the fields and thereby the ranking in the search results. Multiple fields can be boosted by specifying the values as a comma-separated list.
- `docBoost`: Boost entire document while indexing.`docBoost` enhances the score of the documents and thereby the ranking in the search results.

Variations from ColdFusion 9

- ColdFusion 9 had limited support for custom fields namely `custom1`, `custom2`, `custom3`, and `custom4`. In ColdFusion 10, custom fields are dynamic.
- In ColdFusion 9, all custom fields are displayed. In ColdFusion 10, `cfdump` yields only fields that have data{{}}. That is, if you have specified only `custom 1` and `custom 2`, only those two fields are displayed.
- Consider the following code:

```
<cfsearch criteria='some_criteria and column_i: [ 10 - 20 ]'...>
```

  Here, `some_criteria` indicates filtering. For example `column_i: [ 10 - 20 ]` means search all items whose values are between 10 and 20. `column_i` is the custom field provided by user while indexing.This option was available in ColdFusion 9, but limited to four custom fields. In ColdFusion 10, the options are unlimited.

- In ColdFusion 10, you can sort the order in which search results have to be returned.

> **Note**
>
> When you search a Solr collection for field type string, the criteria should be within quotes, for example

```
criteria='string_s:"something missing"'
```

Solr Search example 1

```
<cfsearch collection="custom1" criteria="rank_i:[2 TO 4]" name="s1"
orderby="value_i"
          status="s">
<cfdump var="#s1#">
```

Solr Search example 2: Using wild cards

```
<!------ Searching with wildcard *--------->
<cfsearch collection="custom1" criteria="text1_t:blue*" name="s1" status="s"
          orderby="cf_i">
<cfoutput>
 Searching Text with wildcard * :#s.FOUND#
 <br>
</cfoutput>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

Search limitations

Limitations: Searching custom fields of type string

Strings cannot be searched with wild cards except `*`. Since strings are not tokenized, you cannot search any word in a string. String can be searched as a whole and not as individual words. For example, in the case of `str_s="All work and no play"`, you cannot search for `play` or `work` in this string. You have to perform search using full string. However, strings can be sorted in search (using `orderby` attribute).

Limitations: Searching custom fields of type text

Text type field is tokenized and therefore you can search for any word in the text. You can also search text using wild cards. The only limitation is that text type cannot be sorted while searching. Since text type is tokenized, Solr treats text as a set of tokens, and therefore sorting is not possible.

Limitations: Searching custom fields is case-sensitive

Custom fields can be searched only in lowercase. For example, if the name of the custom field is `newDate`, you must search for `newdate`.

Limitations: Using the attribute orderBy

The attribute `orderBy` must be used with untokenized fields such as stings.

Reload collection

In ColdFusion 9, to reload an individual collection you have to restart Solr, which reloads all the collections. So, whenever you modify schema.xml, for example while adding language or field type, or when you enable Data Import Handler, you have to restart Solr so that changes take effect.
In ColdFusion 10, you can limit the reload to a specific collection which helps in significant performance improvement.
To reload a collection,

1. In the ColdFusion Administrator, go to Data & Services > ColdFusion Collections.
2. For the specific collection, click Reload icon in Solr Collections > Actions.

Support for additional languages

ColdFusion supports search and indexing for 17 languages in addition to English. If your language is not available, you can add to the list, provided Solr supports indexing and search for that language.
For details of the supported languages, see http://wiki.apache.org/solr/LanguageAnalysis.
If Solr supports the language, you can add it as follows:

1. Add filter class in the schema.xml.

- Add the field type as follows:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<fieldtype name="text_th" class="solr.TextField">
 <analyzer class="org.apache.lucene.analysis.th.ThaiAnalyzer"/>
</fieldtype>
....
<fieldtype name="text_hi" class="solr.TextField">
 <analyzer>
  <tokenizer class="solr.StandardTokenizerFactory"/>
  <filter class="solr.IndicNormalizationFilterFactory"/>
  <filter class="solr.HindiNormalizationFilterFactory"/>
  <filter class="solr.HindiStemFilterFactory"/>
 </analyzer>
</fieldtype>
```

- Add the field name as follows:

```
<field name="contents_pt" type="text_pt" indexed="true" stored="false"
       required="false"/>
<field name="contents_hi" type="text_hi" indexed="true" stored="false"
       required="false"/>
```

1. In the ColdFusion Administrator, go to Data & Services > Solr Server.
2. In the section Configure Indexing languages, specify the following:
   - **New language:** Specify the language, for example `Hindi`.
   - **New language suffix:** Specify a suffix for the language, for example `hi` for Hindi.

Security enhancements in ColdFusion 10

Securing Solr

Since Solr cannot be done at a document level or communication level. But you can add security to your Solr search by ensuring that the application server on which Solr runs is secure. To do this,

1. Secure the application server on which Solr runs; the default is jetty.
2. In the ColdFusion Administrator, go to Data & Services > Solr Server.
3. In Configure Solr server, click Show advanced settings.
4. Check Use HTTPS connection and then specify the Solr Admin HTTPS Port.

> **Note**
>
> Recommended to use when you use DIH.

Support for authentication

In ColdFusion 9, any user can access and add, update, and delete documents for indexing. This release provides basic authentication in jetty to secure access to collections.

1. Modify the web.xml of jetty server as follows:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<security-constraint>
 <web-resource-collection>
  <web-resource-name>
   Solr authenticated application
  </web-resource-name>
  <url-pattern>
   /core1/
  </url-pattern>
  {*}
 </web-resource-collection>
 <auth-constraint>
  <role-name>
   core1-role
  </role-name>
 </auth-constraint>
</security-constraint>
<login-config>
 <auth-method>
  BASIC
 </auth-method>
 <realm-name>
  Test Realm
 </realm-name>
</login-config>
```

2. Uncomment the following section in jetty.xml:

```
<set name="UserRealms">
 <array type="org.mortbay.jetty.security.UserRealm">
  <item>
   <new>
    <set name="name">
     Test Realm
    </set>
    <set name="config">
     <systemproperty name="jetty.home" default="."/>
     /etc/realm.properties
    </set>
   </new>
  </item>
 </array>
</set>
```

3. Add your username and password in /etc/example/realm.properties file as follows:`username:password, core1-role`
4. In the ColdFusion Administrator, go to Data & Services > Solr Server > Click Show Advanced Settings in Configure Solr Server section.
5. Specify the username and password and then click Submit.

© 2014 Adobe Systems Incorporated. All rights reserved.

Note

If you do not specify the credentials, index operation occurs without authentication.

© 2014 Adobe Systems Incorporated. All rights reserved.

ColdFusion ORM

Relational databases are the core of most enterprise applications. However, when you map a relational database to objects, it becomes a challenge. Object relational mapping (ORM) is a programming framework that allows you to define a mapping between an application object model and the relational database.

In an object model, the application objects are not aware of the database structure. Objects have properties and references to other objects. Databases consist of tables with columns that may be related to other tables. ORM provides a bridge between the relational database and the object model.

By using ORM, you can access and update data entirely using the object model of an application. ORM provides features such as:

- Database vendor independence
- Caching
- Concurrency
- Performance optimization

- Introducing ColdFusion ORM
- Architecture
- Configure ORM
- Define ORM mapping
- Working with objects
- Using queries
- Transaction and concurrency
- Performance optimization
- ORM session management
- Autogenerating database schema
- Support for multiple data sources for ORM
- ColdFusion ORM search

#back to top

© 2014 Adobe Systems Incorporated. All rights reserved.

Introducing ColdFusion ORM

In previous ColdFusion releases, database access was achieved by:

- Managing relational data using tags such as cfquery, cfinsert, and cfupdate, which handle SQL statements, or via stored procedures.
- Managing objects using ColdFusion components (CFCs), and object lifecycle using the application itself
- Writing SQL queries for each CFC, even for basic CRUD (Create, Retrieve, Update, and Delete) operations.
  The complexity of managing these tasks increases as your application grows.
  ColdFusion ORM automates most of these tasks, which:
- Makes application code cleaner and more manageable
- Enhances your productivity and lets you develop database applications faster
- Creates applications that can run faster because of built-in ORM optimizations
- Minimizes the amount of code you write

Apart from providing a framework for mapping the object model with the relational database, ColdFusion ORM provides data query and retrieval facilities.

For more information, see www.hibernate.org.

© 2014 Adobe Systems Incorporated. All rights reserved.

ColdFusion ORM example

ColdFusion ORM manages persistence through objects, which are also called entities in the ORM context. In ColdFusion ORM, persistence is managed through CFCs and their properties. Each persistent CFC in a ColdFusion application maps to a table in the database. Each property in the persistent CFC maps to a column in the table.

The following example explains these concepts by building a simple application, which would enable you to jumpstart with ColdFusion ORM. The example uses the cfartgallery data source that is shipped with ColdFusion. The cfartgallery data source has Artists and Art tables. Artists has a one-to-many relationship with the Art table.

Step 1:
Specify the ORM settings in the Application.cfc file.
The minimum required settings are mentioned in the following sample code snippet:
Application.cfc

```
<cfset this.name = "ArtGalleryApp">
<cfset this.ormenabled = "true">
<cfset this.datasource = "cfartgallery">
```

Apart from these, there are other settings that you can use to configure ORM. For details, see ORM settings.

> Define these setting only in Application.cfc and not in Application.cfm.

Step 2:
Map the ARTISTS.cfc to the database table.

1. Create the ARTISTS.cfc.
2. Flag it as a persistent CFC and map it to the ARTISTS table.To make the ARTISTS.cfc persistent, the `persistent` attribute should be set to `true` in the `cfcomponent` tag. The table attribute should be set to the table name. If table attribute is not specified, then the CFC name is taken as the table name. Each CFC can be given an entity name. Entity name is the name used by the ORM related functions to work with the persistent CFC. It can be specified by using the `entityname` attribute in `cfcomponent`. If `entityname` is not specified, then the CFC name is taken as the `entityname`.
3. Now, create properties in ARTISTS.cfc and map them to the columns in the table. One property should be created for each column in the table. To map the property to the column, the `column` attribute should be set to the corresponding column name. If the `column` attribute is not specified, then the name of the property is taken as the column name.
   For details on setting the ORM-specific attributes, see Define ORM mapping.
   The `ARTISTS.cfc`is defined as follows:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cfcomponent persistent="true">
 <cfproperty name="id" column = "ARTISTID" generator="increment">
 <cfproperty name="FIRSTNAME">
 <cfproperty name="LASTNAME">
 <cfproperty name="ADDRESS">
 <cfproperty name="CITY">
 <cfproperty name="STATE">
 <cfproperty name="POSTALCODE">
 <cfproperty name="EMAIL">
 <cfproperty name="PHONE">
 <cfproperty name="FAX">
 <cfproperty name="thepassword">
</cfcomponent>
```

Step 3:
Perform CRUD operations.
To retrieve data from the ARTISTS table, use `EntityLoad()`:

```
ARTISTS = EntityLoad("ARTISTS")
```

All the records from the ARTISTS table are retrieved as an object array.
To add a new artist, create a new artist object and call `EntitySave()`for this object.

```
<cfscript>
try {
 newArtistObj = EntityNew("artists");
 newArtistObj.setfirstname("John");
 newArtistObj.setlastname("Smith");
 newArtistObj.setaddress("5 Newport lane");
 newArtistObj.setcity("San Francisco");
 newArtistObj.setstate("CA");
 newArtistObj.setPostalCode("90012");
 newArtistObj.setphone("612-832-2343");
 newArtistObj.setfax("612-832-2344");
 newArtistObj.setemail("jsmith@company.com");
 newArtistObj.setThePassword("jsmith");
 EntitySave(newartistobj);
 ormflush();
} catch(Exception ex) {
 WriteOutput("<p>#ex.message#</p>");
}
</cfscript>
```

To update an existing record, load that object and make changes to it. ColdFusion automatically detects that the row for this object needs to be updated and it will get updated when `ORMFlush()`is called.

© 2014 Adobe Systems Incorporated. All rights reserved.

> **Note**
>
> ORMFlush() is called at the end of the request by default.

In the following code, the `newArtistObj`is already managed by ORM, so it does not need to be loaded again.

```
newArtistObj.setphone("612-832-1111");
ormflush();
```

To delete a record, `EntityDelete()`is used.

```
EntityDelete(newArtistObj);
ormflush();
```

Step 4:
Define Relationships
First define the mapping for the ART table to define a relationship between artwork and artists.
The ART.cfc is defined as follows:

```
<cfcomponent persistent="true">
 <cfproperty name="artid" generator="increment">
 <cfproperty name="artname">
 <cfproperty name="price">
 <cfproperty name="largeimage">
 <cfproperty name="mediaid">
 <cfproperty name="issold">
</cfcomponent>
```

In cfartgallery, the table ARTISTS has a one-to-many relationship with ART table, which are joined using the foreign key column ARTISTID. This means that each artist has created multiple artwork pieces and many artworks are created by one artist. To represent this in the object model, each ARTISTS object would contain an array of ART objects. Each ART object will contain a reference to its ARTISTS object. This is an example of a bidirectional relationship.
To achieve this, you need to add an extra property to the ARTISTS.cfc object that contains the array of ART objects for that ARTIST.

```
<cfproperty name="art" type="array" fieldtype="one-to-many" cfc="Art"
fkcolumn="ARTISTID">
```

`fieldtype="one-to-many"` specifies the type of relation.
`CFC="Art"` is used to convey that the relationship is with "ART" cfc.
`fkcolumn="artistid"` specifies the foreign key.

© 2014 Adobe Systems Incorporated. All rights reserved.

ART forms a many-to-one relationship with ARTISTS table because each piece of artwork is created by an artist and many other pieces of artwork are created by the same artist. To define this relationship, add a property in ART.cfc to define the relationship with ARTISTS.cfc.

```
<cfproperty name="artists" fieldtype="many-to-one" fkcolumn="artistid"
cfc="Artists" lazy="true">
```

`fieldtype="many-to-one"` specifies the type of relation.
`CFC="ARTISTS"` is used to convey that the relationship is with "ARTISTS" cfc.
`fkcolumn="ARTISTID"` specifies the foreign key.

Step 5:

Retrieve records in relationship

```
<cfscript>
 artist = EntityLoad("Artists", 1, true);
 arts = artist.getArts();
 WriteOutput("<b>" & artist.getid() & " " & artist.getfirstname() & " " &
 artist.getlastname() & "</b> has " & ArrayLen(arts) & " arts:<br>");
  if (ArrayLen(arts) > 0)
  {
   for(j = 1; j <= ArrayLen(arts); j ++)
   {
    art = arts[j];
    WriteOutput(art.getartname() & "<br>");
   }
  }
</cfscript>
```

#back to top

© 2014 Adobe Systems Incorporated. All rights reserved.

Architecture

In ColdFusion ORM, you need to define an object mapping to create persistent objects. The object mapping includes details such as:

- The table name for the object's class
- The column name that corresponds to each field in the object
- The join conditions for related objects

ColdFusion allows you to specify the mapping in CFCs. Such CFCs are called as persistent CFCs. Each persistent CFC usually maps to a table in the database. Each property in the CFC usually maps to a column in the table. Additional properties may be used to define relationships and other mapping details.
When ColdFusion creates the Hibernate configuration for the application, these persistent CFCs are used to automatically generate Hibernate mapping files, which have the extension ".hbmxml". For example, if ARTISTS.cfc is a persistent CFC, ColdFusion would automatically generate Artists.hbmxml. Hibernate mapping files contain the mapping information in XML format that Hibernate defines, to work with ColdFusion ORM. These Hibernate mapping files can be created manually.
For more information about creating Hibernate mappings manually, see Advanced mapping.
To use ColdFusion ORM, ColdFusion application must have `ormenabled` set to `true` in the `THIS` scope of Application.cfc. To define a persistent CFC, set `persistent="true"` in cfcomponent tag. An array of attributes are available in cfcomponent and `cfproperty` to specify mapping information.
For details, see Define ORM mapping.
When the application starts, ColdFusion first loads the Hibernate configuration file if it is specified in the application. The Hibernate configuration file contains various configuration parameters like including, dialect, cache settings, and mapping files that are required for the application. If a configuration file is not specified, ColdFusion ORM creates the Hibernate configuration using the default settings.
After loading the Hibernate configuration, all the mapping files (*.hbmxml) in the application folder and its mapped folders are loaded and added to the configuration.
ColdFusion then searches for persistent CFCs in the application folder and its mapped folders. If the hibernate mapping file is not present for any persistent CFC, ColdFusion generates it. If mapping information, such as primary key, foreign key, and column data type is missing in the persistent CFCs, ColdFusion automatically inspects the database and identifies the mapping.
ColdFusion then checks if DDL needs to be generated. This can be configured using the `dbcreate` option in the ORM settings. Based on the configuration option specified in `dbcreate`, tables are created or updated.The Hibernate SessionFactory is then built and made available to the application as long as the application is running. The SessionFactory is used to create Hibernate sessions that manage the persistent object lifecycle.
In ColdFusion, a Hibernate session starts when the first CRUD method is called and ends when the request ends or when the `ORMCloseSession()` method is called.
To improve performance, Hibernate batches all the Create/Update/Delete operations in the session and runs them when the session is flushed or only when necessary. Session Flush happens when the request ends or when the `ORMFlush()` method is called.
For transactions, a new session is always created at the start of a transaction and ends at the end of a transaction. Any previous open sessions are flushed and closed at the start of the transaction.
The Hibernate configuration is created and loaded only when the application starts. Therefore, any modifications to the mapping in the persistent CFCs or in the Hibernate mapping files are not loaded automatically. To load these modifications, you can either restart the application or call `ORMReload()`.
To restart the application, you can stop the application using `ApplicationStop()` and the next request to any page in this application automatically starts it.

#back to top

© 2014 Adobe Systems Incorporated. All rights reserved.

Configure ORM

The configuration for ORM is done in Application.cfc which makes this configuration application specific. For a ColdFusion application to use ORM, the following are the mandatory settings that need to be configured:

1. Enable ORM for the application. To do this, set the `ormenabled` property to true in the THIS scope of application.cfc
2. Provide the data source name by either setting data source property to `true` in the THIS scope of application or by defining it in ORM configuration for the application. Note that the data source should be configured on the server.
   The ORM configuration is specified using a struct called `ormsettings`, which is defined in the THIS scope of Application.cfc. The following table describes the settings for ORM that can be defined in Application.cfc.

Property Name	Description
`ormenabled`	Specifies whether ORM should be used for the ColdFusion application.Set the value to true to use ORM. The default is false.
`datasource`	Defines the data source that should be used by ORM.
`ormsettings`	The struct that defines all the ORM settings. For details, see ORM settings

© 2014 Adobe Systems Incorporated. All rights reserved.

Logging

Monitoring SQL queries that get generated and executed by ORM is critical for troubleshooting and performance optimization.
You can monitor and log the queries by:

- Defining `logsql` in `ormsettings`: This is a simple way to quickly enable SQL logging. The flag should be enabled in application.cfc:`<cfset this.ormsettings.logsql = "true">` This logs all the SQL queries that are generated by Hibernate to the console and server's output log file.
- Using `log4J.properties`: Hibernate uses log4j for its logging and you can completely control its logging including SQL by modifying the `log4j.properties`, which is present under `<CF_HOME>/lib`directory. Following is a sample snippet from the log4j.properties file:

```
###--------------- Hibernate Log Settings ------
### Set Hibernate log
log4j.logger.org.hibernate=ERROR, HIBERNATECONSOLE

### log just the SQL
#log4j.logger.org.hibernate.SQL=DEBUG, HIBERNATECONSOLE
#log4j.additivity.org.hibernate.SQL=false
### Also log the parameter binding to the prepared statements.
#log4j.logger.org.hibernate.type=DEBUG
### log schema export/update ###
log4j.logger.org.hibernate.tool.hbm2ddl=DEBUG, HIBERNATECONSOLE
### log cache activity ###
log4j.logger.org.hibernate.cache=ERROR, HIBERNATECONSOLE
# HibernateConsole is set to be a ColsoleAppender for Hibernate message  using
a PatternLayout.
log4j.appender.HIBERNATECONSOLE=org.apache.log4j.ConsoleAppender
log4j.appender.HIBERNATECONSOLE.layout=org.apache.log4j.PatternLayout
log4j.appender.HIBERNATECONSOLE.layout.ConversionPattern=%d{MM/dd HH:mm:ss}
[%t] HIBERNATE %-5p - %m%n%n
#---------------------------------------------
```

These settings control the SQLs that are generated for entity operations, how the data is bound to the statement while executing, what SQLs are generated for DDL, and what operations are performed on the secondary cache. All the logs get logged to console using HIBERNATECONSOLE which is actually a console appender. It can easily be changed to a FileAppender, which will then be logged to a log file. The configuration controls the logging for the following:

- SQL generated for entity operations
- Parameter binding for the prepared statements
- SQL generated for DDL
- Secondary cache operations

With the default settings, all the logs get logged to console. You can also direct the logging to a log file using the FileAppender provided by log4j. See log4j for more details on Appenders.

Log4j Properties	Description

© 2014 Adobe Systems Incorporated. All rights reserved.

`log4j.logger.org.hibernate.SQL`	This controls when and how the SQL will be logged. DEBUG says all the SQL will be logged
`log4j.logger.org.hibernate.type`	This logs the parameter binding to the prepared statement.
`log4j.logger.org.hibernate.tool.hbm2ddl`	Logs SQL for DDL i.e schema export.
`log4j.logger.org.hibernate.cache`	Logs secondary cache information.

#back to top

© 2014 Adobe Systems Incorporated. All rights reserved.

ORM settings

The following settings can be set in the `ormsettings` struct that ColdFusion uses to configure ORM. All these settings are optional. If you specify the value of any ORM setting to true or yes, then the setting is enabled, otherwise it is disabled.

Property Name	Default	Description
`autogenmap`	true	Specifies whether ColdFusion should automatically generate mapping for the persistent CFCs. If `autogenmap=false`, mapping should be provided in the form of .HBMXML files.
`automanageSession` Added in ColdFusion 9.0.1	true	Lets you specify if ColdFusion must manage Hibernate session automatically. • If enabled: ColdFusion manages the session completely. That is, it decides when to flush the session, when to clear the session, and when to close the session. • If disabled: The application is responsible for managing flushing, clearing, or closing of the session. The only exception is (in the case of transaction), when the transaction commits, the application flushes the session. ColdFusion closes the ORM session at the end of request irrespective of this flag being enabled or disabled.
`cacheconfig`		Specifies the location of the configuration file that should be used by the secondary cache provider.This setting is used only when `secondarycacheenabled=true`.See *Secondary level cache* in Cachingfor details.

© 2014 Adobe Systems Incorporated. All rights reserved.

`cacheprovider`	`ehcache`	Specifies the cache provider that should be used by ORM as secondary cache. The values can be: • Ehcache • JBossCache • Hashtable • SwarmCache • OSCache Fully qualified name of the class for any other cache provider.This setting is used only when `secondarycacheenabled=true`. See *Secondary level cache* in Caching for details.
`catalog`		Specifies the default Catalog that should be used by ORM.
`cfclocation`		Specifies the directory (or array of directories) that should be used by ColdFusion to search for persistent CFCs to generate the mapping. If `cfclocation` is set, ColdFusion looks at only the paths specified in it. If it is not set, ColdFusion looks at the application directory, its sub-directories, and its mapped directories to search for persistent CFCs.
`datasource`		Specifies the data source that should be used by ORM. If it is not specified here, then the data source specified for the application is picked up. Use the following convention to specify a datasource name: `this.datasource="<datasource_name>";`

© 2014 Adobe Systems Incorporated. All rights reserved.

`dbcreate`	none	ColdFusion ORM can automatically create the tables for your application in the database when ORM is initialized for the application. This can be enabled by using `dbcreate` in `ormsettings`. `dbCreate` takes the following values: • `update`: Setting this value creates the table if it does not exist or update the table if it exists. • `dropcreate`: Setting this value drops the table if it exists and then creates it. • `none` (default): Setting this value does not change anything in the database schema.

© 2014 Adobe Systems Incorporated. All rights reserved.

`dialect`		Specifies the dialect. ColdFusion supports the following dialects: • DB2 • DB2AS400 • DB2OS390 • Derby • PostgreSQL • MySQL • MySQLwithInnoDB • MySQLwithMyISAM • Oracle8i • Oracle9i • Oracle10g • Sybase • SybaseAnywhere • MicrosoftSQLServer • Informix Apart from these dialects, you can specify custom dialects by using the fully qualified class name. **Note** For Microsoft Access, dialect cannot be detected automatically. Use Microsoft SQL Server as the dialect for it.
`eventHandling`	false	Specifies whether ORM Event callbacks should be given. See Event Handling in CFC for details.
`flushatrequestend`	true	Specifies whether `ormflush` should be called automatically at request end. If `flushatrequestend` is false, `ormflush` is not called automatically at request end. See ORM session management.
`logSQL`	false	Specifies whether the SQL queries that are executed by ORM will be logged. If `LogSQL=true`, the SQL queries are logged to the console.

© 2014 Adobe Systems Incorporated. All rights reserved.

namingstrategy		Defines database standard and naming convention. See Naming strategy.
`ormconfig`		The Hibernate configuration file. This file contains various configuration parameters like, dialect, cache settings, and mapping files that are required for the application. The settings defined in the `ormsettings` override the settings defined in the Hibernate Configuration XML file.The connection information in the Hibernate Configuration XML file is however ignored because ColdFusion uses its own connection pool.You will need to use this only when you need to use a hibernate setting that is not available using `ormsetting`.
`savemapping`	false	Specifies whether the generated Hibernate mapping file has to be saved to disc. If you set the value to true, the Hibernate mapping XML file is saved with the filename "CFC name".hbmxml in the same directory as the CFC. If any value of `savemapping` is specified in CFC, it will override the value specified in the ormsetting.
`schema`		Specifies the default Schema that should be used by ORM.
`secondarycacheenabled`	false	Specifies whether secondary caching should be enabled. See *Use secondary cache* in Caching for details.
`skipCFCWithError` Added in ColdFusion 9.0.1	false	Lets you specify if ColdFusion must skip the CFCs that have errors. If set to `true`, ColdFusion ignores the CFCs that have errors.

© 2014 Adobe Systems Incorporated. All rights reserved.

sqlscript		Path to the SQL script file that gets executed after ORM is initialized. This applies if dbcreate is set to dropcreate. This must be the absolute file path or the path relative to the application.The SQL script file lets you populate the tables before the application is accessed.
useDBForMapping	true	Specifies whether the database has to be inspected to identify the missing information required to generate the Hibernate mapping. The database is inspected to get the column data type, primary key and foreign key information.

Sample Application.cfc

```
<cfset this.name = "ArtGallery">
 <cfset this.ormenabled = "true">
 <cfset this.ormsettings={datasource="cfartgallery", logsql="true"}>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

Define ORM mapping

The ORM mapping can be defined either in the CFC or in a separate Hibernate mapping file (.hbmxml). See Advanced mapping for details on Hibernate mapping file. The ORM mapping is defined in the CFC using ORM-specific attributes on cfcomponent and cfproperty tag.
Following example shows a CFC (ARTIST.cfc) with mapping information:

```
<cfcomponent persistent="true" entityname="Artist" table="Artists">
 <cfproperty name="id" column="ARTISTID" generator="increment">
 <cfproperty name="firstname">
 <cfproperty name="lastname">
 <cfproperty name="address">
 <cfproperty name="city">
 <cfproperty name="state">
 <cfproperty name="postalcode">
 <cfproperty name="email">
 <cfproperty name="phone">
 <cfproperty name="fax">
 <cfproperty name="thepassword">
</cfcomponent>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

Map a ColdFusion component

The tag `cfcomponent` defines a persistent CFC by setting `persistent="true"`. It allows you to specify various other mappings for a persistent CFC.
If your persistent CFC has `init` method, ensure that either it has no arguments or all arguments are optional.

Syntax

```
<cfcomponent
 accessors= "yes|no"
 persistent="true|[false]"
 entityname="entity_name"
 table="table_name"
 schema="schema"
 catalog="catalog"
 dynamicInsert="true|[false]"
 dynamicupdate="true|[false]"
 readonly="true|[false]"
 selectbeforeupdate="true|[false]"
 discriminatorvalue="discriminator_value"
 discriminatorcolumn="discriminator_column"
 joincolumn="join_column"
 cacheuse="read-only|read-write|nonstrict-read-write|transactional|[none]"
 cachename="cache_name"
 batchsize="batchsize"
 optimisticLock="none|[version]|dirty|all"
 lazy="[true]|false"
 savemapping="true|false">
```

Attributes

Attribute	Req/Opt	Default	Description
`accessors`	true		If set to false, ColdFusion ORM does not generate the implicit getters and setters.
`batchsize`	Optional	1	An integer value that specifies the number of records to be retrieved at a single instance. For details, see Lazy Loading.
`cachename`	optional		Use this value to specify the name of the secondary cache. See Caching for details.

© 2014 Adobe Systems Incorporated. All rights reserved.

`cacheuse`	optional	none	Use this value to specify the caching strategy to be used for caching this component's data in the secondary cache. See Caching for details.
`catalog`	Optional		Used to specify the database catalog name.
`discriminatorcolumn`	optional		Use this attribute to define the discriminator column to be used in inheritance mapping. See *Inheritance mapping* in Advanced mapping for details.
`discriminatorvalue`	optional		Use this attribute to define the discriminator value to be used in inheritance mapping. See *Inheritance mapping* in Advanced mapping for details.
`dynamicInsert`	Optional	false	Whether INSERT SQL is to be generated at runtime: • true • false Only those columns whose values are not null are included in the SQL.
`dynamicupdate`	Optional	false	Whether UPDATE SQL is to be generated at runtime: • true • false Only those columns that have changed values are included in the SQL.

© 2014 Adobe Systems Incorporated. All rights reserved.

`entityname`	Optional	Name of the CFC	Specifies the name of the entity. Entity name is the name used by the ORM related functions to work with the persistent CFC. If `entityname` is not specified, then the CFC name is taken as the `entityname`. The `entityname` must be unique in the application.
`joincolumn`	optional		Use this attribute to define the join column for inheritance mapping. See Inheritance mapping for details.
`lazy`	Optional	true	Whether loading is to be done lazily: • true • false For details, see Lazy Loading.
`optimisticLock`	Optional	version	Determines the locking strategy. It can be any one of the following four values: • none • version • dirty • all For details, see Optimistic locking.
`persistent`	Required	false	Whether the CFC is a persistent CFC: • true • false
`readonly`	Optional	false	Whether the table is read-only: • true • false This is inserted by the `EntitySave()` method.

© 2014 Adobe Systems Incorporated. All rights reserved.

`savemapping`	Optional		Saves the generated Hibernate mapping file to the disk.If you have set `savemapping` for the application, then the value you set here overrides it.
`schema`	Optional		Used to specify the schema name.
`selectbeforeupdate`	Optional	false	Whether Hibernate should never perform an SQL UPDATE unless it is certain that an object is actually modified: • true • false In cases when a transient object is associated with a new session using update(), Hibernate performs an extra SQL SELECT to determine if an UPDATE is actually required.
`table`	Optional	Name of the CFC	Specifies the name of the database table to which the CFC maps.

© 2014 Adobe Systems Incorporated. All rights reserved.

Map the properties

The `cfproperty` tag is used to define:

- Primary or composite key
- Columns
- Relationships
- Versioning

The following table lists the common attributes that are used to define these mappings:

Attribute	Req/Optional	Default	Description
`default`	optional		This sets the default value on the property when the object is created.

© 2014 Adobe Systems Incorporated. All rights reserved.

`fieldtype`	optional	column	This attribute is used to specify the type of the property. Use this attribute to specify the following: • Primary key: Specify `fieldtype = "id"` to map a property to the primary key in the table. For details, see Primary key below. • Column: Specify the `fieldtype = "column"` to map a property to a column in the table. For details, see Column below. • Relationship: Specify the `fieldtype = "`*`relationship_type`*`"`. The_ relationship_type_ can be `one-to-one`, `one-to-many`, `many-to-one`, or `many-to-many`. For details, see Define Relationships. • Version: Specify the `fieldtype="version"` to indicate that the column contains versioned data. For details, see Version below. • Timestamp: Specify the `fieldtype="timestamp"` to indicate the column contains data with timestamp. For details, see Time stamp below. • Collection: Specify the `fieldtype="collection"` to define the collection mapping. For details, see Collection Mapping
`name`	required		Specifies the name of the property.

© 2014 Adobe Systems Incorporated. All rights reserved.

`type`	optional		Specifies the ColdFusion data type for the property.

`persistent`	optional	true	Specifies whether this property should be persisted. If `persistent="false"` all the ORM related attributes are ignored.Set this attribute to false if you do not want this property to be persisted by ORM. For example:
			
			`<cfcomponent persistent=true>`
			
			`<cfproperty name="a">`
			
			`<cfproperty name="b" persistent="true">`
			
			`<cfproperty name="c" persistent="false">`
			`</cfcomponent>`
			
			When an object of this CFC is persisted, the properties a and b would be persisted in the database but the property c would not be persisted.

© 2014 Adobe Systems Incorporated. All rights reserved.

remotingFetch	optional	true	If remotingFetch is false, then the value of that attribute is not sent over flash remoting. The attribute is true by default for all properties. However, for ORM CFCs where persistent = true, the value of the remotingFetch attribute is set to false, by default, for one-to-one, one-to-many, many-to-one, or many-to-many relationships.

Primary key

Simple primary key

In a relational database, a primary key is defined as a key that uniquely identifies a row in a table. Typically, a table has one primary key that represents a single column of information in the table.
To indicate that a cfproperty maps to a primary key in the table, set the attribute fieldtype="id".

Syntax

```
<cfproperty
 name="property_name"
 fieldType="id"
 ormtype="type"
 column="column_name"
 generator="increment|identity
|sequence|sequence-identity|seqhilo
|uuid|guid|native|[assigned]|select|foreign"
 params="{key1=val1,key2=val2...}"
 sqltype="sql_type"
 length="column_length"
 unsavedvalue="instantiated_instance">
```

Example

An example to define an assigned primary key:

```
<cfproperty name="artistid" fieldtype="id" column="ARTISTID" generator="assigned">
```

An example to define a auto-generated primary key using increment generator:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cfproperty name="artid" fieldtype="id" column="ARTID" generator="increment">
```

An example to define a auto-generated primary key using a generator, which requires additional parameters:

```
<cfproperty name="id" fieldtype="id" column="ID" generator="sequence"
params="{sequence='id_sequence'}>
```

Attributes

Attribute	Req/Opt	Default	Description
`column`	Optional	The value of the name attribute	Used to specify the primary key column name.
`fieldType`	Optional		Should be "id" for primary key. If `fieldtype` is not specified and the `useDBForMapping=true`, then the `fieldtype` is determined by inspecting the database.
`generator`	Optional	assigned	Algorithm used to generate unique identifiers for instances of the persistent CFC. See Generators for details.
`length`	optional		Use this attribute to specify the length of the column. This attribute is used for table creation only.
`name`	Required		Name of the property

© 2014 Adobe Systems Incorporated. All rights reserved.

`ormtype`	Optional	String	Used to specify the data type of the primary key. If data type is not set and the ORM setting `useDBFormapping=true`, then the `ormtype` is determined by inspecting the database. The different data types that are supported by ColdFusion are specified in the ORM data types.
`params`			Additional parameters required by the various generators to generate the ID.The value for the `params` attribute should follow the CF Struct syntax. For example: `params = {key1='value1', key2='value2'}`See Generators for details.
`selectkey`	optional		The column name that is used to retrieve the primary key generated by a database trigger.
`sqltype`	optional		Use this attribute to specify the DB-specific SQLType that should be used for the column. This attribute is used for table creation only. If this attribute is not specified, ColdFusion will automatically decide the SQLType based on the `ormtype` specified for this property.
`unSavedValue`	optional		An identifier property value that indicates that an instance is newly instantiated and not saved/loaded in the database

Composite Key

© 2014 Adobe Systems Incorporated. All rights reserved.

If the primary key consists of more than one column, it is called as a composite key. A composite key can be specified by using `fieldtype='id'` on all the properties that form the primary key.

Example

If in a table, the columns Order_id and Product_id form a composite key, then, they should be defined as:

```
<cfproperty name="Order_Id" fieldtype="id" column="Order_Id">
<cfproperty name="Product_Id" fieldtype="id" column="Product_Id">
```

For a composite key, the generator should always be "`assigned`".

Generators

Generators are algorithms that are used to generate unique identifiers for instances of the persistent CFC. Generators can have any one of the following values:

- `increment`: This algorithm generates identifiers of type `long`, `short`, or `int` by incrementing a counter maintained by ORM. This is commonly used when auto-generation for the primary key is not enabled in the table and you want ORM to generate the primary key. This should be used when a single instance of ColdFusion is the only process to insert data into the table.
- `identity`: For databases such as DB2, MySQL, Sybase, and MS SQL, which support identity columns, you can use identity key generation. The key is automatically generated by the database and the object is populated with the generated key. This strategy requires ORM to execute two SQL queries to insert a new object.
- `sequence`: For databases such as DB2, Oracle, PostgreSQL, Interbase, McKoi, and SAP, which support sequences, you can use sequence style key generation. The key is automatically generated by the database and the object is populated with the generated key. This strategy requires ORM to execute two SQL queries to insert a new object. This generator includes the sequence parameter, which needs to be specified in the p `aramsattribute`.For example:

  ```
  <cfproperty name="id" fieldtype="id" generator="sequence"
  params="{sequence='id_sequence'}">
  ```

- `native`: This algorithm is commonly used to automatically generate primary keys. This generator picks identity or sequence depending upon the capabilities of the underlying database.
- `assigned`: This algorithm is used when the application needs to assign its own identifier to the object. It is the default generator used by ColdFusion.
- `foreign`: This is used with a `<one-to-one>` primary key association. In this case, the primary key is the same as the primary key of the associated object. This generator would need the property parameter, which needs to be specified in the `params` attribute. The value of the `param` property should be the name of the relationship property. See One-to-one relationships for details.
- `seqhilo`: See www.hibernate.org/5.html.
- `uuid`: See www.hibernate.org/5.html.
- `guid`: See www.hibernate.org/5.html.
- `select`: See www.hibernate.org/5.html.
- `sequence-identity`: See www.hibernate.org/5.html.

Column

To indicate that a `cfproperty` maps to a column in the table, specify `fieldtype="column"`. If the `fieldtype` is

© 2014 Adobe Systems Incorporated. All rights reserved.

not specified for `cfproperty`, it is mapped as a column property.

Syntax

```
<cfproperty
 name="Property name"
 fieldtype="column"
 column="column_name"
 persistent="true|false"
 formula="SQL expression"
 ormtype="ormtype"
 update="[true]|false"
 insert="[true]|false"
 optimisticLock="[true]|false"
 generated="true|[false]"
 length="column_length"
 precision="precision"
 scale="scale"
 index="index_name"
 unique = "true|[false]"
 uniquekey="uniquekey_name"
 notnull="true|[false]"
 dbdefault="default_col_value"
 sqltype="sql_type">
```

Example

To specify a simple property:

```
<cfproperty name="FIRSTNAME"/>
```

To specify a property which has a different name than that of the column name:

```
<cfproperty name="LNAME" column="LASTNAME"/>
```

To specify a property which should be read-only:

```
<cfproperty name="password" column="THEPASSWORD" insert="false" update="false">
```

Attributes

Attribute	Req/Opt	Default	Description
`column`	optional	Name of the property	Name of the column.

© 2014 Adobe Systems Incorporated. All rights reserved.

`dbdefault`			This sets the default value of a column in the table when schema is exported.	
`fieldType`	optional	column	Should be "column" for column property.	
`formula`	optional		SQL expression that defines the value of the property. If you specify a formula, the column value is ignored. See Computed property below.	
`generated`	optional	never	Specifies that this property value is actually generated by the {{database {never|always	insert}}} • `never`: Specifies that the value for this property is never generated by database • `always`: Specifies that the value for this property is always generated by database • `insert`: Specifies that the value for this property is generated at the time of insert but is not regenerated at updates.
`insert`	optional	true	Specifies whether the column should be included in SQL UPDATE and/or INSERT statements:{true/false}Set `update=false` and `insert=false` if the column needs to be read-only.	

© 2014 Adobe Systems Incorporated. All rights reserved.

`name`	Required		Name of the property. If this attribute is not specified, the name of the property is used as the column name.
`optimisticlock`	optional	true	Whether updates to this property require acquisition of the optimistic lock on the table row:{true/false}
`ormtype`	optional	string	Specifies the data type.If specified, then • If ORM setting `useDBForMapping` is set to `true` and the table exists, then `ormtype` is obtained by inspecting the table • (Otherewise) Type, if specified, is used as `ormtype`
`update`	optional	true	Specifies whether the column should be included in SQL update statement: {true/false}Set `update=false` and `insert=false` if the column needs to be read-only.

DDL-only attributes

The following attributes are used only when DDL generation is required and not used for runtime.

Attribute	Req/Opt	Default	Description
`dbdefault`	optional		Specifies the default value of the column in the table.

© 2014 Adobe Systems Incorporated. All rights reserved.

`index`	optional		Specifies the name of an index that is created using the mapped column.
`length`	optional		Specifies the length value.
`notnull`	optional	false	A Boolean value that specifies whether a notnull constraint should be added for this column.
`precision`	optional		Specifies the precision value.
`scale`	optional		Specifies the scale value.

`sqltype`	optional		This allows user to override the default mapping of `ormtype` to SQL datatype. `sqltype` is used as the DB specific SQL type for a column when creating the table. If this attribute is not specified, ColdFusion will automatically decide the `sqltype` based on the `ormtype` specified for this property.For example:
			 `<cfProperty name="active" ormtype="char" sqltype="bit">`
			 `<cfProperty name="balance" ormtype="float" sqltype="decimal(13, 3)">`

© 2014 Adobe Systems Incorporated. All rights reserved.

`unique`	optional		Specifies if there should be a unique constraint on the column.
`uniquekey`	optional		Groups columns in a single unique key constraint.

Computed property

Computed property is a property whose value does not come from a column but is computed using a SQL query. Use formula attribute to specify the SQL to be used to retrieve the value for this property.

```
<cfcomponent persistent="true" table="ARTISTS" schema="APP">
 <cfproperty name="ID" column="ARTISTID" fieldtype="id"/>
 <cfproperty name="FIRSTNAME"/>
 <cfproperty name="LASTNAME"/>
 <cfproperty name="NumberOfArts" formula="select count(*) from Art art where
      art.ArtistID=ArtistID"/>
</cfcomponent>
```

Versioning

Versioning is a technique that allows you to implement concurrency control fora component. You can specify either `version` or `timestamp` property for a component.
For details, see Optimistic locking.

> **Note**
>
> A component can have only one versioning property, either timestamp or version. If you specify multiple versioning properties, such as two timestamps, or two versions, or a timestamp and a version, an error is thrown.

Version

Use the version attribute to indicate that the column contains versioned data. The version attribute is useful for long transactions.

Syntax

```
<cfproperty
name="fieldname"
fieldtype="version"
column="column name"
ormtype="type"
generated="true|[false]"
insert="[true]|false">
```

© 2014 Adobe Systems Incorporated. All rights reserved.

Example

To create a simple version property:

```
<cfproperty name="version" fieldtype="version">
```

Attribute

Attribute	Req/Opt	Default	Description
`column`	Optional		The name of the column that contains versioned data
`fieldtype`	Required		Should be "version" for primary key.
`generated`	Optional	never	Specifies if the versioned field is generated by the database. The values are `"never"` and `"always"`.
`insert`	Optional		Specifies if the versioned field should be included in the SQL INSERT statement.
`name`	Required		Name of the property.
`ormtype`	Optional	`int`	The data type can be any one of the following:integerlongshort

Time stamp

Use the `timestamp` attribute to indicate that the column contains time-stamped data. Use the `timestamp` attribute as an alternative to the `version` attribute.

Syntax

```
<cfproperty
name="fieldname"
fieldtype="timestamp"
column="column name"
generated="true|[false]"
source="[vm]|db">
```

© 2014 Adobe Systems Incorporated. All rights reserved.

Attribute	Req/Opt	Default	Description
`column`	Optional		The name of the column that contains time-stamped data.
`fieldtype`	Required		Specifies the field type.Specify the field type value as `timestamp` for a time-stamped field.
`generated`	Optional	`false`	Specifies if the timestamp field is generated by the database. You can select from the following values: `falsetrue`
`name`	Required		Name of the property.
`source`	Optional	`vm`	Specifies the source from where the timestamp has to be retrieved. You can select from the following values:`dbvm`

ORM data types

You can use any of the following ORM data types for CFCs:

- `string`
- `character`
- `char`
- `short`
- `integer`
- `int`
- `long`
- `big_decimal`
- `float`
- `double`
- `Boolean`
- `yes_no`
- `true_false`
- `text`
- `date`
- `timestamp`
- `binary`
- `serializable`
- `blob`
- `clob`

Escaping SQL keywords in table and column name

© 2014 Adobe Systems Incorporated. All rights reserved.

ColdFusion automatically escapes the table name or column name if it is an SQL keyword or if there is a space in it. The list of SQL keywords are present in `<CF_HOME>/lib/sqlkeywords.properties` file. This file contains standard ANSI SQL keywords and some database-specific keywords. You can modify this file to include any other SQL keyword that is missing. In case you are adding SQL keyword for a database other than the ones specified in this file, you should also add it to the 'ANSI' list so that ColdFusion can use it.

© 2014 Adobe Systems Incorporated. All rights reserved.

Define Relationships

Relationship is the most crucial aspect of ORM. In a relational database, relation between tables are defined using foreign key. However, in case of objects, relation between two objects is defined using association where one object has a reference to another. ORM defines how the object relation is mapped to the database relation.
In this topic, relation and association would be used interchangeably.
Before you learn how to define the mapping for relation, it is important to understand few key concepts:

- Source object: Object having the reference to the related object is termed as source of the relationship.
- Target object: Object being referred or associated is termed as target of the relationship.
- Direction and Navigability: In relational database, the relationship is always unidirectional, which implies that you can navigate from one table to another but not back to the same table. However, object model can be either unidirectional or bidirectional. A unidirectional association means that source has the reference to the target but the target does not know about the source. A bidirectional association means that both the objects have reference to each other and you can navigate from either object to another. In other words, source has a reference to the target and target also has a reference to the source. This also means that both the objects are source and target at the same time.To set the association between objects, you need to set the references appropriately. For example, in case of Person-Address relation, where one person as one address, you need to associate Address to person as:

  ```
  person.setAddress(address);
  ```

  At this point, person object knows about the Address object but the address object does not know the person object. So, this is a unidirectional relation between Person-Address. To make this bidirectional, you need to associate Person to Address as:

  ```
  address.setPerson(person);
  ```

- Multiplicity: This defines how many target entities can a particular source have and how many source entities can a particular target have. Consider the example of artwork and artist, where an artist has many artwork pieces. In an object model, an artwork has reference to one artist and an artist has reference to many pieces of artwork. So, for artwork and artist the multiplicity is many-to-one and for artist and artwork, it is one-to-many. The other two type of multiplicities are one-to-one and many-to-many. In this topic, multiplicity would be referred to as the type of relationship.
  To indicate that a property defines the relationship between two persistent components, as a result of relationship in the database table, specify the `fieldtype` in the `cfproperty` tag to one of the following:
- one-to-one
- one-to-many
- many-to-one
- many-to-many
  You can also use the link table to establish a relationship. A link table contains the foreign key to both tables that participate in the relationship. ORM looks for the map key column using the link table and not the target table.

Relationship attributes

This table specifies the attribute details for all the relationship types.
The "Applies to" column indicates the relationship type that the attribute is applicable to; "all" indicates that the attribute is applicable to all relationship types.

© 2014 Adobe Systems Incorporated. All rights reserved.

Attribute	Applies to	Re/Opt	Default	Description
`batchsize`	one-to-manymany-to-many	Optional		An integer value that specifies the "`batchsize`" for fetching uninitialized collections. For details, see *Batch fetching* in Lazy Loading.
`cacheuse`	one-to-many many-to-many	optional		Use this value to specify the caching strategy to be used for caching this component's data in the secondary cache.See Caching for details.
`cachename`	one-to-manymany-to-many	optional	<entityname> <relationname>	Use this value to specify the name of the secondary cache.See Caching for details.
`cascade`	all	optional		See the Cascade options section for details.
`cfc`	all	Required		Name of the associated CFC.
`constrained`	one-to-one	Optional	false	Whether a constraint is set on this table's Primary Key column to reference the Primary Key in the other table:truefalseSee One-to-one relationships below for details.

© 2014 Adobe Systems Incorporated. All rights reserved.

`fetch`	all	Optional	select	Specifies whether join query or sequential select query will be used to retrieve the associated objects. The values are: join selectSee Lazy Loading for details.
`fieldtype`	all	Required	column	Specifies the type of relationship mapping:one-to-one one-to-manymany-to-onemany-to-many

© 2014 Adobe Systems Incorporated. All rights reserved.

`fkcolumn`	all	Optional		Specifies the foreign key column.In case the relation is established using link table, this specifies the foreign key column in the link table that references the primary key of the source object.If the relationship is established using multiple foreign key columns (that reference the composite key of the source table), then you must use comma-separated column names.Also, the order in which the column names are specified must match the order of composite keys defined. If you do not specify any values, then • If the table exists and the constraints are defined, ColdFusion automatically selects the values from the table • If the table does not exist, ColdFusion auto-generates the values
`foreignkeyname`	one-to-one many-to-one, many-to-many	optional	autogenerated	Specifies the name of the foreign key constraint. This is used only when the tables are created by ORM.

© 2014 Adobe Systems Incorporated. All rights reserved.

`index`	many-to-one	optional	false	Specifies the name of the index for the foreign key column.
`insert`	many-to-one	Optional	true	Specifies whether the column should be included in SQL UPDATE and/or INSERT statements. Values are:`true` `false}}Set` `{{update=false` and `insert=false` if the column needs to be read-only.
`inverse`	one-to-manymany-to-many	Optional	false	Specifies whether SQL query should be executed for this association when persisting this object. Value are: truefalse See "Inverse" section for details.

© 2014 Adobe Systems Incorporated. All rights reserved.

`inversejoincolumn`	all	Optional		Specifies the foreign key column in the Join table that references the primary key column in the target table. In case of a composite key, you can use a comma-separated list of column names. If the join table has multiple foreign key columns (that reference the composite key of the target table), then you must use comma-separated column names.Also, the order in which the column names are specified must match the order of composite keys defined.If you do not specify any values, then • If the table exists and the constraints are defined, then ColdFusion automatically selects the values from the table • If the table does not exist, then ColdFusion auto-generates the values
`lazy`	all	Optional	true	Specifies if the association should be loaded lazily. truefalseextraSee Lazy Loading for details.
`linkcatalog`	all	Optional		Catalog for the link table.

© 2014 Adobe Systems Incorporated. All rights reserved.

`linkschema`	all	Optional		Schema for the link table.
`linktable`	all	Required		Name of the link table.
`mappedby`	all	Optional		In a relationship, the foreign key can reference a unique column other than the primary key. In such cases, use `mappedby` to specify the property name that maps to the unique key column.
`missingrowIgnored`	many-to-one, many-to-many, (in ColdFusion 9.0.1) one-to-one	Optional	false	Values are: truefalseIf the value is true, and the row that is referenced by the foreign key is missing, it is treated as a null association.The default is false, in which case an exception is thrown.
`name`	all	Required		Name of the field
`notnull`	many-to-one	optional	false	Use this to add the not-null constraint for the foreign key column when the table is created by ORM.
`optimisticlock`	all	Optional	true	Whether updates to this property need acquisition of the optimistic lock on the table row. Values are:truefalseSee Optimistic locking for details.

© 2014 Adobe Systems Incorporated. All rights reserved.

`orderby`	one-to-manymany-to-many	Optional		Specifies the `order by` string that needs to be used to sort the associated collection. Use the following format to specify this string: `"col1 <asc/desc> (, col2< asc/desc>) " or "col1 (, col2) "` In the latter case, `asc` is taken as default.
`readonly`	one-to-manymany-to-many	Optional	false	Values are: true|false If set to true, it indicates that the collection never changes and can be cached.
`remotingFetch`	all	Optional	false	The value of the remotingFetch attribute is false by default for any property that shares one-to-one, one-to-many, many-to-one, or many-to-many relationship. Set this value to true to retrieve data on the client-side.

© 2014 Adobe Systems Incorporated. All rights reserved.

`singularname`	one-to-manymany-to-many	optional	property name	Use this property to define the custom name for generated relationship methods. See Generated methods for relationships between CFCs.
`structkeycolumn`	one-to-manymany-to-manytype=struct			The column in the target table to use as key if the collection type is struct.
`structkeytype`	one-to-manymany-to-many type =struct	Optional		Specifies the data type of the key, when type=struct. For the entire list of data types, see the Data Types section.
`type`	one-to-manymany-to-many	Optional		Specifies the datatype of the relationship property:array struct
`update`	many-to-one	Optional	true	Specifies whether the column should be included in SQL update statement:truefalse Set update=false and insert=false if the column needs to be read-only.
`unique`	many-to-one	optional	false	Use this to add the unique constraint for the foreign key column when the table is created by ORM. This effectively makes this relation one-to-one.
`uniquekey`	many-to-one	optional		Groups columns in a single unique key constraint.

© 2014 Adobe Systems Incorporated. All rights reserved.

where	one-to-manymany-to-many	optional		Use this to specify a SQL that will be used to filter the collection retrieved. See "Applying filters on relationship" for details.

Cascade options

In an association, it is cumbersome to apply an action performed on object to the other object. For example, in case of an Department-Employee one-to-many association, if you add an employee, the same change needs to be updated on the Department as well. The cascade option in Hibernate allows you to perform such operations. You can specify the following values in the cascade attribute:

- `all`: Allows you to apply all operations to be cascaded to the associated object.
- `save-update`: If the parent object is saved, the associated objects are saved as well.
- `delete`: Deletes the child object if the delete operation is called on the parent object.
- `delete-orphan`: This is a special kind of cascade option that applies to one-to-many relation only. Deletes all child objects for which the association has been removed.
- `all-delete-orphan`: Cascades all operations to child objects, and performs delete-orphan action.
- `refresh`: Cascades the refresh action to the child object. The refresh action is used to reload an object and its collections.
  Typically, `cascade` attribute is not used on a many-to-one or a many-to-many relationship.You can also specify comma separated cascade values in the `cascade` attribute. For a one-to-one or a one-to-many relationship, the most common values are all-delete-orphan.For an association where the child object can continue to exist even if the parent object is deleted, you can apply the save-update cascade value.

Applying filter on associated objects

In one-to-many and many-to-many relationships, an array or struct is retrieved. Filters can be applied to retrieve a subset of the associated objects. The filter can be specified in the `where` attribute, which is an SQL where clause. In a one-to-many association for artist and artwork:If you want to retrieve only the unsold artwork for every Artist object, you need to define the mapping as follows:

```
<cfproperty name="unsoldArts" cfc="Art" fieldtype="one-to-many" fkcolumn="ARTISTID"
where="issold=0">
```

Inverse

In a bidirectional relationship, the `inverse` attribute is used on an association property to specify whether an SQL query should be executed for the association, when persisting the object.Consider the ART and ARTIST CFCs, which have a bidirectional one-to-many relationship. This means that each ART object has a reference to the ARTIST object and ARTIST object has a reference to the ART object. While persisting ARTIST and the associated ART, the relationship can be established in the database from both sides of the object. Setting `inverse=true` on one side of the relation tells ORM to ignore this side of relation for executing the SQL. As a general rule, in a bidirectional relation, one side must set `inverse` to `true`. For one-to-many or many-to-one relation, `inverse` shou ld be set on the many side of the relation. For example, in ARTIST-ART relation, inverse should be set to `true` on the '`art`' property in ARTIST. In many-to-many relation, you can set `inverse=true` on any side of the relation.

One-to-one relationships

© 2014 Adobe Systems Incorporated. All rights reserved.

A one-to-one relationship is where the source object has an attribute that references another single target object and vice-versa. An example of this relationship is the relationship between an employee and the assigned office cubicle, where one employee has one office cubicle and one office cubicle belongs to only one employee.
A one-to-one relationship between two persistent components are defined using `fieldtype` value one-to-one.
Syntax:

```
<cfproperty name="fieldname"
fieldtype="one-to-one"
cfc="Referenced_CFC_Name"
linktable="Link table name"
linkcatalog="Catalog for the link table"
linkschema="Schema for the link table"
fkcolumn="Foreign Key column name"
inversejoincolumn="Column name or comma-separated list of primary key columns"
cascade="cascade_options"
constrained="true|[false]"
fetch="join|[select]"
lazy="[true]|false">
```

There are two types of one-to-one relationships:

- Primary key association
- Unique Foreign Key association

Primary key association

In this type of association, the primary key of one table references to the primary key of another table. That is, both the tables share the same primary key. The following example shows how to define this mapping.

Example

Consider the EMPLOYEE and OFFICECUBICLE example. Both the tables share the same primary key. The mapping for these tables are as follows:

- EMPLOYEE.cfc

```
<cfcomponent persistent="true" table="Employee">
 <cfproperty name="id" fieldtype="id" generator="native">
 <cfproperty name="firstname">
 <cfproperty name="lastname">
 <cfproperty name="officecubicle" fieldtype="one-to-one" cfc="OfficeCubicle">
</cfcomponent>
```

- OFFICECUBICLE.cfc

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cfcomponent persistent="true" table="OfficeCubicle">
 <cfproperty name="id" fieldtype="id" generator="foreign"
params="{property='Employee'}" ormtype="int">
 <cfproperty name="Employee" fieldtype="one-to-one" cfc="Employee"
constrained="true">
 <cfproperty name="Location">
 <cfproperty name="Size">
</cfcomponent>
```

`fieldtype=one-to-one` specifies that the property is a one-to-one property.`constrained=true` on `Employee` property in OFFICECUBICLE.cfc, means that a constraint is set on the OFFICECUBICLE table for its ID to reference the ID in the EMPLOYEE table.

The ID of EMPLOYEE table is auto-generated. The ID of the OFFICECUBICLE table should be the same as the ID of the Employee table. For this, set `generator="foreign"`. Foreign generator takes one parameter `'property'` as input, which should be the relationship property name of OFFICECUBICLE entity which is 'EMPLOYEE' in this case.Here, primary key values of related rows in both the tables must be the same. The identity generator algorithm in the mapping for the component (whose mapped table has the constraint), must be set to foreign.

Unique foreign key association

In this type of association, the foreign key of one table references the primary key of another table, and the foreign key column has a unique constraint. To define this relationship, `fkcolumn` attribute should be specified on the relationship-property of the CFC whose table contains the foreign key column. The other end of relation should use `mappedby` attribute.Syntax

```
<cfproperty
name="fieldname"
fieldtype="one-to-one"
cfc="Referenced_CFC_Name"
linktable="Link table name"
linkcatalog="Catalog for the link table"
linkschema="Schema for the link table"
fkcolumn="Foreign Key column name"
inversejoincolumn="Column name or comma-separated list of primary key columns"
mappedby="Mapped_Field_name_in_referenced_CFC"
cascade="none"
fetch="join|[select]"
lazy="[true]|false">
```

Note

The mappedby attribute can not be specified with the fkcolumn attribute.

Example

In the EMPLOYEE and OFFICECUBICLE example, OFFICECUBICLE has a foreign key column, EMPLOYEEID. This foreign key references the primary key of the Employee table. OFFICECUBICLE has an auto-generated primary key, which does not participate in the relationship.EMPLOYEE.cfc

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cfcomponent persistent="true" table="Employee">
 <cfproperty name="EmployeeID" fieldtype="id" generator="native">
 <cfproperty name="firstname">
 <cfproperty name="lastname">
 <cfproperty name="officecubicle" fieldtype="one-to-one" cfc="officecubicle"
mappedby="Employee">
</cfcomponent>
```

OFFICECUBICLE.cfc

```
<cfcomponent persistent="true" table="officecubicle">
 <cfproperty name="id" fieldtype="id" generator="native">
 <cfproperty name="Employee" fieldtype="one-to-one" cfc="Employee"
fkcolumn="EmployeeID">
 <cfproperty name="Location">
 <cfproperty name="Size">
</cfcomponent>
```

- In OFFICECUBICLE entity, `fkcolumn="EmployeeID"` specifies that EmployeeID is the foreign key column in OFFICECUBICLE table.
- `mappedby="Employee"` specifies that the one-to-one relationship is with the foreign-key property 'EMPLOYEE' in OFFICECUBICLE entity and not with its primary key.
- In Employee entity, fkcolumn should not be specified.
  In this case, OFFICECUBICLE entity has a independent Primary key which is auto-generated.

One-to-many relationship

A one-to-many relationship is where the source object has field that stores a collection of target objects. These target objects may also have an inverse relationship back to the source object. This relationship is established by having a foreign key in the target table that maps to the primary key of the source table.
An example of a one-to-many relationship is the relation between artist and art, where the artist has many artwork pieces.
A one-to-many relationship between two persistent components is defined using the `fieldtype` value `one-to-many` in the `cfproperty` tag. The source object contains a collection of target objects. ColdFusion allows the collection to be one of the following types:

- Array
- Struct
  This collection is a persistence aware collection. Any addition or deletion from this collection is automatically persisted in the database.

Array

An Artist object can contain the Art objects as an array. To define this mapping in the CFC, use the following syntax:

Syntax

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cfproperty
name="field_name"
fieldtype="one-to-many"
cfc="Referenced_CFC_name"
linktable="Link table name"
linkcatalog="Catalog for the link table"
linkschema="Schema for the link table"
fkcolumn="Foreign Key column name"
inversejoincolumn="Column name or comma-separated list of primary key columns "
type="array"
orderby="order_by_string"
cascade="cascade_options"
lazy="[true]|false|extra"
fetch="join|[select]"
inverse="true|[false]"
batchsize="N"
optimisticlock="[true]|false"
readonly="true|[false]">
```

For the artist-art example, the relationship property in Artist.cfc is defined as follows:

```
<cfproperty name="art" type="array" fieldtype="one-to-many" cfc="Art"
fkcolumn="ARTISTID">
```

- `type-array` specifies that the artist object will contain art objects in an array.
- `fkcolumn="ArtistID"` specifies that the foreign key column is ARTISTID that references the primary key of ARTIST table.

Struct

An Artist object can contain the Art objects as a struct. The key would be any column in the ART table (usually the primary key or a unique key). The value would be the Art object. To define this mapping, use the following syntax.

Syntax

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cfproperty
name="field_name"
fieldtype="one-to-many"
cfc="Referenced_CFC_name"
linktable="Link table name"
linkcatalog="Catalog for the link table"
linkschema="Schema for the link table"
fkcolumn="Foreign Key column name"
inversejoincolumn="Column name or comma-separated list of primary key columns"
type="struct"
orderby="order_by_String"
structkeycolumn = "Structure_Key_Column"
structkeytype="ormtype"
cascade="cascade_options"
lazy="[true]|false|extra"
fetch="join|[select]"
inverse="true|[false]"
batchsize="N"
optimisticlock="[true]|false"
readonly="true|[false]">
```

For the artist-art example, you can define the relationship property as:

```
<cfproperty name="art" type="struct" fieldtype="one-to-many" cfc="Art"
fkcolumn="ARTISTID" structkeytype="int" structkeycolumn="ArtID">
```

- `type=struct` specifies that the artist object will contain art objects in a struct.
- `structkeycolumn="ArtID"` specifies that the key of the struct would be ArtID. Note that ARTID is the primary key in Art table.
- `structkeytype="int"` specifies the datatype of `structkeycolumn`.
- `fkcolumn="ArtistID"` specifies that the foreign key column is ARTISTID that references the primary key of Artist table.

Many-to-one relationship

A many-to-one relationship is the inverse of a one-to-many relationship. In this relationship, many source objects can reference the same target object.
An example of this relationship is the relation between Art and Artist, where many Art are created by the same Artist. This relationship is established with the foreign key in the source table that references the primary key in the target table.
A many-to-one relationship between two persistent components is defined using the `fieldtype` value `many-to-one` in the `cfproperty` tag.

Syntax

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cfproperty
name="fieldname"
fieldtype="many-to-one"
cfc="Referenced_CFC_Name"
linktable="Link table name"
linkcatalog="Catalog for the link table"
linkschema="Schema for the link table"
fkcolumn="Foreign Key column name"
inversejoincolumn="Column name or comma-separated list of primary key columns"
column="Foreign_Key_Column"
mappedby="Mapped_Field_name_in_referenced_CFC"
cascade="cascade_options"
fetch="join|[select]"
lazy="true|false"
insert="[true]|false"
update="[true]|false"
optimisticlock="[true]|false"
missingrowIgnored="true|[false]">
```

For the art-artist example, the relationship in the ART.cfc can be defined as:

```
<cfproperty name="artist" fieldtype="many-to-one" fkcolumn="artistid" cfc="Artist">
```

`fkcolumn="ARTISTID"` indicates that the foreign key column in Art table references the primary key ARTISTID of ARTIST table.

Many-to-many relationships

A many-to-many relationship is where the source objects contain a collection of target objects and the target objects in turn contain a collection of source objects.
An example of a many-to-many relationship is the relation between Order and Product, where an order has many products and a product has many orders.
This relationship is established by using a third table called a 'LinkTable'. The LinkTable contains the foreign key to both the tables participating in the relation. ORM looks for the map key column in the LinkTable and not the target table.
In the preceding example of Order-Product, a many-to-many relationship is established by using LinkTable.
A many-to-many relationship between two persistent CFCs is defined using the `fieldtype="many-to-many"` value in the `cfproperty` tag.

> **Note**
>
> If the fkcolumn name is not specified, ORM generates the fkcolumn name in the "#relationName#_ID" format.

Syntax

Order.cfc

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cfproperty
name="fieldname"
fieldtype="many-to-many"
cfc="fully qualified name"
linktable="Link table name"
linkcatalog="Catalog for the link table"
linkschema="Schema for the link table"
fkcolumn="Foreign Key column name"
inversejoincolumn="Column name or a composite key with comma-separated primary key
columns"
mappedby="Property in the target component that is referenced by fkcolumn in join
table"
type="[array]|struct"
orderby="order by String
structkeycolumn="The structure key column name"
structkeydatatype="datatype".
cascade="cascade options" inverse="true|[false]" lazy = "[true]|false" [optional]
fetch="join|[select]" [optional] batchsize="integer" optimisticlock="[true]|false"
readonly="true|[false]"
missingrowIgnored="true|[false]">
```

For the Order-Product example, the many-to-many relationship is established using a third table "OrderProduct" that has two foreign keys: OrderId and ProductId. OrderId references the primary key orderId in the order table, and ProductId references the primary key productId in the Product table. This relationship can be defined as follows:

- Order.cfc

```
<cfproperty
name="products"
fieldtype="many-to-many"
CFC="Product"
linktable="Order_Product"
FKColumn="orderId"
inversejoincolumn="productId"
lazy="true"
cascade="all"
orderby="productId">
```

- Product.cfc

```
<cfproperty
name="orders"
fieldtype="many-to-many"
CFC="Order"
linktable="Order_Product"
FKColumn="productId"
inversejoincolumn="orderId"
lazy="true"
cascade="all"
orderby="orderId">
```

© 2014 Adobe Systems Incorporated. All rights reserved.

The `fkcolumn` here is the foreign key in the link table that references the primary key of the source table. `Inverseje joinColumn` is the foreign key in the link table that references the primary key of the target table. This attribute can also take a composite key value, for example you can specify inversejoincolumn="field1, field2", where field1 and field2 form the composite key.

© 2014 Adobe Systems Incorporated. All rights reserved.

Advanced mapping

Collection Mapping

Collection mapping is similar to a one-to-many relationship mapping. However, in collection mapping, you have a collection of values instead of a collection of persistent target objects.
Consider the Artist-Art tables. If you want each Artist object to contain an array of artwork names instead of artwork objects, collection mapping should be used.
To define collection mapping in the CFC, use `fieldtype="collection"` in the `cfproperty` tag.
The collection can either be `Array` or `Struct`.

Array

Syntax

```
name="field_name"
fieldtype="collection"
type="array"
table="table_name"
fkcolumn="foreign_key_column_name"
elementtype="ormtype"
elementColumn="column_name from the link table that should be
used for populating"
orderby="order by string"
lazy = "true|[false]"
readonly="true|[false]"
optimisticlock="[true]|false"
batchsize="batch size">
```

Example

If each Artist object contains an array of artwork names instead of artwork objects, this mapping can be defined in Artist.cfc as:

```
<cfproperty name="artNames" fieldtype="collection" type="array" table="ART"
fkcolumn="ARTISTID" elementcolumn="ARTNAME" elementtype="string">
```

Attribute

Attribute	Req/Opt	Default	Description
`batchsize`	Optional		An integer value that specifies the "`batchsize`" for fetching uninitialized collections. For details, see Lazy Loading.

© 2014 Adobe Systems Incorporated. All rights reserved.

`elementColumn`	Required		Specifies the column name that contains the data to be fetched for collection.
`elementtype`	Optional	String	Data type of the selected column. See *ORM data types* in Map the properties for details.
`fieldtype`	Required		Should be "collection".
`fkcolumn`	Optional		The foreign key column in the specified table.If you do not specify the foreign key column and `useDBForMapping` is `true` in `orm settings`, ColdFusion automatically determines a foreign key column after inspecting the database.
`lazy`	Optional	true	Specifies if loading is to be done lazily:truefalseSee Lazy Loading for details.
`name`	Required		Name of the collection.
`optimisticlock`	Optional	true	Specifies the locking strategy.truefalse
`orderBy`	Optional		Specifies the Order By string.
`readonly`	Optional	false	truefalseIf set to true, it indicates that the collection never changes and can be cached.
`table`	Required		Name of the table from where the values will be fetched.
`type`	Optional	array	Specifies if the collection type is:arraystruct

Struct

Syntax

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cfproperty
name="field_name"
fieldtype="collection"
type="struct"
table="table_name"
fkcolumn="foreign_key_column_name"
structkeycolumn="column in the target table to be used as key in the struct"
structkeytype="ormtype of the key in the struct"
elementtype="ormtype of the valye in the struct"
elementColumn="column name from the table that should be used in
value of struct"
orderby="order by string"
lazy = "[true]|false"
readonly="true|[false]"
optimisticlock="[true]|false"
batchsize="batch size">
```

Attribute	Req/Opt	Default	Description
`batchsize`	Optional		An integer value that specifies the "`batchsize`" for lazily fetching instances of this collection.
`elementcolumn`	Required		Specifies the column name that contains the data to be fetched for collection.
`elementtype`	Required		Data type of the value. See *ORM data types* in Map the properties for details.
`fieldtype`	Required		Should be a collection.
`fkcolumn`	Optional		The foreign key column in the table.If foreign key column is not specified and `useDBForMapping` is set to `true` in `ORMSetting`, ColdFusion automatically determines the Foreign Key column after inspecting the database.

© 2014 Adobe Systems Incorporated. All rights reserved.

`lazy`	Optional	true	Specifies if loading is to be done lazily: truefalseSee Lazy Loading for details.
`name`	Required		Name of the collection property.
`optimisticlock`	Optional	true	truefalse
`orderby`	Optional		Specifies the Order By string.
`readonly`	Optional	false	Value are:truefalseIf you set it to true, the collection never changes and can be cached.
`structkeycolumn`	Required		Column name in the table that will be used as key of struct.
`structkeyType`	Required		Specifies the data type of the key, when `type=struct`. For the entire list of data types, see the Data Type section.
`table`	Required		Name of the table from where the collection will be fetched.
`type`	Optional	array	Specifies if the collection type is: arraystruct

Inheritance mapping

If the object you need to persist has a hierarchy, the CFCs of that object hierarchy need to be mapped to the relational tables such that the entire hierarchy is persisted.
There are multiple strategies followed for inheritance mapping:

- Table per hierarchy
- Table per subclass without discriminator
- Table per subclass with discriminator

Table per hierarchy

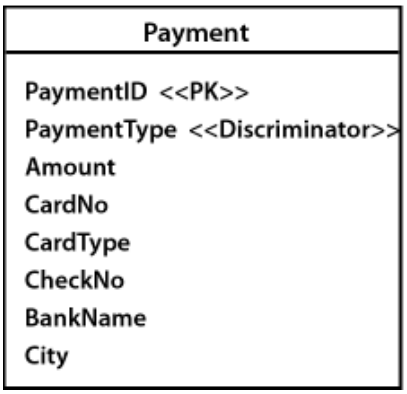
In this model, the object hierarchy is persisted in a single table. This table includes columns for all the properties of all the CFCs in the hierarchy. The concrete subclass represented by a row is identified based on the value of the `discriminator` column. In this strategy, all the CFCs of the hierarchy must have the same table name.

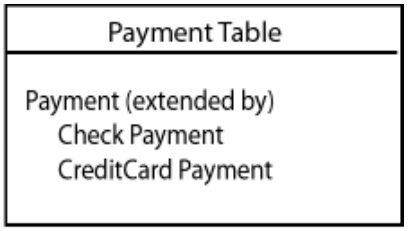
© 2014 Adobe Systems Incorporated. All rights reserved.

> **Note**
>
> If the discriminator column and discriminator value is not specified, a default discriminator column name and value is picked up.

Example

The following example demonstrates an implementation of table per hierarchy:





In the preceding figure, `discriminatorColumn` is `PaymentType`. Depending on the values of `PaymentType` wh ether it is credit card or check, the row is represented as a `CreditCardpayment` or `checkPayment` object respectively.

The following example illustrates how you can model the table per hierarchy:

Payment.cfc (parent class)

```
<cfcomponent persistent="true" table="Payment" discriminatorColumn="paymentType">
<cfproperty name="id">
  <cfproperty name="amount">
</cfcomponent>
```

CreditCardPayment.cfc

```
<cfcomponent persistent="true" extends="Payment" table="Payment"
discriminatorValue="CCard">
 <cfproperty name="cardNo">
 <cfproperty name="cardType">
</cfcomponent>
```

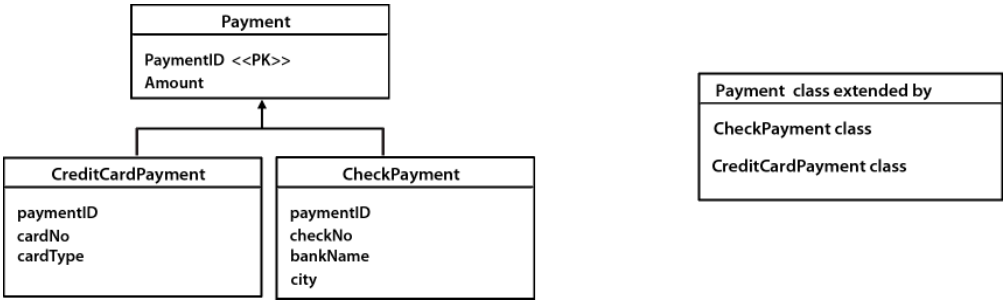
CheckPayment.cfc

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cfcomponent persistent="true" extends="Payment" table="Payment"
discriminatorValue="check">
 <cfproperty name="checkNo">
 <cfproperty name="bankName">
 <cfproperty name="city">
</cfcomponent>
```

Table per subclass without discriminator

In this model, there are separate tables for each class in the hierarchy and these tables are joined by a primary key. When the object is persisted, properties of the parent component are stored in the parent table and the remaining properties are stored in the child table.



In the preceding figure, the tables are joined by join column `paymentId`. You can model the tables as follows:

Payment.cfc

```
<cfcomponent persistent="true" table="Payment">
  <cfproperty name="paymentId">
  <cfproperty name="amount">
</cfcomponent>
```

CreditCardpayment.cfc

```
<cfcomponent persistent="true" extends="Payment" table="CreditCardPayment"
              joinColumn="paymentId">
 <cfproperty name="cardNo">
  <cfproperty name="cardType">
</cfcomponent>
```

CheckPayment.cfc

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cfcomponent persistent="true" extends="Payment" table="CheckPayment"
joinColumn="paymentId">
 <cfproperty name="checkNo">
  <cfproperty name="bankName">
  <cfproperty name="city">
</cfcomponent>
```

When an object of type `CreditCardPayment` is persisted, the property `amount` is stored in the payment table and the properties `cardNo` and `cardType` are stored in the CreditCardPayment table. The primary key of the CreditCardPayment remains the same as the primary key of the Payment table.

Table per subclass with discriminator

This model is similar to the table per subclass without discriminator strategy except that there is a discriminator column in the parent table. In addition, the child components has a `disciminatorValue` attribute in the `cfcomponent` tag.
The following example demonstrates the table per subclass with discriminator attribute:
Payment.cfc

```
<cfcomponent persistent="true" table="Payment" discriminatorColumn="paymentType">

 <cfproperty name="paymentId">
 <cfproperty name="amount">
</cfcomponent>
```

CreditCardPayment.cfc

```
<cfcomponent persistent="true" extends="Payment" table="CreditCardPayment"
joinColumn="paymentId" discriminatorValue="CCard">
 <cfproperty name="cardNo">
 <cfproperty name="cardType">
</cfcomponent>
```

CheckPayment.cfc

```
<cfcomponent persistent="true" extends="Payment" table="CheckPayment"
joinColumn="paymentId" discriminatorValue="Check">
 <cfproperty name="checkNo">
 <cfproperty name="bankName">
 <cfproperty name="city">
</cfcomponent>
```

When an object of type CreditCardPayment is persisted, the property `amount` is stored in the payment table and the properties `cardNo` and `cardType` are stored in the CreditCardPayment table. The primary key of CreditCardPayment remains the same as the primary key of the Payment table. The value of PaymentType is the

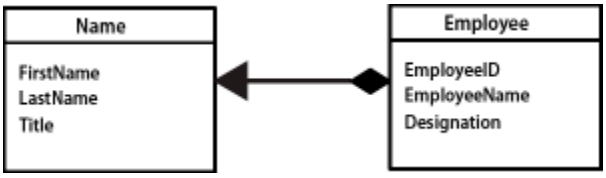
© 2014 Adobe Systems Incorporated. All rights reserved.

value of `disciminatorColumn` attribute of the respective object.

Embedded mapping

This mapping is used when a CFC has an embedded object which also needs to be persisted along with the parent's data. The CFC of the embedded object must have the attribute `embedded` set to "`true`" on the cfcomponent tag.

> The embedded object cannot be a persistent object. This feature is supported only when the hibernate mapping is explicitly defined in the hibernate mapping file (.hbmxml files).



The diagram shows two CFCs Employee and Name where EmployeeName field of the Employee.cfc is an object of Name.cfc. In the database, both these objects are persisted in the Employee table as a single row. Name object itself does not have its own identity. This mapping can be modeled as follows:

name.cfc

```
<cfcomponent embedded="true">
 <cfproperty name="FirstName">
 <cfproperty name="LastName">
 <cfproperty name=" Title">
</cfcomponent>
```

employee.cfc

```
<cfcomponent persistent="true">
 <cfproperty name="EmployeeID">
 <cfproperty name="EmployeeName">
 <cfproperty name="Designation">
</cfcomponent>
```

employee.hbmxml

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<hibernate-mapping >
 <class  name="cfc:Employee" table="Employees">
  <id name="EmployeeID" type="integer" column="EmployeeID">
   <generator class="native"/>
  </id>
  <component name="EmployeeName" class="cfc:Name">
  <property name="LastName" type="string" column="LastName"/>
  <property name="FirstName" type="string" column="FirstName"/>
  <property name="Title" type="string" column="Title"/>
  </component>
  <property name="Designation" type="string" column="Designation"/>
 </class>
</hibernate-mapping>
```

If the persistent CFC has a collection of embedded objects, then this mapping also has to be defined in the XML as shown in the following example. Here, employee object has a collection of IMData objects. Note that the IMData object is not persistent.

employee.cfc

```
<cfcomponent     persistent="true">
 <cfproperty name="EmployeeID">
 <cfproperty name="EmployeeName">
 <cfproperty name= "IMIDs" type="array">
 <cfproperty name="Designation">
</cfcomponent>
```

IMData.cfc

```
<cfcomponent embedded="true">
 <cfproperty name="type">
 <cfproperty name="ID">
</cfcomponent>
```

employee.hbmxml

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<hibernate-mapping>
    <class  name="cfc:Employee" table="Employees">
        <id name="EmployeeID" type="integer" column="EmployeeID">
                        <generator class="native"/>
        </id>
        <property name="EmployeeName" type="string" column="EmployeeName"/>
            <bag name="IMIDs" table="IMData" lazy="true">
                <key column="EmployeeID" />
                <composite-element class="cfc:IMData">
                    <property name="type" type="string" column="Type"/>
                    <property name="ID" type="string" column="ID"/>
                </composite-element>
            </bag>
            <property name="Designation" type="string" column="Designation"/>
    </class>
</hibernate-mapping>
```

Emp.cfm

```
<cfscript>
 employee = EntityNew("Employee");
 employee.setEmployeeName("Dan Watson");
 imdata1 = new IMData();
 imdata1.setType("IMClient1");
 imdata1.setID("msngrId1");
 imdata2 = new IMData();
 imdata2.setType("IMClient 2");
 imdata2.setID("msngrId2");
 employee.setIMIDs([imdata1, imdata2]);
 EntitySave(employee);
</cfscript>
```

Join mapping in a CFC

Join mapping is used to map the properties of one CFC to several tables. In this case, tables are joined using a join column. For example, consider a case where the Employee and Address tables, are mapped to a single CFC Employees. Therefore, the `employee.cfc` has some fields, which are persisted in Employee table and some fields that are persisted in Address table. The attributes `joincolumn` and `table` should be specified for those fields that need to be persisted in Address table. In this case, table would be `Address` and `joinColumn` would be `AddressID`.

> **Note**
>
> Hibernate uses outer join by default for join fetching. For inner join, use HQL.

Following is a sample employee.cfc
Employee.cfc

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cfcomponent persistent="true">
   <cfproperty name="id">
 <cfproperty name="name">
 <cfproperty name="houseno" column="houseno" table="Address" joincolumn="addressId">

 <cfproperty name="street" table="Address" joincolumn="addressId">
 <cfproperty name="city" table="Address" joincolumn="addressId">
 <cfproperty name="country" table="Address" joincolumn="addressId">
</cfcomponent>
```

Define the ORM mapping in the Hibernate mapping file

ColdFusion can also use the standard Hibernate Mapping XML file to define the mapping between the object and the database. You can use both Java classes and CFCs in Hibernate mapping.
Note the following points when using Hibernate mapping files.

- The extension of the Hibernate configuration file is *.hbmxml.
- The file is placed in the Application folder.
- The class name must be specified as *cfc:<fully qualified name of cfc>*. If a package is specified in the hibernate mapping, then specify the class name as *cfc:<name of cfc>*.
- The entityname attribute is optional. If you do not specify this attribute, it takes the component name, by default. For example, for the component artgallery.art, the value of the entityname attribute is "Art", by default.
- The entity name must be unique for an application. If there are two components with the same name (even if they are in different packages), specify different entity names for each of these components.
  The following is an example of Hibernate mapping:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD
3.0//EN" "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<hibernate-mapping>
 <class lazy="true" name="cfc:artGallery.Art" schema="APP" table="Art">
 <id name="artid" type="int">
  <column length="10" name="ARTID"/>
  <generator class="identity"/>
 </id>
  <property name="artname" type="string">
   <column length="50" name="ARTNAME"/>
  </property>

  <property name="price" type="java.math.BigDecimal">
   <column length="19" name="PRICE"/>
  </property>

  <property name="largeimage" type="string">
   <column length="30" name="LARGEIMAGE"/>
  </property>

  <property name="mediaid" type="int">
   <column length="10" name="MEDIAID"/>
  </property>

  <property name="issold" type="boolean">
   <column length="5" name="ISSOLD"/>
  </property>
  <many-to-one class="cfc:artGallery.Artists"  column="artistid"
name="artist"/>
 </class>
</hibernate-mapping>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

Mapping CFCs without properties

If a persistent CFC has no property, all table columns are added as properties provided that the table exists and you have specified it.

Example

Artist.cfc

```
component persistent="true" table="artists"
{
}
```

index.cfm

```
<cfset artists = entityLoad("Artist")>
<cfdump var="#artists#">
```

The columns are fetched only as column properties. Relationship, timestamp, or version field are not added.

Limitations

- If the mapping file for the CFC exists, then the properties are not added automatically.
- In the case of inheritance, properties are not added to the CFC automatically.

#back to top

© 2014 Adobe Systems Incorporated. All rights reserved.

Working with objects

Operations can be performed on an Entity object, and the auto-generated methods in the entity can be called.

© 2014 Adobe Systems Incorporated. All rights reserved.

Generated accessors

As described in Define ORM mapping, the persistent fields for an object are defined in the CFC using `cfproperty`. ColdFusion generates the accessor methods (getter and setter) for each property in the CFC that can be invoked. For more information, see *Implicit Get and Set Functions* in Functions. For example, if a property is defined in Artist as follows:

```
<cfproperty name="firstName" >
```

There are two methods generated in the Artist object:

- `setFirstName(firstName)`
- `getFirstName()`
  You can invoke these methods like regular methods in the CFC. For a property, the generated setter saves the value for the property in the object's VARIABLES scope. The generated getter retrieves the value of the property from the VARIABLES scope. ORM always uses the property value from the VARIABLES scope. That is, while saving the object's data in the table, ORM retrieves the value of the property from VARIABLES scope. While populating the object after reading from the table, ORM puts the property's value in the VARIABLES scope.If you define your own accessor methods for a property, store the property value in the VARIABLES scope for ORM to access it.

© 2014 Adobe Systems Incorporated. All rights reserved.

Generated methods for relationships between CFCs

When a relationship is defined in a CFC, ColdFusion generates a few methods in the CFC object to add or remove associated objects and to check their existence, for each relationship defined in the CFC.
The generated methods for relationships include:

- `add<relationship_property_name>()}}This method is generated for one-to-many and many-to-many relationships. The method adds the given object to the association collection (array or struct) of the component. For a bidirectional relationship, this method does not set the association on the other end.For{{ type="array"`, the method signature is:

  ```
  add<relationship_property_name>(<associated_object>)
  ```

  For `type="struct"`, the method signature is:

  ```
  add<relationship_property_name>(<key>, <associated_object>)
  ```

- `boolean remove<relationship_property_name>()}}This method is generated for one-to-many and many-to-many relationships. The method removes the object from the associated collection (array or struct) of the component. If the associated object was removed from the collection successfully, then true is returned. For a bidirectional relationship, this method does not remove the association from the other end.For {{type="array"`, the method signature is:

  ```
  boolean remove<relationship_property_name>(<associated_object>)
  ```

  For `type="struct"`, the method signature is:

  ```
  boolean remove<relationship_property_name>(<key>).
  ```

- {{boolean Has<relationship_property_name>()}}This method is generated for all the relationships. For one-to-many and many-to-many, this method checks whether the association collection is empty. If the association collection is empty, it will return true. For one-to-one and many-to-one, this method checks whether the associated object exists.
- `boolean Has<relationship_property_name>(<associated_object>)}}This method is generated for one-to-many and many-to-many relationships. The method checks whether the given associated object is present in the association collection. If it is present, it returns true.For {{type="array"`, the method signature is

  ```
  {{boolean has<relationship_property_name>(<associated_object>)}}
  ```

  For `type="struct"`, the method signature is

© 2014 Adobe Systems Incorporated. All rights reserved.

```
{{boolean has<relationship_property_name>(<key>)}}
```

Example

Consider the following example of artists (ARTISTS table) and artwork (ART table), where the artist forms a one-to-many relationship with artwork.Artist.cfc

```
<cfcomponent persistent="true" schema="APP" table="Artists">
 <cfproperty name="artistid" fieldtype="id"/>
 <cfproperty name="firstname"/>
 <cfproperty name="lastname"/>
 <cfproperty name="state"/>
 <cfproperty name="art" fieldtype="one-to-many" cfc="Art" fkcolumn="ArtistID" >
</cfcomponent>
```

Art.cfc

```
<cfcomponent persistent="true" schema="APP" table="Art">
 <cfproperty name="artid" fieldtype="id"/>
 <cfproperty name="artname"/>
 <cfproperty name="issold"/>
</cfcomponent>
```

In this example Artist has a relation field `art` with Art. The following methods are implicitly added to the Artist object:

- addArts(Art art)
- booleanremoveArts(Art art)
- booleanhasArts()
- booleanhasArts(Art art)
  The attribute `singularName`provides the flexibility to change the name of the generated relationship methods. For example, if the relationship property of Artist is specified as follows:

  ```
  <cfproperty name="art" fieldtype="one-to-many" cfc="Art" fkcolumn="ArtistID"
  singularName="Art">
  ```

  then the following methods are generated:

- addArt(Art art)
- removeArt(Art art)
- hasArt()
- hasArt(Art art)

© 2014 Adobe Systems Incorporated. All rights reserved.

Perform create, read, update, delete operations on ORM Objects

In any data-centric application, you can perform the following operations on the database:

- Insert (Create)
- Update
- Retrieve
- Delete
  Once the object relational model is defined in a ColdFusion application, you can perform CRUD operations on the objects directly using the methods provided by ColdFusion ORM. ColdFusion ORM, in turn, takes care of persisting the object in the database.

Create entities

`EntityNew`
Creates an object for persistent CFC with the given entity name. This is similar to CreateObject but it uses `entityname` whereas CreateObject takes CFC name. If there is no CFC defined in the application with the given `entityname`, an error will be thrown.
If the persistent CFC has an `init` method, then the function `EntityNew` calls the `init` method while creating the object.

Syntax

```
<entity> EntityNew("<entityName>")
```

Save entities

`EntitySave`
Saves or Updates the data of the entity and all related entities to the database.
ColdFusion automatically tries to find if a new record should be inserted or an existing record be updated for the given entity. If `forceinsert=true,` then ColdFusion always tries to insert the entity as a new record.
Calling this method may not run the insert/update SQL immediately. It is possible that various SQL statements are queued and then run as a batch for performance reasons. The SQL statements are run when the ORM session is flushed.

Syntax

```
EntitySave(entity, [forceinsert])
```

Parameter	Description
`entity`	The Entity that needs to be saved in the database.
`forceinsert`	If true, then ColdFusion always tries to insert the entity as a new record.

Example:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cfset artist = EntityNew("Artist")>
<cfset artist.setFirstName("Marcia")>
<cfset artist.setlastName("Em")>
<cfset EntitySave(artist)>
```

Update objects

`EntitySave`

The method to update an object is the same as saving an object. Load the object that needs to be updated, make updates, and save the object.

Syntax

```
EntitySave(entity, [forceinsert])
```

Example

The following example changes the first name of an artist with an Artist ID 1:

```
<cfset artist1 = EntityLoad("Artist", 1, true)>
<cfset artist1.setFirstName("Garcia")>
<cfset EntitySave(artist1)>
```

Read/Load objects

Entities are loaded using the `EntityLoad` methods. All `EntityLoad` methods take the entity name as input. If the persistent CFC has an `init` method, the methods call the `init` method while creating objects.

Syntax

```
EntityLoad (entityname)
EntityLoad (entityname, id [, unique])
EntityLoad (entityname, filtercriteria [,unique]
EntityLoad(entityname, filtercriteria, sortorder [, options])
EntityLoadByExample(sampleentity [, unique])
EntityReload(entity)
```

Examples

- {{EntityLoad (entityname)}}Loads and returns an array of entities of the specified entity name. For example, to retrieve all the objects of the "artist" CFC:

```
<cfset artist = EntityLoad('ARTIST')>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

- {{EntityLoad (entityname, id , unique)}}Loads and returns an entity whose Primary key's value is id. The entity is returned as an array by default. If unique is true, then the entity is returned.If the entity has a composite key, then the id has to be specified as key-value pairs (ColdFusion struct).Example 1: This example loads the Artist object with PK 100 and returns a single-element array containing the artist object.

```
<cfset artistArr = EntityLoad('Artist', 100)>
```

  Example 2:This example loads the Artist object with PK 100 and returns the artist object.

```
<cfset artistobj = EntityLoad('Artist', 100, true)>
```

  Example 3:This example loads the OrderDetail object which has the composite key OrderID=100 and ProductID=1 and returns the orderdetail object.

```
<cfset orderDetail = EntityLoad('orderdetails', {OrderID=100, ProductID=1},
true)>
```

- `EntityLoad (entityname, filtercriteria ,unique}}Loads and returns an array of entities of the given entity name that matches the {{filtercriteria. filtercriteria` is a key-value pair (ColdFusion struct) of property names and its values. If there are more than one key-value pair in `filtercriteria`, then they always use the AND operator. If you are sure that only one record exists that matches this `filtercriteria`, `unique=true` can be specified so that a single entity is returned instead of an array. If `unique=true`and multiple records are returned, then an exception occurs. For example, to retrieve details of all artists that have state CA:

```
<cfset artistsFromCA = EntityLoad('Artist', {state="CA"})>
```

  To retrieve a unique object, specify{{ unique= "true"}}. If more than one object satisfies the condition, an exception occurs.This example loads the artist object whose firstName is "Austin" and lastname is "Weber".

```
<cfset artist = EntityLoad('artist', {firstname="Austin", lastname="Weber"},
"true")>
```

- `EntityLoad(entityname,filtercriteria,sortorder, options)}}Loads and returns an array of entities that satisfy the {{filtercriteria` that is sorted as specified by the `sortorder` parameter.`filtercriteria` is a key-value pair (ColdFusion struct) of property names and its values. If there are more than one key-value pairs in `filtercriteria`, then they always use the AND operator.`sortorder` is a string, and should be specified in the following syntax:"propname1 asc, propname2 desc, ..."Some examples of `sortorder`are as follows:"firstname asc, lastname desc""firstname""country, age desc"Example: To retrieve artists whose state is CA, and sorted by City and FirstName:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cfset artistsFromCA = EntityLoad('artist', {state="CA"}, "city asc,
firstName")>
```

Certain configuration options can be input as name-value pairs as options argument. Several options can be specified to control the behavior of entity retrieval.

- `maxResults`: Specifies the maximum number of objects to be retrieved.
- `offset`: Specifies the start index of the resultset from where it has to start the retrieval.
- `cacheable`: Whether the result of this query is to be cached in the secondary cache. Default is false.
- `cachename`: Name of the cache in secondary cache.
- `timeout`: Specifies the timeout value (in seconds) for the query.
  `Maxresults` and `timeout`are used for pagination.*Example*To load first 5 artists whose state is "CA" that are sorted on the firstName.

```
<cfset artists = EntityLoad("Artist",{state='CA'}, "FirstName",
{maxResults=5})>
```

- `EntityLoadByExample(sampleentity ,unique)}}Loads and returns an array of objects that match the {{sampleentity`. The filter criteria is constructed by ANDing all the non-null properties of the `sampleentity`.For example, to retrieve an array of objects matching the specified values:

```
<cfset artist= CreateObject("component", "artist")>
<cfset artist.setState("CA")>
<cfset artist.setCity("Berkeley")>
<cfset artist=EntityLoadByExample(artist)>
```

  If you are sure that only one record exists that matches the specified `filtercriteria`, you can specify `unique=true` so that a single entity is returned instead of an array. If `unique=true` and multiple records are returned, then an exception occurs.

- {{EntityReload(entity)}}Reloads data for an entity that is already loaded in this session. This method refetches data from the database and repopulates the entity.

Delete objects

EntityDelete
This method deletes the record from the database for the specified entity. Depending on the cascade attribute specified in the mapping, it also deletes the associated objects.

Syntax

`EntityDelete(entity)`

Example

For example, to delete an Artist with ArtistID 5:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cfset artist = EntityLoad('artist', 5, true)>
<cfset EntityDelete(artist)>
```

Convert object to query

`EntitytoQuery}}This method converts the input entity object or the input array of entity objects to a query object. The name of the properties are used as the query column names. Use the optional parameter {{Entity_name` to return the query of the given entity in the case of inheritance mapping. All the objects in the input array should be of the same type. Relationship properties are not be included in the result query.

Syntax

```
EntitytoQuery (orm_object, [entity_name])
EntitytoQuery (orm_object_array, [entity_name])
```

Example 1

```
<cfset artists = EntityLoad("Artist")>
<cfset artistQuery = EntityToQuery(artists)>
```

Example 2

```
<cfset creditCardPayments = EntityLoad("CreditCardPayment")>
<cfset paymentQuery = EntityToQuery(creditCardPayments, "payment")>
```

Merge entities

`EntityMerge`
To attach an entity to the current ORM session you can use the entitymerge function. It copies the state of the given object onto the persistent object with the same identifier and returns the persistent object.
If there is no persistent instance currently associated with the session, it is loaded. The given instance is not associated with the session. You need to use the returned object from this session. For details, see EntityMerge in *C FML Reference*.

#back to top

© 2014 Adobe Systems Incorporated. All rights reserved.

Using queries

ColdFusion lets you use HQL (Hibernate Query Language) to run queries directly on the database. If you are familiar with HQL, you can use it for running complex queries.
In general, use HQL in the following scenarios:

- The query is not specific to a particular object but only to some fields in the object.
- To retrieve some fields of the object without loading the object.
- When you use table joins.
- When you use aggregate functions like `min`, `max`, `avg`, and `count`.
- To retrieve entities by specifying a filter that needs to use operators other than AND.
  The HQL methods return a single or multi-dimensional array of values or entities, based on what the HQL query returns.
  If you are sure that only one record exists that matches this filter criteria, specify `unique=true`so that a single entity is returned instead of an array. You can use unique=true to suppress the duplicate records from the query result.

  > **Note**
  >
  > `entityname` and `properties` used in HQL are case sensitive.

  The following HQL methods are available:

```
ORMExecuteQuery(hql, [params] [,unique])
ORMExecuteQuery(hql, [,unique] [, queryoptions])
ORMExecuteQuery(hql, params [,unique] [,queryOptions])
ORMExecuteQuery (hql, params, boolean unique, Map queryOptions)
```

© 2014 Adobe Systems Incorporated. All rights reserved.

ORMExecuteQuery(hql, [,unique] [, queryoptions])

Runs the HQL on the default data source specified for the application. You can specify several options to control the behavior of retrieval using queryoptions:

- `maxResults`: Specifies the maximum number of objects to be retrieved.
- `offset`: Specifies the start index of the resultset from where it has to start the retrieval.
- `cacheable`: Whether the result of this query is to be cached in the secondary cache. Default is false.
- `cachename`: Name of the cache in secondary cache.
- `timeout`: Specifies the timeout value (in seconds) for the query
  `Maxresults` and `timeout`are used for pagination.

> **Note**
>
> If the query returns an object or an array of objects, the `init` method of the persistent CFC is called (if available).

*Examples*To retrieve an array of artwork objects from the ART table:

```
<cfset art = ORMExecuteQuery("from ART")>
```

To retrieve an array of artwork objects that have a price greater than 400 dollars:

```
<cfset art = ORMExecuteQuery("from ART where price > 400")>
```

To retrieve an array of artwork objects that have a priceid 100:

```
<cfset artObj = ORMExecuteQuery("from ART where priceid = 100>
```

To retrieve an array of objects that contain the first name of artists:

```
<cfset firstNameArray = ORMExecuteQuery("select FirstName from Artist")>
```

To retrieve the number of artwork objects:

```
<cfset numberOfArts = ORMExecuteQuery("select count(*) from Art")>
```

To retrieve an array of objects that have an artistid 1:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cfset firstName = ORMExecuteQuery("select FirstName from Artist where
ARTISTID = 1", true)>
```

To retrieve an array of ten artist objects starting from the fifth row in the query result:

```
<cfset artists = ORMExecuteQuery("from Artist", false, {offset=5,
maxresults=10, timeout=5})>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

ORMExecuteQuery(hql, params [,unique] [,queryOptions])

This type of `ORMExecuteQuery` lets you pass unnamed parameters to the query. Use '?' (question mark) as the place-holder for the parameters. The values to the parameters should be passed as an array to `params`.**Examples**: unnamed parametersTo retrieve an array of artist objects with artistid equal to 40:

```
<cfset artists = ORMExecuteQuery("from ARTIST where artistid > ?", [40])>
```

To retrieve an array of artwork objects with a priceid equal to 1:

```
<cfset artObj = ORMExecuteQuery("from ART where priceid=?", [1], true)>
```

To retrieve an array of objects with a price id equal to 40, and price lesser than 80 dollars:

```
<cfset artists = ORMExecuteQuery("from ART where priceid > ? and price < ?", [40,
80])>
```

Note

In case of more than one parameter, values are picked up based on the parameter sequence, for example, the first parameter will be replaced by first value and second parameter will be replaced by second value.

Examples: named parametersThis type of `ORMExecuteQuery` lets you pass named parameters to the query. The placeholder for the parameter should be a name and should start with ":" as in ":age" or ":id". The values to the names should be passed as key-value pairs.For example, to retrieve artist details of all artists whose reside in USA and are also citizens of USA, your code should look like this:

```
<cfset USArtists = ORMExecuteQuery("from ARTIST where country=:country and
citizenship=:country", {country='USA'})>
<cfset orderDetail = ORMExecuteQuery("from Orders where OrderID=:orderid and
ProductID=:productid", {orderid=1, productid=901}, true)>
```

Note

Parameters are not case-sensitive.

Examples: group byThis type of ORMExecuteQuery lets you retrieve aggregate or grouped values for the query. For example, to retrieve the first name and last name along with the status of the artwork being sold or not, you can write a query similar to the following:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cfset artist = ORMExecuteQuery(
   "SELECT art.Artist.Firstname, art.Artist.Lastname, SUM(art.Price) as Sold FROM
Art as art WHERE art.IsSold=1 GROUP BY art.Artist.Firstname, art.Artist.Lastname")>
 <cfloop array="#artist#" index="artistItem">
 <cfoutput>
 #artistItem[1]# #artistItem[2]# #artistItem[3]#<br>
 </cfoutput>
</cfloop>
```

Note

Built-in functions to obtain the data such as getFirstName() or getLastName() cannot be used if you are using select queries with specific column names. The result will be returned as an array object and values can be retrieved using array index.

Example: order byThis type of ORMExecuteQuery lets you retrieve sorted data from a data source using the order by clause. For example, to sort the data from the Artist table by firstname, use the following code:

```
<cfset artist = ORMExecuteQuery('FROM Artist ORDER BY firstname ASC', false,
{maxresults=5} )>
<cfloop array="#artist#" index="artistObj">
 <cfoutput>Name = #artistObj.getFirstName()#
#artistObj.getLastName()#<br></cfoutput>
<br>
</cfloop>
```

Example: aggregate functionsThis type of `ORMExecuteQuery` lets you retrieve data when using aggregate functions such as sum, count, avg.

```
<cfset artist = ORMExecuteQuery(
   "SELECT COUNT(*) FROM Art as art WHERE art.Artist.ArtistID=:ArtistID AND
art.IsSold=:Sold", { ArtistID=1, Sold=True }, True )>
<cfoutput>
 #artist#
</cfoutput><br>
```

Example: expressionsThis type of ORMExecuteQuery lets you retrieve data using expressions such as mathematical operators, logical operators, binary comparisons, and many others. For example, the following code is used to retrieve the price of an artwork, which is greater than or equal to 10000 along with the name and description of the artwork.

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cfset artArr = ORMExecuteQuery("from Art where price>=10000")>
<cfloop array="#artArr#" index="artObj">
 <cfoutput>
  Art Name = #artObj.getArtName()#<br>
  Description = #artObj.getDescription()#<br>
  Price = #artObj.getPrice()#<br>
 </cfoutput>
<br>
</cfloop>
```

#back to top

© 2014 Adobe Systems Incorporated. All rights reserved.

Transaction and concurrency

When ORM methods are invoked without any transaction, all the data is committed to the database when the ORM session is flushed. ORM session is flushed when`ORMFlush()`is called or if autoflush is enabled when the request ends.

This works fine when there is not much concurrency, however in most practical scenarios you would need to use transaction in your application so that the data in your database is always in a consistent state.

With ColdFusion ORM, you can manage transactions in the following two ways:

- **Using Hibernate transaction:** User has full control and ColdFusion does not intervene. The application has to flush/close the session and commit/rollback the transaction.

  For more information on transactions, go to the following URL:

  http://community.jboss.org/wiki/sessionsandtransactions

- **Using CFTransaction:** ColdFusion manages the transaction. Since a transaction cannot be distributed (across different data sources), application must ensure that the changes made in the transaction affect only one Hibernate session. That is, only one data source.

  ColdFusion allows reading of data from other sessions (data source) in a transaction but changes must be made in only one session. Multiple dirty sessions at any time in the transaction can result in exceptions and the transaction is rolled back. Before transaction begins, all existing sessions in the request are flushed. The previous session (if any) is reused.

  When the transaction is committed, the dirty session is automatically flushed (before committing the transaction). When the transaction is rolled back, the changed session cannot be used any longer because it can cause rolled back data to get committed later. Therefore, the session participating in the transaction is cleared when transaction is rolled back.

A description of transaction is beyond the scope of this document. For more information on transactions, see the hibernate documentation.

To run the ORM methods inside a transaction, they must be inside`<cftransaction>`. A simple example snippet of using ORM with`<cftransaction>`is as follows:

```
<cftransaction>
    <cfset acct1 = EntityLoad("Account", "101")>
    <cfset acct2 = EntityLoad("Account", "102")>
    <cfset acct1.debit(1000)>
    <cfset acct2.credit(1000)>
    <cfset EntitySave(acct1)>
    <cfset EntitySave(acct2)>
</cftransaction>
```

Because we have not called commit on the`<cftransaction>`specifically, it is automatically committed when the`<cftransaction>`ends.

All`<cftransaction>`semantics including savepoint, multiple rollbacks, multiple commits, and nested transactions work with ORM. You can also have both queries and ORM in the same`<cftransaction>`.

© 2014 Adobe Systems Incorporated. All rights reserved.

When<cftransaction>begins, any existing ORM session is flushed and closed, and a new ORM session is created. The<cftransaction>can be committed or rolled back using appropriate ColdFusion tags in<cftransaction>. When the transaction ends and has not been committed or rolled back explicitly, it is automatically committed and the ORM session is closed. If there is any error inside the transaction, without any exception handling, the transaction is rolled back.

For more details on<cftransaction>, see the *CFML Reference Guide*.

> Even if ORMFlush() is called explicitly inside a <cftransaction> tag, the SQL runs but the data is committed only when the transaction commits.

© 2014 Adobe Systems Incorporated. All rights reserved.

Change of behavior in ColdFusion 9 Update 1

When..	Behavior in ColdFusion 9	Changed behavior
When the transaction starts	The existing session closes and a new session starts.	The existing session is flushed and is reused.
When the transaction is committed	The existing session is flushed and closed.	The existing session is flushed.
When the transaction is rolled back	The existing session is closed without flushing.	The existing session is cleared.

© 2014 Adobe Systems Incorporated. All rights reserved.

Optimistic locking

A `<cftransaction>` could prevent scalability in highly concurrent applications because it locks a database for a transaction that could run for a longer time. Also, a transaction cannot run beyond the duration of a request. There can be scenarios where an object is loaded in one request and the same instance is updated in another request or by another application. In this scenario, the application needs to maintain the transaction semantics and prevent the update if the row has been modified by some other request. This can be achieved by using optimistic concurrency control, which allows high concurrency in your application along with high scalability.
Optimistic concurrency control uses either number-based or timestamp-based versioning approach. In a number-based approach, a version number is incremented and for the timestamp approach, a timestamp is set to the current time whenever the object is modified. It must be noted that version increment or timestamp updation is managed by Hibernate and is not triggered at the database level.

- Using version: To use optimistic concurrency control using version numbers, add a property with `fieldtype ='version'`in your CFC.For example:

```
/**
* @persistent
* @table Users
*/
component{
 property name="id" fieldtype="id" datatype="int" generator="native";
 property string fname;
 property string lname;
 property name="version" fieldtype="version" datatype="int" ;
}
```

  Whenever a user object is updated, its version number is automatically incremented. The version number is used in the SQL update statement in such a way that updating proceeds only when the version number has not been changed by some other request or some other application.In case updating fails because the version number was changed outside the current session, an error is thrown specifying that the session contained stale data.

- Using timestamp: To use optimistic concurrency control using timestamp, add a property with `fieldtype="timestamp"`in your CFC. For example:

```
/**
* @persistent
* @table Users
*/
component{
 property name="id" fieldtype="id" datatype="int" generator="native";
 property string fname;
 property string lname;
 property name="lastModified" fieldtype="timestamp";
}
```

  Whenever a user object is updated, its timestamp is automatically set to the current time. Sometimes this is preferred over version because it also tells you when the user object was last modified.In case updating fails because the timestamp was changed outside of the current session, an error is thrown specifying that the session contained stale data.

© 2014 Adobe Systems Incorporated. All rights reserved.

If you do not have version or timestamp properties in your object, it is still possible to use optimistic locking, but only for objects that are retrieved and modified in the same ORM session. For optimistic locking of detached objects (objects that were loaded in some other request/ORM session), you must use a version number or timestamp.
To use optimistic locking for objects that do not have version or timestamp, you need to set attribute `'optimistic-lock'` on the CFC. This attribute can take the following values:

- `all`: This means that all the properties are included in the `where` clause of update query.
- `dirty` (default): This means that only the modified properties are included in the `where` clause of the update query.
- `version`: This means that only the version field is included in the `where` clause of update query.
- `none`: This means that none of the properties are included in the `where`clause, which in effect means that optimistic concurrency is disabled for that component.Example:

```
/**
* @persistent
* @table Users
* @optimistic-lock all
*/
component{
 property name="id" fieldtype="id" datatype="int" generator="native";
 property string fname;
 property string lname;
}
```

Apart from defining optimistic lock control at the CFC level, you can also define it at the property level using `'optimisticlock'` (true|false: default true) attribute.
You can specify `optimisticlock=true` for a property to acquire optimistic lock when the property is updated. Setting this attribute determines whether a version increment will occur when the property is dirty.
In case of one-to-many and many-to-many association, if the state of the collection changes, then version of the corresponding entity is incremented. It is advised that you disable this setting for one-to-many associations.

#back to top

© 2014 Adobe Systems Incorporated. All rights reserved.

Performance optimization

© 2014 Adobe Systems Incorporated. All rights reserved.

Lazy Loading

Optimizing SQL queries enhances the performance of any data-centric application. Some of the common approaches used to optimize SQL queries are:

- Avoid round trips to the database and fetch all required data for an operation using a single SQL query using Joins.
- Fetch only required data to reduce the load on the database
  SQL queries are generated and executed by the underlying ORM engine. Therefore, Hibernate provides various hooks to optimize SQL. The fetching strategy is one of the most important hooks, which defines the data that to be fetched, the time of fetching the data, and the way in which it needs to be fetched.
  There are four strategies for loading an object and its associations.
- Immediate fetching
- Lazy fetching
- Eager fetching
- Batch fetching

> **Note**
>
> If memory tracking is enabled on a server, it accesses each field of the object to compute its size. As a result, even lazy fields are accessed causing the lazy fields to get loaded immediately.

Immediate fetching

In this strategy, the associated object is fetched immediately after the owning entity is fetched, either from the database using a separate SQL query or from the secondary cache. This is not an efficient strategy to use, unless the associated object is cached in the secondary cache or when separate queries are more efficient than a Join query. You can define this strategy by setting `lazy="false"` and `fetch="select"` for the relationship property definition in the CFC.

```
<cfproperty name="art" fieldtype="one-to-many" cfc="ART" fkcolumn="ARTISTID"
lazy="false" fetch="select">
```

With this strategy, on loading the artists object, its art object is loaded immediately using a separate SQL query. As a result, this strategy is extremely vulnerable to 'N+1 Select problem'.

Lazy fetching

In this strategy, the associated object or collection is fetched only when required. Although you need to send a new request to the database each time you need data, this strategy controls how much of data is loaded and when is it loaded. This helps in reducing the database load.
When you load an entity, by default, ColdFusion ORM loads the entity's data but relations and any mapped collections and are not loaded. They are loaded only when you want to load them by calling the getter method. Therefore, the relations and collection mappings are lazily loaded. For example, when the `artist` object is loaded, all its artworks are not loaded and they are loaded only when `getarts()` is called.
ColdFusion ORM provides three types of lazy loading for relationships:

- lazy: This is the default lazy loading that applies to collection mapping, one-to-many and many-to-many relationship. In this case, when you call the accessor for the collection/relation, the collection is fully loaded.

© 2014 Adobe Systems Incorporated. All rights reserved.

So, when you call `EntityLoad()` for a particular artist, its artworks are not loaded at that time. When you call artist.getarts(), all the art object belonging to the artist will get loaded. This is achieved by setting `lazy="true"`on the relationship property definition in the CFC.Example: In artist.cfc

```
<cfproperty name="art" fieldtype="one-to-many" cfc="ART" fkcolumn="artistId"
lazy="true">
```

- Extra lazy: This applies to one-to-many and many-to-many relationships. This type of lazy loading goes one step ahead of lazy and does not load all the associated objects when the accessor for that relation is called. It just loads the primary keys for those objects and keeps a proxy object for them. When you call any method on the wrapper object, that object's data is loaded from the database. For example, when you call `artist.getarts()`, it executes a query on the database to fetch the primary key of the related artwork objects and creates a proxy artwork object. So, you do not load the data for all the artwork objects in memory. When you access a particular artwork object and invoke any method on it, then it fires another query to the database to load that particular artwork object. This is achieved by setting `lazy="extra"`on the relationship property definition in the CFC.Example: In artist.cfc

```
<cfproperty name="art" fieldtype="one-to-many" cfc="art" fkcolumn="artistId"
lazy="extra" >
```

- proxy: This applies to one-to-one and many-to-one relationships. When the owner object is loaded, the related object is not loaded from the database. ColdFusion only creates a proxy object for the related object and when any method is invoked on the related object, the data for the proxy object is loaded from the database and populated in the proxy object. For example, if the art-artist table relation is lazy, when the art object is loaded, the artists object is not loaded and when you call `art.getartist()`, you would only get a proxy object. When you call any method on the proxy object, query gets executed on the database to load artist object's data. This is achieved by setting `lazy="true"`on the relationship property definition in the CFCExample: In ART.cfc

```
<cfproperty name="artist" fieldtype="many-to-one" cfc="artist"
fkcolumn="artistId" lazy="true">
```

> An entity is loaded only once in the request (in Hibernate session) and there is always only one copy of it in the request. So, for artwork and artist relationship, which is lazy, if the artist is already loaded, calling `art.getartist()` will not create a proxy object and will return the loaded artist object.

Lazy loading can be disabled by setting `lazy="false"` on the relationship property definition in the CFC. Choosing an appropriate lazy loading option is very important for the performance of your application. Extra lazy means more number of trips to the database (each trip to the database is expensive) but less data in memory whereas no lazy loading means a huge object graph in the memory. So, you need to balance the approach based on the application need.

Eager fetching

In this strategy, the associated object or collection is fetched together with the owning entity using a single SQL Join

© 2014 Adobe Systems Incorporated. All rights reserved.

query. This strategy reduces the number of trips to the database and is a good optimization technique when you always access the associated object immediately after loading the owning entity. You can define this strategy by setting `fetch="join"` for the relationship property definition in the CFC.

Batch fetching

This strategy tells Hibernate to optimize the second SQL select in Immediate fetching or lazy fetching to load batch of objects or collections in a single query. This allows you to load a batch of proxied objects or uninitilized collections that are referenced in the current request. This is generally useful in nested tree loading. You can specify this using `"batchsize"` attribute for CFC or relationship property.
There are two ways you can tune batch fetching:

- **Batch fetching at CFC level:** This allows batch fetching of the proxied objects and is applied to one-to-one and many-to-one relationship. For example, consider artwork and artist example where there are 25 art instances loaded in the request (ORM session). Each artwork has a reference to the artist and the relationship is lazy. Therefore, art objects contain the proxied object for artist. If you now iterate through all the art objects and call `getartist()` on each, by default 25 SELECT statements are executed to retrieve the proxied owners, one for each artist proxy object. This can be batched by specifying the `'batchsize'`attr ibute on the artist CFC:

  ```
  <cfcomponent table="artist" batchsize="10" ...>
  ```

  When you call `getartist()` on the first art object, it batch fetches 10 artist objects that are proxied in the current request.So for 25 art objects, this type of batch fetching makes Hibernate execute a maximum of three queries in batches of 10, 10, and 5.

- **Batch fetching at collections:** This allows batch fetching of value collections, one-to-many or many-to-many relationships that are uninitialized. For example, consider artist-art one-to-many relationship where there are 25 artists loaded and each artist has a lazy collection of artworks. If you now iterate through the artists and call `getarts()` on each, by default 25 SELECT statements are executed, one for each artist to load its art objects. This can be optimized by enabling batch fetching, which is done by specifying `"batchsize"`on the relationship property: Example:In artist.cfc:

  ```
  <cfproperty name="art" fieldtype="one-to-many" cfc="art" fkcolumn="artistId"
  lazy="true" batchsize="10">
  ```

  One important thing to understand here is that `batchsize` here does not mean that 10 artworks are loaded at one time for a artist. It actually means that 10 artwork collections (artworks for 10 artists) are loaded together.When you call `getarts()` on the first artist, artworks for 9 other artists are also fetched along with the one that was asked for. The value for `batchsize` attribute should be chosen based on the expected number of proxied objects or uninitialized collections in the session.

© 2014 Adobe Systems Incorporated. All rights reserved.

Caching

Caching is extensively used for optimizing database applications and effectively reducing traffic between the database and the application.
ColdFusion ORM supports two levels of caching:

- Session level
- Secondary level

Session level cache

Objects that are loaded from the database are always cached in the ORM Session as long as the session is open. When `EntityLoad` is called to retrieve an object in a session for the first time, ORM fetches the data from the database and constructs the object. In any subsequent call to load the same object in the same session, ORM fetches the object from the session cache. To forcefully retrieve the object from the database, `EntityReload` shoul d be called on the object.
For more details on ORM Sessions and its lifecycle, see ORM session management and Architecture.

Secondary level cache

ColdFusion provides the ability to store the data that is retrieved from the database in secondary cache. The contents in secondary cache live longer than the life-time of a session. It can also be the life-time of the process or in-definite (disk-caching), depending on the ability of the secondary cache provider. The cache can also be used in a distributed environment depending on the ability of the secondary cache provider.
An important difference between session level cache and secondary level cache is that the session level caches the whole object but the secondary level caches only the data.
Secondary level cache can be leveraged by using an external cache provider with ColdFusion ORM. EHCache, JBossCache, OSCache, SwarmCache, and Tangosol Coherence Cache are some popular secondary cache providers, which can be plugged into Hibernate.
ColdFusion uses EHCache as the default secondary cache provider. EHCache is a distributed caching solution that supports memory and disk-based caching. EHCache can be configured using a configuration file. Different cache regions can be defined in the configuration file. Each cache region has its own configuration that specifies details including the number of elements it can store, eviction policy, time to live (ttl), and idle time.
ehcache.xml is available in the following location: *CF_root*\lib\. For details of the properties in the ehcache.xml, refer to the documentation available at the following URL:
http://ehcache.org/
The following is a sample EHCache configuration file (`ehcache.xml`):

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<ehcache>
    <diskStore path="java.io.tmpdir"/>
    <defaultCache
        maxElementsInMemory="10000"
        eternal="false"
        timeToIdleSeconds="120"
        timeToLiveSeconds="120"
        overflowToDisk="true"
        diskPersistent="false"
        diskExpiryThreadIntervalSeconds="120"
        memoryStoreEvictionPolicy="LRU"
        />
    <cache name="Artist"
        maxElementsInMemory="20"
        eternal="true"
        overflowToDisk="false"
        />
</ehcache>
```

Modifications to ehcache.xml in ColdFusion 9.0.1

ehCache.xml includes the following configuration properties:

- `diskSpoolBufferSizeMB`: Size to allocate the DiskStore for a spool buffer. The default size is 30 MB. Each spool buffer is used only by its cache.Turning on trace-level logging shows if backup for cache created/updated using `action="put"` occurs in the diskstore.
- `clearOnFlush`: Determines if the MemoryStore must be cleared when the cache is flushed. By default, the MemoryStore is cleared.
- `diskExpiryThreadIntervalSeconds`: The number of seconds between runs of the disk expiry thread. The default value is `120`seconds.

> **Note**
>
> The functions `cacheGetProperties` and `cacheSetProperties` can be used to get/set these properties.

Use secondary cache

To use secondary cache, you must configure the following settings in the application:

- {{ormsettings.secondarycacheenabled}}This setting defines whether the secondary cache would be used by the application. By default, this is set to false.
- {{ormsettings.Cacheprovider}}This setting defines the cache provider that needs to be used for secondary cache. This defaults to EHCache. The other values for this setting are JBossCache, OSCache, SwarmCache and Hashtable. You can also specify the fully qualified class name of the cache provider.
- `ormsettings.cacheconfig}}This setting defines the configuration file required by the secondary cache provider. For example, EHCache requires {{EHCache.xml` that defines the configuration settings for the secondary cache. Specify the path to the XML file in this setting. If this setting is not defined, cache provider uses its default configuration.
  After you have configured the secondary cache, it is critical to identify the objects in your application that can be cached because the data cached by secondary cache is shared by all the sessions of an application. Typically, caching should be enabled for a CFC that represents:

© 2014 Adobe Systems Incorporated. All rights reserved.

- Data that changes rarely
- Data that is local to an application and is not modified by other applications
- Non-critical data
  For each type of object that needs to be cached, you also need to decide the access strategies. ORM provides the following cache strategies that you can use for your objects:
- read-only This strategy is useful for data that is read frequently but never updated. This is the best performing cache strategy.
- nonstrict-read-write This strategy is useful for data that is updated occasionally. Typically, it is very unlikely that two transactions would update the same object simultaneously.
- read-writeThis strategy may be appropriate if your data needs to be updated. It carries more overhead than the two preceding strategies.
- TransactionalThis strategy provides the support for transactional cache. It can only be used if the cache provider is transaction aware.
  Support for these strategies depend on the cache provider. Not all the cache providers support all the cache strategies.
  The secondary cache can cache the following types of data.
- Persistent object data
- Persistent object association
- Query data

Cache data of a persistent object

In this case, the data of the persistent object is cached. It will not cache the associations or associated object's data. To enable this flag on a persistent CFC, specify the following attributes on the component.

- `cacheuse`: Defines the caching strategy.
- `cachename`: Defines the name of the cache region to be used by the secondary cache provider. If you do not specify a region name for the component, the entity name of the component is considered as the cache name. In case a region is not specified in the configuration file, a region is automatically created with the default configuration.
  For example:

```
<cfcomponent persistent="true" schema="APP" table="Artists" cachename="artist"
cacheuse="read-only">
```

Cache the association data of a persistent object

In this case, the primary key of the associated objects are cached. It does not cache the objects loaded as part of the association unless caching is enabled for those objects. To cache an association, specify the following attributes on the association property.

- `cacheuse`: Defines the caching strategy.
- `cachename`: Defines the name of the cache region to be used by the secondary cache provider. If you do not specify a region name for the association property, the <comoponent_name>.<property_name> is considered as the cache name. In case a region is not specified in the configuration file, a region is automatically created with the default configuration.
  For example:

```
<cfproperty name="art" fieldtype="one-to-many" cfc="CArt" fkcolumn="ArtID"
cachename="ArtistArts" cacheuse="read-only">
```

© 2014 Adobe Systems Incorporated. All rights reserved.

Cache query data

In this case, the results of queries that are executed by `ORMExecuteQuery()` or `EntityLoad()` methods are cached in the secondary cache. To enable caching query data, pass "`cacheable=true`" and "`cachename='cachename'`" values in the options struct of the methods. If you do not specify the `cachename`, the query is cached in the default query cache. It is recommended that you to specify the `cachename` so that you can control eviction.
For example:

```
availableArts = ORMExecuteQuery("from CArt where issold=0", {}, false,
{cacheable=true, cachename="availableArtsQuery"});
```

Secondary cache example using EHCache

Step 1: Set the following in `Application.cfc`:

```
<cfset this.name="Caching_Example">
<cfset this.datasource="cfartgallery">
<cfset this.ormenabled="true">
<cfset this.ormsettings.secondarycacheEnabled=true>
<cfset this.ormsettings.cacheProvider= "ehcache">
<cfset this.ormsettings.cacheConfig="ehcache.xml">
```

Step 2: Define the cache settings in the CFCs.
CArtist.cfc

```
<cfcomponent persistent="true" schema="APP" table="Artists" cachename="artist"
cacheuse="read-only">
 <cfproperty name="artistid" fieldtype="id"/>
 <cfproperty name="firstname"/>
 <cfproperty name="lastname"/>
 <cfproperty name="state"/>
 <cfproperty name="art" fieldtype="one-to-many" cfc="CArt" fkcolumn="ArtID"
cachename="ArtistArts" cacheuse="read-only">
</cfcomponent>
```

CArt.cfc

```
<cfcomponent persistent="true" schema="APP" table="Art">
 <cfproperty name="artid" generator="identity" fieldtype="id"/>
 <cfproperty name="artname"/>
 <cfproperty name="issold"/>
</cfcomponent>
```

Step 3:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cfscript>
//This will cache the Artist Component and also the association.  It wouldn't cache
the Art objects.
artistObj = EntityLoad("CArtists", 3, true);
//This will cache the query.
availableArts = ORMExecuteQuery("from CArt where issold=0", {}, false,
{cacheable=true, cachename="availableArtsCache"});
</cfscript>
```

Evict content from secondary cache

ColdFusion provides the following methods to evict contents from the secondary cache.

`ORMEvictEntity("<component_name>", [primarykey])`

This method is used to evict items for the given component name, from the secondary cache. If the primary key is specified, then the data of the entity with that primary key is evicted. Primary key should be a value in case of simple primary key or should be a struct in case of composite primary key.

Example:

```
<cfset ORMEvictEntity("CArtists")>
```

Evicts all the cache data of CArtist entity.

```
<cfset ORMEvictEntity("CArtists", 1)>
```

Evicts the cache data of CArtists entity whose primary key is 1.

`ORMEvictCollection("<component_name>", "<collection_name>", [primarykey])`

This method is used to evict all the collection/association data for the given component name and collection name, from the secondary cache. If the primary key is specified, then, the collection or association data of the entity with the primary key is evicted.

Example:

```
<cfset ORMEvictCollection("CArtists", "art")>
```

Evicts all the association or collection data of collection art belonging to the component CArtists.

```
<cfset ORMEvictCollection("CArtists", "art", 1)>
```

Evict the association or collection data of collection art belonging to the component CArtists with primary key 1.

`ORMEvictQueries([cachename])`

This method is used to evict the data of all the queries from the default query cache. If cache name is specified, then, the data of all the queries belonging to the cache region with the given cache name are evicted. Example:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cfset ORMEvictQueries()>
```

Evicts the data of all the queries from the default query cache.

```
<cfset ORMEvictQueries("availableArtsCache")>
```

Evicts the data of all the queries from the cache region with the name `availableArtsCache`.

Support for user-defined caches in ColdFusion 9.0.1

Except in `cacheSetProperties` and `cacheGetProperties`, user-defined caches are supported in all caching functions.
Edit ehCache.xml (*cfroot*/lib)to set the properties for user-defined caches as shown in the following example:

```
<!--- item to put in user-defined cache --->
<cfset currentTime = Now()>
<!--- put item in user-defined cache --->
<cfset timeToLive=createtimespan(0,0,0,30)>
<cfset timeToIdle=createtimespan(0,0,0,30)>
<cfset customCache = "usercache">
<cfset id = "cache1">
<cfset cachePut(id,currentTime,timeToLive,timeToIdle,customCache)>
<!-- list items in the cache --->
List Items in cache:
<cfset cacheIds = cacheGetAllIds(customCache)>
<cfdump var="#cacheIds#"><br>
<!--- print cache data --->
<cfset cachedData = cacheGet(id,customCache)>
<cfoutput>#cachedData#</cfoutput>
<!--- print cache metadata --->
Cache metadata:
<cfset mdata = cacheGetMetadata(id,"object",customCache)>
<cfdump var="#mdata#">
<!--- clear user-defined cache --->
<cfset cacheRemove(ArrayToList(cacheIds),true,customCache)>
```

#back to top

© 2014 Adobe Systems Incorporated. All rights reserved.

ORM session management

Hibernate Session is a thread-safe and short-lived object that represents a conversation between the application and the persistence layer. This is required to perform CRUD operations (Create, Delete, Update, Retrieval) on a persistent entity in Hibernate. For ColdFusion applications that use ORM, this session is automatically managed by ColdFusion. Hibernate sessions also act as the first level of cache, which ensures that only one copy of an object exists in the session.

When CRUD operations are performed in the session, data of the entity is not synchronized with the database immediately. That is, the SQL statements for the operations are not issued immediately and they are queued. The data is synchronized with the database when the session is flushed. When the session is flushed, the SQL operations are performed in the following order:

1. all entity insertions, in the same order the corresponding objects were saved using `EntitySave()`
2. all entity updates
3. all collection deletions
4. all collection element deletions, updates, and insertions
5. all collection insertions
6. all entity deletions, in the same order the corresponding objects were deleted using `EntityDelete()`
   The only exception to this is that objects with nativeId generation are inserted immediately when the object is saved.

> **Note**
>
> ColdFusion creates and manages Hibernate sessions only if `ormenabled` is set to `true` i n application scope.

When the ColdFusion application starts, it builds the Hibernate session factory that is available for the application life time. This factory is used to create Hibernate sessions that manage the persistent object lifecycle. When the first CRUD method is called, a Hibernate session gets created and is closed when the request ends or when the `ormclosesessionmethod` is called. For details on how ColdFusion works with Hibernate, see the Architecture.

In multiple data source scenarios supported in ColdFusion 9 Update 1, there are multiple sessions (one for each data source) in the same request. For all entity functions, the appropriate sessions are used transparently.

ColdFusion exposes a few methods to let CFML developers work with the Hibernate sessions directly. ORM session-related functions also take optional data source argument. If you do not specify a data source, the default data source specified for ORM is used. The methods are as follows:

© 2014 Adobe Systems Incorporated. All rights reserved.

Event Handling in CFC

ORM provides callbacks to the event listeners for all the persistence events like `Load`, `Insert`, `Update`, `Delete`. These events can be used for data validations or transformation or for some generic functions such as auditing. These events can be handled at two levels in ColdFusion ORM:

- In a persistent CFC
- Using an event handler CFC
  To enable event handling for an application, define the following setting: `ormsettings.eventhandling="true"`
  By default, this flag is disabled. If you do not specify this flag while the event handler CFC is defined, the flag is considered as enabled.

© 2014 Adobe Systems Incorporated. All rights reserved.

Event handling in a persistent CFC

A persistent CFC can have various methods and if these methods are present, callbacks can be sent on those events to the CFC. The CFC can then handle these events. In this case, the event for entity persistence comes to the CFC that the system loads, inserts, updates, or deletes. These methods are:

- `preLoad()`: This method is called before the load operation or before the data is loaded from the database.
- `postLoad()`: This method is called after the load operation is complete.
- `preInsert()`: This method is called just before the object is inserted.
- `postInsert()`: This method is called after the insert operation is complete.
- `preUpdate(Struct oldData)`: This method is called just before the object is updated. A struct of old data is passed to this method to know the original state of the entity being updated.
- `postUpdate()`: This method is called after the update operation is complete.

> **Note**
>
> When you call the EntitySave() method on an object that is not loaded using EntityLoad(), it gets updated but the intercepter call fails. This happens because an empty map is created for the object and there is no previous data associated with it.

- `preDelete()`: This method is called before the object is deleted.
- `postDelete()`: This method is called after the delete operation is complete.

© 2014 Adobe Systems Incorporated. All rights reserved.

Event handling using an event handler CFC

An application-wide event handler CFC can be defined to handle callback when **any** entity is inserted, updated, deleted, or retrieved. This CFC must be configured at the application level as an ORM setting:

```
ormsettings.evenHandler="X.Y.EventHandler"
```

The event handler CFC needs to implement the `CFIDE.ORM.IEventHandler` interface. This CFC gets the callbacks from all persistence-related events and handles them accordingly. In this case, a single CFC handles the events for all the CFCs.
This interface contains the following methods for the event handler CFC:
For application-wide event handler CFC, you need to specify the component name also along with other arguments. The methods for application-wide event handler are:

- `preFlush(entity): This method is called before the flush operation is complete. This function can be overridden only in a global event handler for an application. CFCs for individual entities will not get this event call.`
- `postFlush(entity): This method is called after the flush operation is complete. This function can be overridden only in a global event handler for an application. CFCs for individual entities will not get this event call.`
- `preLoad(entity)`: This method is called before the load operation or before the data is loaded from the database.
- `postLoad(entity)`: This method is called after the load operation is complete.
- `preInsert(entity)`: This method is called just before the object is inserted.
- `postInsert(entity)`: This method is called after the insert operation is complete.
- `preUpdate(entity, Struct oldData)`: This method is called just before the object is updated. A struct of old data is passed to this method to know the original state of the entity being updated.

  > **Note**
  >
  > When you call the EntitySave() method on an object that is not loaded using EntityLoad(), it gets updated but the intercepter call fails. This happens because an empty map is created for the object and there is no previous data associated with it.

- `postUpdate(entity)`: This method is called after the update operation is complete.
- `preDelete(entity)`: This method is called before the object is deleted.
- `postDelete(entity)`: This method is called after the delete operation is complete.

  > **Note**
  >
  > If event handlers are defined in both persistent CFC and event handler CFC, the persistent CFC is given the callback before calling the application wide event handler.

#back to top

© 2014 Adobe Systems Incorporated. All rights reserved.

Autogenerating database schema

ColdFusion automatically creates tables when ORM is initialized for the application. For auto-generating tables, do the following:
In the THIS scope of Application.cfc, in `ormsettings` struct, set the `dbCreate` property to one of the following values:

- `update`: Creates the table (if it does not exist) or updates the table (if it exists).
- `dropcreate`: Drops the table if it exists and then creates it.
  For example,

```
<cfset this.ormsettings.dbCreate="update">
```

Certain specific attributes (DDL-only attributes) defined for the tags `cfcomponent` and `cfproperty` can be use to define various attributes for the auto-generated tables and columns. DDL-only attributes are used only for DDL generation. For details of these attributes, see the table in the section DDL-only attributes in Column.

© 2014 Adobe Systems Incorporated. All rights reserved.

Naming strategy

When you build a database centric application, typically you would follow some database standard and naming convention. ColdFusion ORM allows you to define this convention at one central place for the application using the 'naming strategy'.
The advantage of using a naming strategy is that you do not need to change the code throughout your application.
The naming strategy specifies how the table and column have to be named for a CFC and its properties.
Naming strategy takes "logical name" for a table or column and returns the actual table or column name that should be used.

- Logical table name: This is the table name specified for the CFC. If it is not specified, the entity name is taken as the logical table name. If the entity name is also not specified, the unqualified CFC name, for example, Person for a.b.c.Person, is taken as the logical table name.
- Logical column name: This is the column name specified for a CFC property. If it is not specified, the property name is taken as the logical column name.
  Naming strategy is applied to an application by setting the following in `Application.cfc`

  ```
  <cfset this.ormsettings.namingstrategy="strategy">
  ```

  The value of strategy could be:

- `default`: This strategy uses the logical table or column name as it is. ColdFusion ORM using this value as the default strategy.
- `smart`: This strategy changes the logical table or column name to uppercase. Also, if the logical table or column name is in camel case, this strategy breaks the camelcased name and separates the broken words using underscore. For example, for a CFC named "OrderProduct", this strategy changes the table name as "ORDER_PRODUCT".
- `your own cfc` : You can get complete control of the naming strategy by providing your own implementation. You need to specify the fully qualified name of the CFC as the value for naming strategy. This CFC must implement `cfide.orm.INamingStrategy` interface.
  The `cfide.orm.INamingStrategy`interface is as follows:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
/**
* Strategy to specify the table name for a CFC and column name for a property
in the cfc.
* This can be used to specify the application specific table and column naming
convention.
* This rule will be applied even if the user has specified the table/column
name in the mapping so that
* the name can be changed for any application at one place without changing
the names in all the code.
*/
interface
{
/**
* Defines the table name to be used for a specified table name. The specified
table name is either
* the table name specified in the mapping or chosen using the entity name.
*/
public string function getTableName(string tableName);

/**
* Defines the column name to be used for a specified column name. The
specified column name is either
* the column name specified in the mapping or chosen using the proeprty name.
*/
public string function getColumnName(string columnName);
}
```

This interface is specified in the application using:

```
this.ormsettings.namingstrategy="com.adobe.UCaseStrategy"
```

Note

The naming strategy applies to all the table or column names, which you use in the mapping including link table and fkcolumn, even though there is no CFC or cfproperty associated with them.

© 2014 Adobe Systems Incorporated. All rights reserved.

Example

Application.cfc

```
<cfset this.name = "AG"/>
<cfset this.ormenabled=true/>
<cfset this.datasource = "ORM_DDL">
<cfset this.ormsettings.dbCreate="dropcreate">
<cfset this.ormsettings.sqlscript="mysqlscript.sql">
```

Artists.cfc

```
<cfcomponent persistent="true" table="Artists">
      <cfproperty name="artistid" fieldtype="id" ormtype="integer" length=10>
      <cfproperty name="firstname" ormtype="string" length="20" notnull="true">
      <cfproperty name="lastname" ormtype="string" length="20" notnull="true">
      <cfproperty name="address"  ormtype="string" length="50">
      <cfproperty name="city" ormtype="string" length="20">
      <cfproperty name="state" ormtype="string" length="2">
      <cfproperty name="postalcode" ormtype="string" length="10">
      <cfproperty name="email" ormtype="string" length="50" unique="true">
      <cfproperty name="phone" ormtype="string" length="20">
      <cfproperty name="fax" ormtype="string" length="12">
      <cfproperty name="thepassword" ormtype="string" length="20">
</cfcomponent>
```

art.cfc

```
<cfcomponent persistent="true" table="Art">
      <cfproperty name="artid" generator="identity" fieldtype="id">
      <cfproperty name="artname" ormtype="string" length="50">
      <cfproperty name="price" ormtype="double">
      <cfproperty name="largeimage" ormtype="string" length="30">
      <cfproperty name="mediaid" ormtype="integer" length="10">
      <cfproperty name="issold" ormtype="boolean" dbdefault=1>
      <cfproperty name="artist" fkcolumn="artistid" fieldtype="many-to-one"
cfc="CArtists">
</cfcomponent>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

Populating the database using script

After the tables are created using DDL, you can optionally populate the database with data using SQL script. To do this, specify the path (absolute file path or path relative to the application) to the SQL script file that has to be executed. The script is run only when `dbcreate` is set to `dropcreate`. The SQL script file lets you populate the tables before the application is accessed. Ensure that each SQL statement starts on a new line and ends with a semi-colon.

Example

Mysqlscript.sql

```
insert into Artists(artistid, firstname, lastname, address, city, state, postalcode,
email, phone, fax, thepassword)
values(1, 'Aiden', 'Donolan', '352 Corporate Ave.', 'Denver', 'CO', '80206-4526',
'aiden.donolan@donolan.com', '555-751-8464', '555-751-8463', 'peapod');
insert into Artists(artistid, firstname, lastname, address, city, state, postalcode,
email, phone, fax, thepassword)
values(2, 'Austin', 'Weber', '25463 Main Street, Suite C', 'Berkeley', 'CA',
'94707-4513', 'austin@life.com', '555-513-4318', '510-513-4888', 'nopolyes');
insert into Art(artname, price, largeimage, mediaid, issold, artistid)
values('Michael', 13900, 'aiden02.jpg', 1, 0, 1);
insert into Art(artname, price, largeimage, mediaid, issold, artistid)
values('Space', 9800, 'elecia01.jpg', 2, 1, 2);
```

#back to top

© 2014 Adobe Systems Incorporated. All rights reserved.

Support for multiple data sources for ORM

Note

This feature applies only if you have installed ColdFusion 9 Update 1.

© 2014 Adobe Systems Incorporated. All rights reserved.

Configuring the application to use multiple data sources

Configure Persistent CFCs with the attribute `datasource` pointing to the appropriate data source. You can specify the attribute data source on the CFC using the tag `cfcomponent` or by specifying the annotation on Component in the CFC definition. If you do not specify a data source, the default data source is used for that CFC.
Since a Hibernate configuration uses a single data source, all related CFCs (using ORM relationships) must have the same data source.

Example

Art.cfc

```
<cfcomponent persistent="true" datasource="artgallery" table="Art">
  ...
</cfcomponent>
```

Author.cfc

```
<cfcomponent persistent="true" datasource="bookclub" table="author">

</cfcomponent>
```

ORM settings

The following are the data source-specific ORM settings for which you can specify string or struct values in the Application.cfc:

- `schema`
- `catalog`
- `dialect`
- `dbcreate`
- `sqlscript`
  For multiple data sources, a struct can be specified with data source name as the key and the appropriate setting as the value. If a string value is specified, it applies to the default data source of ORM.

Example 1

```
<cfset this.ormsettings.dbcreate={artgallery="dropcreate", bookclub="none"}>
```

Example 2

`<cfset this.ormsettings.dbcreate="dropcreate">` If multiple data sources are used for ORM, these settings apply to the default ORM data source.

Mapping using Hibernate mapping files

In multiple data source scenarios, the data source information must be provided in the CFC (and not in .hbmxml file).
Also, all CFCs used in one .hbmxml file must have the same data source.

Example

© 2014 Adobe Systems Incorporated. All rights reserved.

The following example illustrates two different entities using two different data sources. In this example, art.cfc and artist.cfc are related and therefore use the same data source.art.cfc

```
<cfcomponent persistent="true" table="art" datasource="cfartgallery">
<cfproperty name="ArtID" fieldtype="id" generator="native">
<cfproperty name="ArtName">
<cfproperty name="IsSold">
</cfcomponent>
```

artists.cfc

```
<cfcomponent persistent="true" table="artists" datasource="cfartgallery">
<cfproperty name="ArtistID" fieldtype="id">
<cfproperty name="FirstName">
<cfproperty name="LastName">
<cfproperty name="art" fieldtype="one-to-many" cfc="art" fkcolumn="ArtistID">
</cfcomponent>
```

authors.cfc

```
<cfcomponent persistent="true" table=authors datasource="cfbookclub">
<cfproperty name="AuthorID" fieldtype="id">
<cfproperty name="LastName">
<cfproperty name="FirstName">
</cfcomponent>
```

index.cfm

```
<cfoutput>Original Data<br></cfoutput>
<cfset artistObj = EntityLoad("artists", 1, true)>
<cfoutput>#artistObj.getArtistID()# | #artistObj.getFirstName()# |
#artistObj.getLastName()#<br></cfoutput>
<cfset artObj = artistObj.getart()>
<cfoutput>#artObj[1].getartname()# <br></cfoutput>
<cfset authorObj = EntityLoad("authors", 1, true)>
<cfoutput>#authorObj.getFirstName()#</cfoutput>
<cfoutput>#authorObj.getLastName()#</cfoutput>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

Introduction-Multiple data sources for ORM

You can use multiple data sources for ORM in ColdFusion applications. A multiple data source setup is useful in scenarios where your application has multiple modules that interact with each other.
Hibernate inherently supports single data source for a Hibernate configuration. To support multiple data sources, ColdFusion builds and manages multiple Hibernate configurations and SessionFactory objects, one for each data source in the application.

© 2014 Adobe Systems Incorporated. All rights reserved.

Usage scenario

Consider an application with the following three modules:

- HR
- Finance
- Sales
  Assume that all these modules have their own databases (and therefore separate data sources). But at the application-level, all the three modules have to interact with each other. A single data source makes it impossible to build the entire application using ORM. Building three separate applications is not advisable as the interaction between the applications is possible only using web services.
  If you use a multiple data source setup for ORM, all the three modules can be built in ORM. They can be part of the same application and the modules can interact with each other.

© 2014 Adobe Systems Incorporated. All rights reserved.

ORM Function enhancements

Multiple data source support impacts the following ORM functions:

ORMGetSession

Description

Returns the Hibernate session associated with the data source in the request. If ORM is not configured for this data source, it results in an exception. If data source is not specified, the Hibernate session of the default data source is returned.
Use this session object to call the APIs, which, otherwise, ColdFusion does not expose.
For information on session APIs, see: http://docs.jboss.org/hibernate/core/3.3/api/org/hibernate/Session.html

Function syntax

```
ormgetsession([datasource])
```

ORMCloseSession

Description

Closes the Hibernate session associated with the data source in the request. If you do not specify a data source, the Hibernate session associated with the default data source is closed.

Function syntax

```
ormclosesession([datasource])
```

ORMCloseAllSessions

Description

Closes all Hibernate sessions in the request.

Function Syntax

```
ormcloseallsessions()
```

History

ColdFusion 9 Update 1: Added this function

ORMFlush

Description

Flushes the Hibernate session associated with the data source in the request. ORMFlush flushes all pending CRUD operations in the request. Any changes made in the objects, in the current ORM session, are saved to the database. If you do not specify the data source, the Hibernate session associated with the default data source is flushed.

Function syntax

```
ormflush([datasource])
```

ORMFlushall

Description

© 2014 Adobe Systems Incorporated. All rights reserved.

Flushes all the current Hibernate sessions in the request.

Function syntax

```
ormflushall()
```

History

ColdFusion 9 Update 1: Added this function

ORMClearSession

Description

Clears the Hibernate session associated with the given data source.The function clears the first level cache and removes the objects that are not yet saved to the database.If you do not specify the data source, the Hibernate session associated with the default data source is cleared.

Function syntax

```
Ormclearsession([datasource])
```

ORMGetSessionFactory

Description

Returns the Hibernate Session Factory object associated with the data source. Results in an error if ORM is not configured for this data source. If you do not specify the data source, the Hibernate session factory object associated with the default data source is returned.
For information on Session API, go to the following URL: http://docs.jboss.org/hibernate/core/3.3/api/org/hibernate/SessionFactory.html

Function syntax

```
Ormgetsessionfactory([datasource])
```

ORMEvictQueries

Description

This method is used to evict the data of all the queries from the default query cache of the specified data source. If cache name is specified, then the data of all queries belonging to the cache region with the given cache name are evicted.If no data source is specified, the default query cache of the default data source is evicted.

Syntax

{{ORMEvictQueries([cachename])
ORMEvictQueries([cachename], datasource)}}

Parameter	Description
`cachename`	Name of the cache region that you want to evict.
`datasource`	Name of the data source whose cache you want to evict. If you do not specify the cache, the default query cache is evicted.

© 2014 Adobe Systems Incorporated. All rights reserved.

ORMExecuteQuery

Description

Executes a Hibernate Query Language (HQL) query.By default, this function works on ORM's default data source. To use this function for another data source, specify the data source key-value pair within the `queryoptions`.

Syntax

{{ORMExecuteQuery(hql, [params] [,unique])
ORMExecuteQuery(hql, [,unique] [, queryoptions])
ORMExecuteQuery(hql, params [,unique] [,queryOptions])}}

Parameters

Parameter	Description
Hql	The HQL query that has to be executed.
Params	Object parameter for the entity.
Unique	Specifies if the object parameter is unique.
Queryoptions	Key-value pair of options for the query.

Example

```
<cfset artistArr = ORMExecuteQuery("from Artists where artistid=1", true,
{datasource="cfartgallery"})>
<cfset countArray = ORMExecuteQuery("select count(*) from Authors", \[], false,
{datasource="cfbookclub"})>
```

#back to top

© 2014 Adobe Systems Incorporated. All rights reserved.

ColdFusion ORM search

Enhancements in ColdFusion 10 provide indexing and search capabilities for ColdFusion ORM. When you develop an application that uses ColdFusion ORM, the search feature facilitates full text search. You can load all the persistent entities that match your search criteria based on a query text.

Full text search is a two-step process that involves indexing the persistent entity and search based on the indexed information.

© 2014 Adobe Systems Incorporated. All rights reserved.

Indexing modes

Using ColdFusion ORM, you can perform indexing either automatically or in offline mode.

Auto-indexing

Indexing is performed every time an entity is added, revised, or deleted from the database.
To enable auto-indexing, set `ormsettings.searchenabled` to `true` in the Application.cfc.
Indexing is automatically done whenever an ORM entity is persisted, based on the configuration in CFCs. For details, see *Specify the ORM search settings* in Indexing the persistent entity.

Offline indexing

Indexing is performed manually with the help of ColdFusion functions. In this mode, indexing is performed in batches.
You may want to perform offline indexing in the following scenarios:

- Index all or some of the pre-existing data in a database.
- Avoid in-request indexing (default behavior) to minimize CPU load, and later perform indexing as batch operation.
  The function `ORMIndex` lets you perform offline indexing.

© 2014 Adobe Systems Incorporated. All rights reserved.

Indexing the persistent entity

Indexing stores indexable data of the persistent entity in index files. Search is performed on the indexed data. The configuration you specify in the component decides the indexable data.

Specify the ORM search index directory

You can specify the index directory (the one in which all persistent entities, of an application's indexable data, are saved) either at the server-level or application-level.

At server level

1. In the ColdFusion Administrator, go to Server Settings > Settings.
2. In the Settings page, specify the directory details in Specify the absolute path to store index files for ORM search.

> **Note**
>
> A directory is created for each application with the application name, in the path you specify.

At application level

Specify the index directory path as absolute or path relative to the current Application.cfc. If you specify the relative path, then it is resolved with respect to the current folder.

> **Note**
>
> If you do not specify a path, the index files are stored, by default, in *cfroot*/ormindex.

Search settings at application level

Specify the ColdFusion ORM search settings in the Application.cfc using the properties provided in the following table:

Property	Req./Opt.	Default	Description
`searchenabled`	Required if using ORM search	`false`	If `yes`, ORM search is enabled.
`search.autoIndex`	Optional	`false`	If `yes`, autoindexing is enabled.

© 2014 Adobe Systems Incorporated. All rights reserved.

search.indexDir	Optional		If specified, creates a folder by entity name in the file system where indexes for all entities are saved. The directory you set in the ColdFusion Administrator is used for this purpose.This folder has all index information stored.If not specified, creates the folder in the application root directory.
search.language	Optional	english	Specify the language that is used to index and search.

Example

```
component
{
 this.name = "ORM_Search";
 this.ormEnabled = "true";
 this.ormSettings.datasource = "ORM_Dummy";
 this.ormSettings.dbcreate = "dropcreate";
 this.ormsettings.searchenabled = "true";
 this.ormSettings.search.autoindex = "true";
 this.ormSettings.search.indexDir = "C:/ormindex";
 this.ormSettings.search.language = "English";
}
```

Search settings at the component level

Specify the following in the tag `cfcomponent`. The settings override the values in the Application.cfc.

Attribute	Req/Opt	Default	Description
indexable	Optional	false	If `true`, enables indexing for the component.
indexLanguage	Optional	english	Specify the language that is used to index and search.The value you set overrides the value defined in the Application.cfc.

© 2014 Adobe Systems Incorporated. All rights reserved.

autoIndex	Optional	true	If false, auto-indexing of CFC does not occur. That is, indexing occurs only in offline mode.

Search settings at the property level

Specify the following in the tag cfproperty. The settings override the values in the Application.cfc and cfcomponent.

Attribute	Req/Opt	Default	Description
indexable	Optional	false	If true, marks the column for indexing.Except in the case of PK and compositekey (which are indexed if any of the non-PK property is set to indexed), the default value is false.
indexTokenized	Optional	true	If true, the field text is broken into sub-keys for indexing.Applies only if indexed is set to true.
indexFieldName	Optional	value of the attribute name	Specifies the field name that is used in search query while indexing and performing search.
indexBoost	Optional		Used to prioritize the search results. This is a numeric value.Results from this column appears above others in the query result. Higher the boost, the more the priority.

© 2014 Adobe Systems Incorporated. All rights reserved.

`indexStore`	Optional	`false`	The values are `true`, `false`, and `compressed`. • `true`: Stores the value in the original form, without tokenizing. • `false`: Does not store the value. • `compressed`: Stores the original value in a compressed form, based on Lucene implementation.
`indexLanguage`	Optional	`english`	Specify the language that is used to index and search.The value overrides the value defined in `cfcomponent` and the Application.cfc.

Example

Empoyee.cfc

```
component persistent="true" indexable="true" indexlanguage="English"
table="Employee"
{
 property name="id" fieldtype="id" ormtype="int" generator="native";
 property name="firstname" indexable="true" indexstore="yes" indexboost="5.0";
 property name="lastname" indexable="true" indextokenize="true"
indexfieldname="surname";
 property name="description" indexlanguage="Greek";
 property name="dept" fieldtype="many-to-one" cfc="dept" indexable="true"
indexfieldname="department" notnull="false";
}
```

© 2014 Adobe Systems Incorporated. All rights reserved.

Other enhancements

Logging

Enabling request debugging in ColdFusion Administrator helps logging of HQL queries. To do this,

1. In the ColdFusion Administrator, go to Debugging & Logging > Debug Output Settings.
2. Check the option Enable Request Debugging Output.
    - Logging

© 2014 Adobe Systems Incorporated. All rights reserved.

Performing the search

Using the ORM search functions, you can perform search in both online and offline mode in

- An entity
- An indexable field in an entity
- Common fields across entities
- Multiple fields in an entity
- Multiple fields in multiple entities

© 2014 Adobe Systems Incorporated. All rights reserved.

Search based on indexed information

Search can be performed using the functions provided in ColdFusion. Search can either be a

- Lucene query search
- Full text search where ColdFusion ORM generates the Lucene query

Specify the ORM search settings

You have to specify the ORM search settings at the application level, component level, or property level.

© 2014 Adobe Systems Incorporated. All rights reserved.

ColdFusion and HTML 5

ColdFusion 10 comes to you with a powerful set of HTML 5 features. Since being the future standard of interaction and communication on the Web, ColdFusion 10's HTML 5 features make your web experience richer and easier than before.

- Using ColdFusion WebSocket
- Media Player enhancements-Developing guide
- Displaying geolocation - Developing guide
- Client-side charting-Developing guide

#back to top

© 2014 Adobe Systems Incorporated. All rights reserved.

Using ColdFusion WebSocket

- WebSocket Enhancements
- Browsers and fallback
- ColdFusion and WebSocket
- How is WebSocket different from conventional communication models
- WebSocket communication models
- What is WebSocket
- Using WebSocket to broadcast messages
- Using WebSocket in point-to-point communication
- Error handling in ColdFusion WebSocket
- Using ColdFusion Administrator for WebSocket Configurations
- Browser support for WebSocket

© 2014 Adobe Systems Incorporated. All rights reserved.

WebSocket Enhancements

ColdFusion 10 implemented WebSocket by providing a messaging layer for the WebSocket protocol, which you can control using CFML and JavaScript. The messaging layer has features that make WebSocket reliable and scalable. Using WebSocket, you can develop real-time applications for stock, charting, online gaming, social networking, dashboard for various purposes, and monitoring.

As you are already aware of, WebSocket connections use the standard HTTP ports (80 and 443) for sending and receiving data packets. Due to this reason, WebSockets are mostly firewall-friendly since almost all firewall configurations allow net traffic on the HTTP ports. Hence the HTML5 WebSockets do not require new hardware to be installed, or new ports to be opened on internal networks.

When there is no proxy server, reverse proxy server, or a firewall between the browser and the ColdFusion Server, the WebSocket connection works perfectly, as long as both the server and the client understand the WebSocket protocol. However, most of the times, there will not be any direct connection between the server and the client. ColdFusion 11 has introduced the proxy support for WebSocket. There is a new proxy module (that runs inside IIS and Apache Web Server) that can intercept the ColdFusion WebSocket requests and redirect the requests to the ColdFusion Server.

What is new

ColdFusion 11 supports the following new features:

- **Proxy support** - Support is now provided for setting up a proxy server for using ColdFusion WebSockets.
- **SSL support** – Support is now provided for enabling SSL for ColdFusion WebSockets.
- **Cluster support** – Support is now provided for enabling the cluster support for ColdFusion WebSockets.

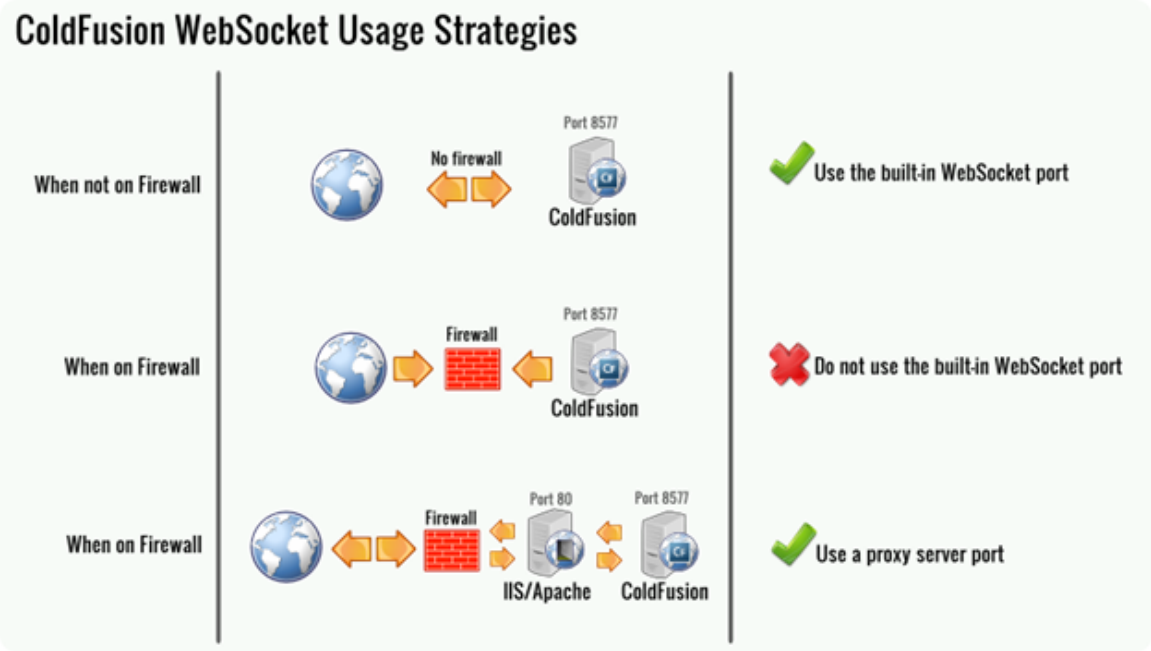
Using ColdFusion WebSockets

The support for WebSocket was available in the previous version of ColdFusion. What is new in ColdFusion 11 is the support for WebSocket proxy.

Consider the following usage strategies for WebSocket proxy:

- **Strategy 1: When you don’t have a firewall configured** – You CAN use the WebSocket proxy
- **Strategy 2: When you have a firewall configured** – You MUST use the WebSocket proxy

© 2014 Adobe Systems Incorporated. All rights reserved.



Strategy 1 - Using the built-in WebSocket server (ColdFusion 10)

For more information, see, https://learn.adobe.com/wiki/display/coldfusionen/Using+ColdFusion+WebSocket

Also, see:

- Browsers and fallback
- ColdFusion and WebSocket
- How is WebSocket different from conventional communication models
- WebSocket communication models
- What is WebSocket
- Using WebSocket to broadcast messages
- Using WebSocket in point-to-point communication
- Error handling in ColdFusion WebSocket
- Using ColdFusion Administrator for WebSocket Configurations
- Browser support for WebSocket

Strategy 2 - Using a proxy server for WebSockets (ColdFusion 11)

ColdFusion 11 has introduced the proxy support for WebSocket. There is a new proxy module (that runs inside IIS and Apache Web Server) that can intercept the ColdFusion WebSocket requests and redirect the requests to the ColdFusion Server.

You can use one of the following external servers along with the ColdFusion Server:

- IIS 8 or higher running on Windows 8 or Windows Server 2012
- Apache 2.2 or higher

The following steps enable you to quickly configure a WebSocket Proxy server for ColdFusion.

Step 1 – Configuring the external server

Before you begin setting up a WebSocket proxy, you need to configure the external server to support the

© 2014 Adobe Systems Incorporated. All rights reserved.

WebSocket protocol.

Configuring IIS

For enabling WebSocket protocol for IIS, see this document.

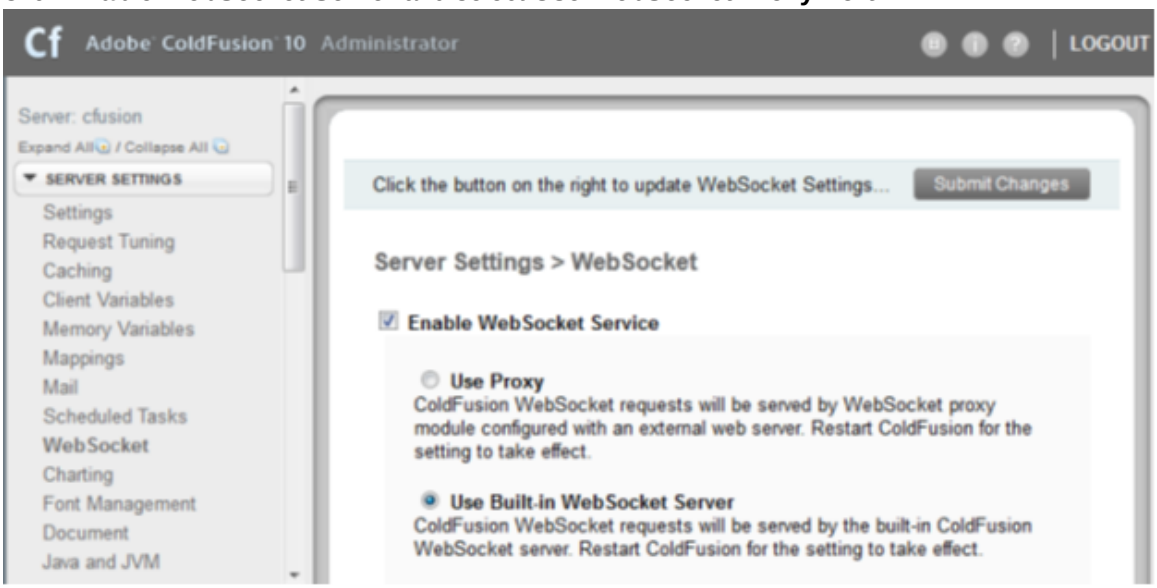
Configuring Apache Server

No configuration required.

Step 2 – Enabling WebSocket Proxy

You need to enable the WebSocket Proxy port at the server-level through the ColdFusion Administrator. Perform the following steps:

1. Go to the ColdFusion Administrator page.
2. Click Server **Settings > Web Socket**
3. Click **Enable WebSocket Server** and select **Use WebSocket Proxy Port**:



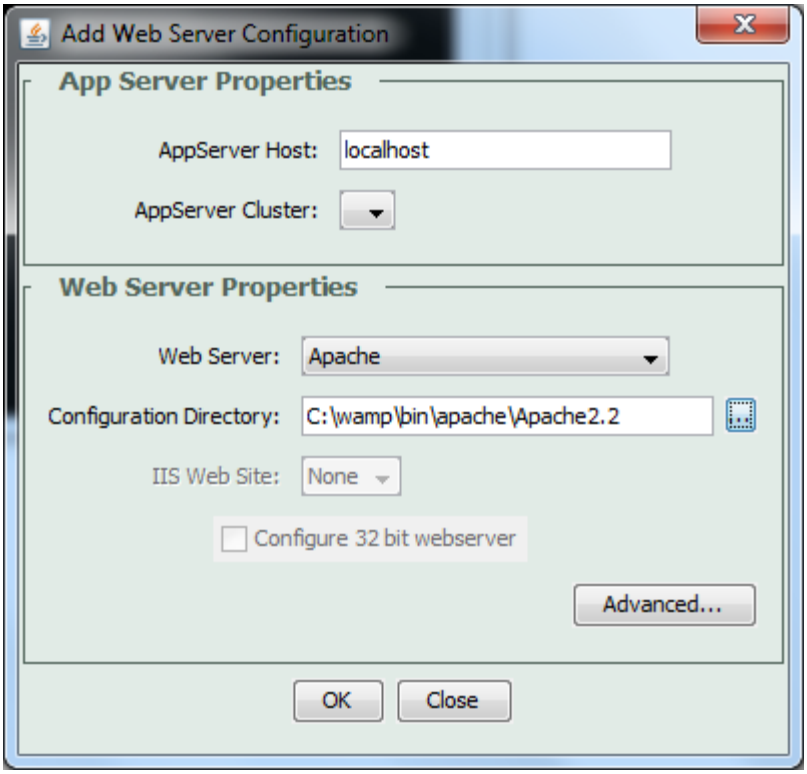
Step 3 – Adding the external server

You need to configure the external Web Server connector with the ColdFusion Server using the wsconfig tool.

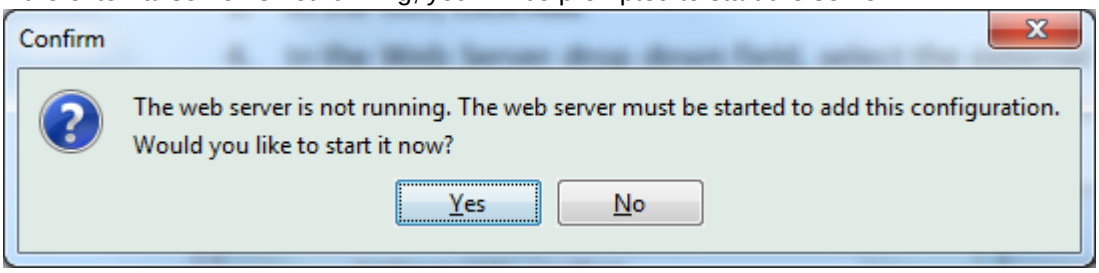
Perform the following tasks:

1. Go to <*CF_INSTALL_HOME*>/cfusion/runtime/bin
2. Run the wsconfig tool
3. In the GUI, click Add
4. In the **Web Server** drop down field, select the external server
5. In the **Configuration Directory** field, browse and select the path to the external server's configuration directory:
6. Click OK

© 2014 Adobe Systems Incorporated. All rights reserved.



7. If the external server is not running, you will be prompted to start the server:



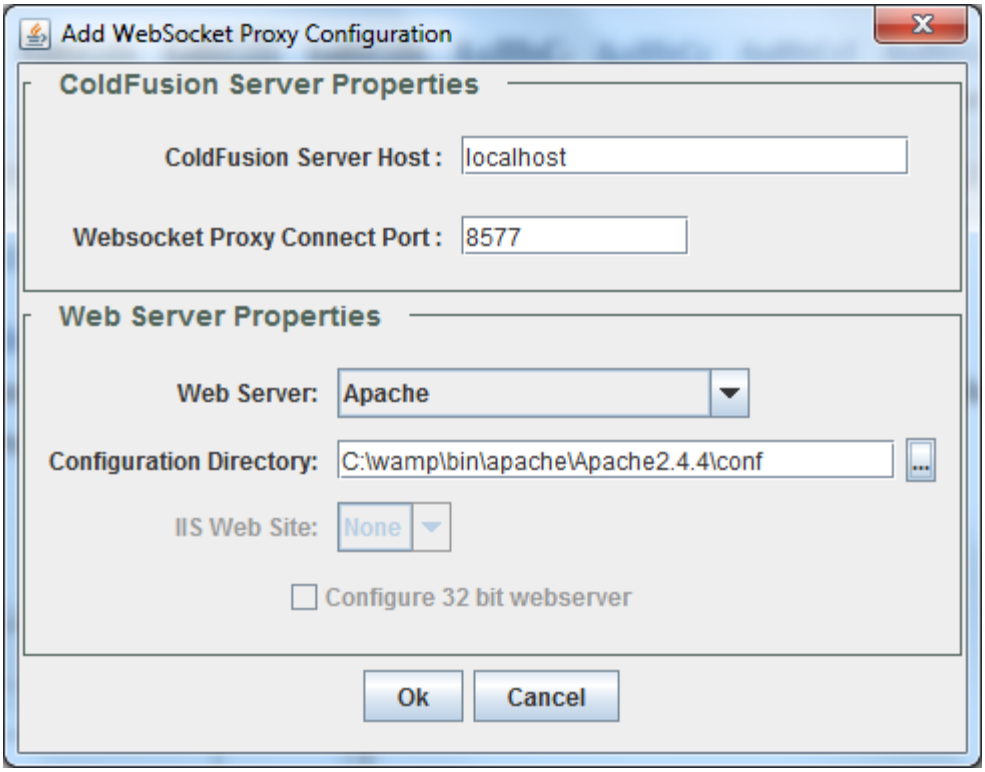
8. Click Yes to start the external server
9. Click Exit

Step 4 - Setting up the WebSocket Proxy Server

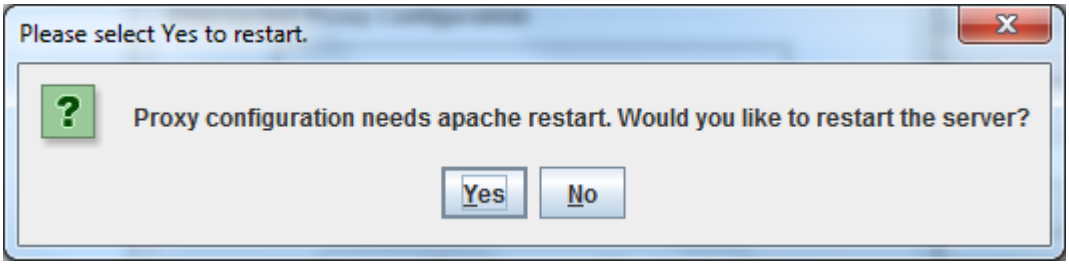
ColdFusion 11 has introduced a new configuration tool, wsproxyconfig, which will allow you to quickly setup a WebSocket Proxy. Perform the following tasks:

1. Go to <*CF_INSTALL_HOME*>/cfusion/bin
2. Run the wsproxyconfig tool
3. In the GUI, click Add
4. In the **Web Server** drop down field, select the external server
5. In the **Configuration Directory** field, browse and select the path to the external server's configuration directory:

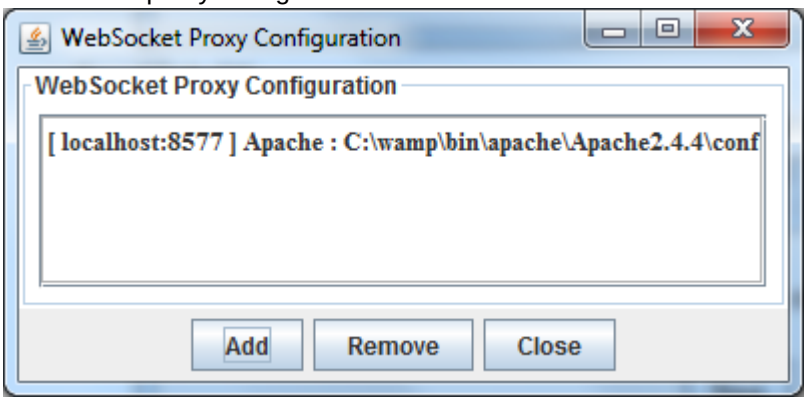
© 2014 Adobe Systems Incorporated. All rights reserved.



6. Click OK
7. If the external server is not running, you will be prompted to start the server:



8. Confirm the proxy configuration:



9. Click Close

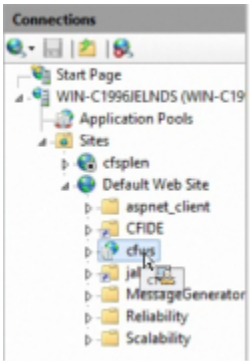
Step 5 – Verify the proxy configuration

After you have successfully configured the WebSocket Proxy, verify if the required proxy files are created at <*CF_INSTALL_HOME*>/config/wsproxy/1

You can also manually modify the config.ini file located at <*CF_INSTALL_HOME*>/config/wsproxy/1.

© 2014 Adobe Systems Incorporated. All rights reserved.

Also, a new application, **cfws** will be made available under the application root of the external servers:



The <cfwebsocket> tag supports a new attribute:

- secure – (Optional) If only secure port (SSL) is configured for the WebSocket server (for web service communication), all the WebSocket communication happens over the secure channel automatically. But in the case if both ports (non-SSL and SSL) are defined , if you require the communication to be set through a secured TCP socket, you must set secure to true. Default is false.

ColdFusion 11 has been enhanced to provide enterprise-level capabilities like real-time message broadcasting across cluster of nodes and SSL support for secure websocket communication (WSS protocol).

Cluster Support

If your application is running across various nodes and on publishing a message to a single node, you want the same to be broadcasted across all the nodes, you can use this feature.

For example, Node1 has two clients (C1 and C2) subscribed to the 'stock' channel. Similarly Node2 has two clients (C4, C6) and Node3 has two clients (C8, C10) also subscribed to the 'stock' channel. When client C1 tries to publish a massage on this 'stock' channel, rather than just sending this message to just C1 and C2 only, the clients from other nodes C4, C6, C8, and C10 must also receive this message.

On using this feature, you can publish a message to all the subscribers of the channel across various nodes of the cluster by enabling cluster. To enable the feature, choose the cluster option and define the multicast port to broadcast node up and node down (events).

> The cluster support is not available in the standard edition.

JavaScript Functions

The following JavaScript functions were enhanced to auto-support the cluster functionality:

Functions	Argument	Return Type
publish	("channelname", messageBody, {filterOptions}	boolean
getSubscriberCount	(“channelname”)	boolean

© 2014 Adobe Systems Incorporated. All rights reserved.

invokeAndPublish	("channelName", cfcName, cfcMethod, [function_parameter] ,{custom_options})	boolean

publish

In ColdFusion 10, this function was enabled to send messages to a specific channel based on the filter criteria. In ColdFusion 11, this function is enhanced to send message to all nodes in a cluster by default.

```
Publish("channelname ", messageBody, {filterOptions});
```

getSubscriberCount

In ColdFusion 10, this function was enabled to get subscriber count from a specific channel based on the filter criteria. In ColdFusion 11, this function is enhanced to fetch the subscriber count from all nodes by default.

```
getSubscriberCount(“channelname”)
```

invokeAndPublish

Use this function for message publishing and CFC invocation to be executed as separate thread.

```
invokeandPublish("channelname ", cfcName, cfcMethod, [function_parameter]
,{custom_options})
```

In-built ColdFusion functions

To further facilitate cluster support, the following in-built functions are enhanced for ColdFusion 11:

WSPublish

Trigger a message publish from the server without any input from client. A new parameter "Clustered" has been added to facilitate message broadcasting to all subscribers connected to multiple nodes.

```
wsPublish(“channel name”, messageBody [clustered])
```

Or

```
wsPublish(“channel name”, messageBody , {filterOptions},[cluster])
<cfset wsPublish("publishdemochannel","Welcome to publishdemo")/>
```

If the "clustered" parameter is not defined, then the default value will be taken based on the cluster

© 2014 Adobe Systems Incorporated. All rights reserved.

<Enable/Disable> setup.

WSGetSubscribers

Fetches the subscriber information from all the configured nodes. A new parameter "Clustered" has been added to fetch subscriber information from all configured nodes.

```
WSGetSubscribers("channel")
```

Or

```
WSGetSubscribers ("channlName" ,[clustered]);
<cfset wsgetSubscribers(“stocks”, true)>
```

If the "clustered" parameter is not defined, then the default value will be taken based on the cluster <Enable/Disable> setup.

WSGetAllChannels

Fetches the sub-channel information from all the configured nodes. A new parameter "Clustered" has been added to fetch sub-channel information from all configured nodes.

```
WSGetAllChannels ("channel", clustered)
<cfset wsgetAllChannels(“stocks” true,)>
```

If the "clustered" parameter is not defined, then the default value will be taken based on the cluster <Enable/Disable> setup.

Secure WebSocket communication (WebSocket over SSL)

The Websocket screen (*Administrator Console* > *ColdFusion Administrator* > *Server Settings* > *WebSocket*) has been enhanced in ColdFusion 11 with the following options:

- Enable SSL
- SSL port
- KeyStore File location
- KeyStore password

Option	Description
Enable SSL	Enables Secure Sockets Layer (SSL) for running WebSocket over SSL.
KeyStore File location	The location of keyStore in server’s file system. For example, C:\OpenSSL\bin\keystore.jks.

© 2014 Adobe Systems Incorporated. All rights reserved.

KeyStore password	Password for the keyStore.
SSL port	Default port is 8543.

To enable Web socket cluster from the Administrator node, select *Server Setting* > *Web Socket*. In the Web Socket screen, select the *Enable Web Socket Cluster* check box and enter the Multicast Port details. In the Multicast Port field, the default Multicast Port 45566 is displayed. Change the port details if required. Make sure the multicast port value is same across all the nodes in the cluster.

© 2014 Adobe Systems Incorporated. All rights reserved.

Browsers and fallback

ColdFusion leverages the browser-supported HTML5 native WebSocket to provide server controlled client-to-client communication. If a browser *currently* does not have HTML5 support or the native WebSocket support, ColdFusion uses Flash player as the fallback option. Fallback to Flash is smooth and automatic.See also, Browser support for WebSocket.The following table provides the fallback plan for ColdFusion WebSocket:

Scenario	What happens
Native WebSocket support available	Establishes a WebSocket connection.
Native WebSocket unavailable, but Flash is installed	Automatically falls back to Flash.
No native WebSocket support or Flash installation	Sends a message indicating that the connection is not successful.To proceed, either move on to a compliant browser or install Flash.

© 2014 Adobe Systems Incorporated. All rights reserved.

ColdFusion and WebSocket

ColdFusion 10 implements WebSocket by providing a messaging layer for the WebSocket protocol, which you can control using CFML and JavaScript. The messaging layer has features that make WebSocket reliable and scalable. Using WebSocket, you can develop real-time applications for stock, charting, online gaming, social networking, dashboard for various purposes, and monitoring.

© 2014 Adobe Systems Incorporated. All rights reserved.

How is WebSocket different from conventional communication models

WebSocket is based on HTML 5 server Push Technology. The communication request is initiated by the publisher or central server following a publish/subscribe model. This is unlike the request/response model that HTTP relies on, which has drawbacks such as request time-out and browser refresh. In the case of WebSocket, a client subscribes to various information channels. If new content is available in the channel, it is pushed to the clients. This method has the following advantages:

- Simultaneous bi-directional communication
- Lesser browser reloads
- Ease in creating applications that handle real-time data

© 2014 Adobe Systems Incorporated. All rights reserved.

WebSocket communication models

ColdFusion WebSocket offers the following two communication models:

- **Broadcast:** Follows a subscriber/publisher model where hosts include multiple clients and server. Publishing is possible in either of the following ways:
- From client-side (by way of the server)
- Initiation from the server without client involvement
- **Point-to-point:** A bi-directional communication between a specific client and server. This model does not involve any other hosts/multiple clients.

© 2014 Adobe Systems Incorporated. All rights reserved.

What is WebSocket

WebSocket is a protocol for two-way communication with a remote host over TCP protocol exposed through JavaScript interface in HTML 5 compliant browsers. WebSocket facilitates communication between hosts in both the directions simultaneously.

© 2014 Adobe Systems Incorporated. All rights reserved.

Using WebSocket to broadcast messages

- Defining a communication channel
  - Description
- Creating WebSocket object using cfwebsocket
  - Syntax
  - Attribute
  - Usage
  - Example
- Using the WebSocket JavaScript functions
  - Example
- Scenarios: Subscribing and publishing
- Scenarios: Getting subscriber list
- Using channel listener functions
  - A channel listener can be:
  - Setting up user-defined channel listener
  - How the listener functions receive/provide information
  - Channel listener functions
- Using channel-specific message handler
  - Example
- Specifying custom information
  - Example 1: Custom information using subscribe
  - Example 2: Custom information using publish
- Using selectors
- Specifying WebSocket authentication details
- onWSAuthenticate
  - Description
  - Syntax
  - Parameters
  - Example
- Enabling Single sign-on in WebSocket
  - Example
- Server- side APIs
- Interpreting the Server response

The following procedures let you create a broadcast setup:

Step	Procedure	Description
1	Define communication channels in the Application.cfc	Client can subscribe or publish to a pre-defined set of channels. To listen to the channel, first register it at the application level.

© 2014 Adobe Systems Incorporated. All rights reserved.

2	Create WebSocket object using the `cfwebsocket` tag in your CFM template	Creates WebSocket connection with all required properties automatically, and wraps it in a JavaScript object that can invoke various JavaScript methods. The tag internally identifies the IP address and port of the server.
3	Implement the business logic using the JavaScript functions provided by the WebSocket object	On runtime, you can make use of the supported JavaScript functions for example, `subscribe` or `publish` to achieve your business goals.

Defining a communication channel

Specify the following settings in the `Application.cfc` to define the communication channel:

```
component
{
this.name="testapp";
this.wschannels=[{name = channelName,cfclistener= channel_listener_CFC}];
}
```

Description

- Specify the channel name.

  > **Note**
  >
  > Though you can use any number of sub-channels, you do not specify them. They are dynamically created.

- If necessary, specify the ColdFusion channel listener CFC, the component responsible for creating the channel. If not specified, the default `ChannelListener.cfc`, available in `wwwroot\CFIDE\websocket` directory is called.For further details, see *Using channel listener functions* below.

Creating WebSocket object using cfwebsocket

The new ColdFusion tag {{cfwebsocket }}lets you create the WebSocket object in your CFM template. The tag creates a reference to the WebSocket JavaScript object at the client-side.

Syntax

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cfwebsocket
 name="websocketName"
 onMessage="JavaScript function name"
 onOpen="JavaScript function name"
 onClose="JavaScript function name"
 onError="JavaScript function name"
 useCfAuth=true|false
 subscribeTo="channel_list">
```

Attribute

Attribute	Req/Opt	Descriptions
`name`	Required	The name of the WebSocket object. This is the reference to the JavaScript objects that are used to call WebSocket JavaScript functions.
`onMessage`	Required	The JavaScript function that is called when the WebSocket receives a message from the server. The function requires one argument.
`onOpen`	Optional	The JavaScript function that is called when the WebSocket establishes a connection.The function does not require any arguments.
`onClose`	Optional	The JavaScript function that is called when the WebSocket closes a connection.The function does not require any arguments.
`onError`	Optional	The JavaScript function that is called if there is an error while performing an action over the WebSocket connection. The function requires one argument.

© 2014 Adobe Systems Incorporated. All rights reserved.

usecfAuth	Optional	If set to true (default), authentication is not required for WebSocket connection (provided they have already logged in to the application). This is the default value.If false, to authenticate, you have to use the javaScript function authenticate.
subscribeTo	Optional	Comma-separated list of channels/subchannels to subscribe to. You can specify any or all channels set in the Application.cfc.

Usage

By specifying the attribute name, for example, mycfwebsocketobject (as in the following example), you create a JavaScript object. The JavaScript object can be used to call various JavaScript functions which you use for the WebSocket communication.

Example

In the following example,

- The user is automatically subscribed to the channel stocks.
- You have specified the channel name in the Application.cfc.
- Uses the default channel listener (as you have not specified a custom channel listener).
- In the Index.cfm, you specify the channels to which the user can automatically subscribe to (using the attribute subscribeTo) and also define the message handler.
  Application.cfc

```
component
{
        this.name="websocketsampleapp1";
        this.wschannels=[{name="stocks"}];
}
```

  Index.cfm

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<script type="text/javascript">
  /* mymessangehandler function recieves all messages that comes via
WebSocket.   This function requires one argument.
  */

       function mymessagehandler(messageObject)
       {
   //JavaScript messageobject is converted to a string.
   var message = ColdFusion.JSON.encode(messageObject);
           var txt=document.getElementById("myDiv");
           txt.innerHTML +=message  +"<br>";
       }
</script>
<cfwebsocket name="mycfwebsocketobject"  onmessage="mymessagehandler"
subscribeto="stocks" >
<cfdiv id="myDiv"></cfdiv>
```

The code creates a JavaScript WebSocket object named `mycfwebsocketobject`. If the server sends a message, it calls `mymessagehandler` function.Since you have specified `subscribeTo` in the `cfwebsocket` tag, the WebSocket object automatically subscribes you to the channel `stocks`.

Using the WebSocket JavaScript functions

After you create a JavaScript WebSocket object using the `cfwebsocket` tag, you can use it to call the following JavaScript functions:

Function	Description	Syntax	Parameters

`authenticate`	Authenticates the user to a specific channel.	`authenticate(`*`id, password`*`)`
		For example,
		`mycfwebsocketobject.authenticate("Ben", "password");`
			• `id`: The user-ID against which a user is authenticated.
			• `password`

© 2014 Adobe Systems Incorporated. All rights reserved.

`subscribe`	Lets the client subscribe to a specific channel. You can send additional information such as age of the user (in the form of `custom_header` JavaScript object) to the server. This information can be used to take a decision on whether to let the client subscribe or not.	`subscribe(channel[, custom_header][, messageHandler])`
		For example,
		`mycfwebsocketobject.subscribe(channel);`
			• `channel`: Name of the channel/sub-channel to which the client wants to subscribe.
			• `custom_header`: Optional. Passed as key-value pairs. You can also use ColdFusion keyword {{selector }}and specify a condition that can be used as a filter criteria, for example, `mycfwebsocketobject.subscribe(channelname,{age : 25,selector:"symbol eq 'adbe' and value gt 20"}).`
			• `messageHandler`: Optional. Specifies the channel-specific message handler. If a message handler is specified, data sent from the server (to the channel) is passed to the specific message handler rather than to tag's `onMessage` attribute (that you specified while creating the WebSocket object). For more details, see *Using channel-specific message handler* below .

© 2014 Adobe Systems Incorporated. All rights reserved.

`publish`	Publishes the message to the channel. If the message is not available use the function `invokeAndPublish`.Message can be passed in any JavaScript data type (such as array, string, int, or a key-value pair) as a JavaScript object.You can send additional information (in the form of custom_header). The information can be used to decide if to allow the client to publish, for example Only platinum members have rights to publish.	`publish(channel, message [, custom_header])`
		
		For example,
		
		`mycfwebsocketobject.publish("stocks",msg);`
			• `channel`: Name of the channel/sub-channel to which the message has to be published.
			• `message`: The message object that has to be published. Message is published to clients subscribed to the specific channel.
			• `custom_header`: Optional. Passed as key-value pairs. You can also use ColdFusion keyword {{selector }}and specify a condition that can be used as a filter criteria, for example, `mycfwebsocketobject.publish("mychannel", message, {item:phone, selector:"itemcode eq HTC HD"});`.

© 2014 Adobe Systems Incorporated. All rights reserved.

`invokeAndPublish`	Invokes a specified method in a CFC file to generate the message that has to be published. Used in scnearios where you have raw data (that needs to be processed) to create a message.	`invokeAndPublish(channel, cfcName, functionName [, argumentsArray] [, custom_header])`
		
		For example,
		
		`mycfwebsocketobject.invokeAndPublish(ch, "employee", "getEmployeeDetails", ["eid_1"]);`
			• `channel`: The name of the channel/sub-channel to which a message has to be published.
			• `cfcName`: The CFC that contains the method that is invoked for the message. Users can pass the `cfcName` as fully qualified path or path relative to the application root.
			• `functionName`: The name of the function in CFC that has to be invoked for the message.
			• `argumentsArray`: A JavaScript array of arguments that has to be passed to invoke the function on the given CFC.
			• `custom_header`: Optional. Passed as key-value pairs. You can also use ColdFusion keyword {{selector }}and specify a condition that can be used as a filter criteria.

© 2014 Adobe Systems Incorporated. All rights reserved.

`getSubscriberCount`	Provides the number of subscribers for a specific channel.	`getSubscriberCount(channel)`
		
		For example,
		
		`mycfwebsocketobject.getSubscriberCount("stocks");`
			• {{channel: }}Name of the channel/sub-channel whose subscriber count is sought.
			
			Note
			
			Supports inline callback functionality to receive asynchronous response.

`getSubscriptions`	Provides all channels the client is subscribed to, as a JavaScript array.	getSubscription()	**Note** Supports inline callback functionality to receive asynchronous response.
`openConnection`	Opens a WebSocket connection (if not already open).	`openConnection()`	

© 2014 Adobe Systems Incorporated. All rights reserved.

`isConnectionOpen`	Verifies whether the WebSocket connection is open. Returns `true` if the connection is open.	`isConnectionOpen()`	
`closeConnection`	Closes a WebSocket connection.	`closeConnection()`	

`unsubscribe`	Unsubscribes a client from the specified channel. After the client unsubscribes, the messages in the channel are not published to the client.	`unsubscribe(channel)`
		For example,
		`mycfwebsocketobject.unsubscribe(channel);`
			• {{channel: }}Name of the channel from which to unsubscribe.

Note

When you use these functions, a boolean value indicating the status of the call is returned. But the result of the call is acknowledged at the message handler side. You can categorize the responses at the message handler using the request type token as shown in the following example: `if (token.reqType =="getSubscriberCount)`.

Example

The following example illustrates how to use various JavaScript functions.Application.cfc

```
component
{
        this.name="websocketsampleapp3";
        this.wschannels=[{name="stocks"}];
}
```

Index.cfm

```
<script>
        //messagehandler recieves all the messages from websocket
        function mycbHandler( messageobj)
```

© 2014 Adobe Systems Incorporated. All rights reserved.

```
{
       var message = ColdFusion.JSON.encode(messageobj);
    var txt=document.getElementById("myDiv");
    txt.innerHTML +=message  +"<br>";
}

//openhandler is invoked when socket connection is
function openHandler()
{
       var txt=document.getElementById("myDiv");
    txt.innerHTML +="open Handler invoked <br>";
}

function subscribeMe()
{
       var channelname = document.getElementById("channelname").value;
       mywsobj.subscribe(channelname);
}
function getSubscribers()
{
       var channelname = document.getElementById("channelname").value;
       mywsobj.getSubscriberCount(channelname);
}
function unsubscribe_Me()
{
    var channelname = document.getElementById("channelname").value;
    mywsobj.unsubscribe(channelname);
}
function publishClient()
{
       var channelname = document.getElementById("channelname").value;
       var message =  document.getElementById("msg").value;
mywsobj.publish(channelname,message);
}
function get_Subscriptions()
{
    mywsobj.getSubscriptions();
}
function invokenpublish()
{
       cfcname = document.getElementById("cfcname").value;
       fnname = document.getElementById("fnname").value;
       channelname = document.getElementById("channelname").value;
       mywsobj.invokeAndPublish(channelname, cfcname, fnname);
}
function invokefn()
{
       cfcname = document.getElementById("cfcname").value;
       fnname = document.getElementById("fnname").value;
       channelname = document.getElementById("channelname").value;
       mywsobj.invoke(cfcname, fnname);
}
function opensocket()
{
       var txt=document.getElementById("myDiv");
       txt.innerHTML+="opening socket"+"<br >";
       x=mywsobj.openConnection();
}
function stopsocket()
```

© 2014 Adobe Systems Incorporated. All rights reserved.

```
        {
                var txt=document.getElementById("myDiv");
                txt.innerHTML+="closing socket"+"<br >";
                x=mywsobj.closeConnection();
        }
        function checksocket()
        {
                var x=mywsobj.isConnectionOpen();
                var txt=document.getElementById("myDiv");
                txt.innerHTML+=x+"<br >";
        }
</script>
<form name="f">
<!---Define JS websocket object name and messagehandler and openhandler  --->
<cfwebsocket name="mywsobj" onMessage="mycbHandler" onOpen="openHandler"/>
<br> Subscribe to:
<input  id="channelname" name="channelname" type="text" value="stocks" >
<input  id="stocksSubscribe" type="button" value="stocksSubscribe"
onclick="subscribeMe();">
<input  id="unsubscribeMe" type="button" value="unsubscribeMe"
onclick="unsubscribe_Me();">
<input  id="getSubscribersCF" type="button" value="getSubscribersCF"
onclick="getSubscribers();">
<input  id="getSubscriptions" type="button" value="getSubscriptions"
onclick="get_Subscriptions();">
<br>
Message :<input id="msg" type="text" >
<input  id="publishMe" type="button" value="publishMe" onclick="publishClient();">
<br>
CFC Name: <input  id="cfcname" name="cfcname" type="text" value="invokeandpublish" >

Function Name: <input  id="fnname" name="fnname" type="text" value="publishall" >
<input  id="invoke_publish" type="button" value="invoke_publish"
onclick="invokenpublish();">
<input  id="invoke" type="button" value="invoke" onclick="invokefn();">
<br>
<input  id="stop" name ="Close" type="button" value ="stop" onclick="stopsocket()" >

<input  id="open" name ="Open" type="button" value ="open" onclick="opensocket()" >
<input  id="check" name ="Check" type="button" value="check" onclick="checksocket()"
>
<br>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<div id="myDiv"></div>
</form>
```

invokeandpublish.cfc

```
component
{

        public function publishall()
        {
                return "All Clients";
        }
}
```

Scenarios: Subscribing and publishing

If the users subscribe to a parent channel, automatically they subscribe to all subchannels. If users subscribe to a child channel, they are not subscribed to its parent channel.

The following table elaborates the conditions.

The table shows four clients Client 1, Client 2, Client 3, and Client 4 subscribed to the following channels: Stocks, Stocks.Finance, Stocks.Banks, and Stocks.Finance.Adobe.

Client	Channel subscribed to	Receives messages when publishing to Stocks	Receives messages when publishing to Stocks.Finance	Receives messages when publishing to Stocks.Banks	Receives messages when publishing to Stocks. Finance.Adobe
Client 1	Stocks	Yes	Yes	Yes	Yes
Client 2	Stocks.Finance	No	Yes	No	Yes
Client 3	Stocks.Banks	No	No	Yes	No
Client 4	Stocks.Finance. Adobe	No	No	No	Yes

Scenarios: Getting subscriber list

The following table explains the conditions on which you receive the subscriber list when you use the function `getSubscribers`.

Assume that there are four clients. Each client is subscribed to only one of the following channels (and no client is subscribed to the same channel): Stocks, Stocks.Finance, Stocks.Banks, and Stocks.Finance.Adobe.

© 2014 Adobe Systems Incorporated. All rights reserved.

Function getSubscribers called with the channel	Returns the ID of clients subscribed to the following channels
Stocks	Stocks
Stocks.Finance	Stocks.Finance and Stocks
Stocks.Banks	Stocks.Banks and Stocks
Stocks.Finance.Adobe	Stocks.Finance.Adobe, Stocks.Finnance, and Stocks

Using channel listener functions

Channel listener is a set of functions that decide the folow of messages to a channel. When a WebSocket object's JavaScript function is called, in turn functions in the channel listener are called. For example, when you call the `subscribe` function, it calls the `allowSubscribe` function from the listener CFC.

A channel listener can be:

- **Default channel listener:** Used if no channel listener is specified while defining the channel in the Application.cfc.
- **User-defined channel listener:** Used if a channel listener CFC is specified while creating a channel. Update the methods in the channel listener file to implement a specific business logic. Specify a custom channel listener to restrict subscription to a channel or right to publish. You can also use custom channel listener to format messages and decide if the client is eligible to receive message.

Setting up user-defined channel listener

- (You must) Extend the default listener present in webroot/cfide/websocket/ChannelListener as shown in the following code:`component extends="CFIDE.websocket.ChannelListener"`
- A user-defined channel listener CFC must be in the same directory or subdirectory of the application.
- You can only override the functions specified in the default listener CFC to implement the business logic. All custom functions are ignored.

How the listener functions receive/provide information

The functions in the listener receive or provide information in the following three structs. You can use these three structs to add custom information.

- **clientInfo**: Information about the client. The following are the values:
  - `channelName`
  - `clientId`: A unique client ID.It can be used for setting additional properties, such as roles.
  - `connectionTime`: Connection time as date object.
- **subscriberInfo**: A struct that contains custom options passed in the subscribe request. Also contains the object `ConnectionInfo` as subkey `connectionInfo`.
- **publisherInfo**: A struct that contains custom options passed in the publish request. Also contains the object `ConnectionInfo` as subkey `connectionInfo`.

The following table lists the WebSocket JavaScript functions and the corresponding function in the channel listener CFC:

WebSocket object JavaScript function	Function(s) called in the channel listener CFC

© 2014 Adobe Systems Incorporated. All rights reserved.

subscribe	allowSubscribe
unsubscribe	afterUnsubscribe
publish	allowPublish
invokeAndPublish	allowPublish
getSubscribers	None
getSubscriptions	None
invoke	None
openConnection	None
isConnectionOpen	None
closeConnection	None

Channel listener functions

The following table lists the functions in the listener file:

Function	Descriptions	Syntax	Parameters
allowSubscribe	Invoked before subscribing to a channel/sub-channel. Used to check if the requested client can be allowed to subscribe to the given channel. If returns `true`, allows the requested client to subscribe.Properties defined in the object `subscriberInfo` can be used decide on authorization.	`allowSubscribe(subscriberInfo)`	• `subscriberInfo` : Subscriber information struct. Contains information that is passed by the function `subscribe` in the `custom_header` struct. It has a `connectionInfo` object which can be accessed through `ConnectionInfo` key.

© 2014 Adobe Systems Incorporated. All rights reserved.

`allowPublish`	Called before publishing to a channel/sub-channel. Used to check whether the requested client can be allowed to publish to the given channel. Properties defined in the publisherInfo struct can be used to take the authorization decision. If returns `true`, allows the requested client to publish.	`allowPublish(publisherInfo)`	• `publisherInfo` : Publisher information struct. Contains the information that is passed by the function `publish` in the custom_headerstruct. It has a `connectionInfo` object which can be accessed through `ConnectionInfo` key.
`beforePublish`	Invoked before publishing the message on requested channel/sub-channel. Used to execute a business logic if required and to format messages.	`beforePublish(message, publisherInfo)`	• `message` : The message that has to be published to the client. This can be of type `Any` . • `publisherInfo` : Publisher information struct.
`canSendMessage`	Invoked before sending the message to a subscribed client.Used to decide whether the message should be sent to a specific client. Called for all clients subscribed to a channel separately. Properties defined in the object `subscriberInfo` and `publisherInfo` help you find client's interest in the message.	`canSendMessage(message, subscriberInfo, publisherInfo)`	• `message` : The message that has to be published to the client. This can be of type `Any` . • `subscriberInfo` : Subscriber information struct. • `publisherInfo` : Publisher information struct.
`beforeSendMessage`	Invoked before sending the message to a subscribed client. Can be used to format the message as per client requirement before sending. This function is executed for each client.	`beforeSendMessage(message, subscriberInfo)`	• `message`: The message that has to be published to the client. This can be of type `Any` . • `subscriberInfo` : Subscriber information struct.

© 2014 Adobe Systems Incorporated. All rights reserved.

`afterUnsubscribe`	Called after you unsubscribe from a channel/sub-channel. Used to clear the settings if necessary..	`afterUnsubscribe (subscriberInfo)`	• `subscriberInfo` : Subscriber information struct.

Using channel-specific message handler

When you create a WebSocket object, you can define a message handler to manage responses from the server. The handler can manage messages from all the channels. Additionally, while subscribing to a channel, you can define channel-specific message handler.

Example

Application.cfccomponent

```
{
 this.name = "channelspecifichandlerexample";
 this.wschannels = [{name="stocks"}, {name="news"}, {name="products"}];
}
```

index.cfm

```
<script type="text/javascript">
 function stockhandler(stocksmessageobj){
     //write appropriate logic here to handle data coming on stock channel
     if (stocksmessageobj.data != null) {
         var message = stocksmessageobj.data;
         var datadiv = document.getElementById("myDiv");
         datadiv.innerHTML += "Stock Handler:  " + message + "<br />";
     }
 }
 function newshandler(newsmessageobj){
     //write appropriate logic here to handle data coming on news channel

     if (newsmessageobj.data != null) {
         var message = newsmessageobj.data;
         var datadiv = document.getElementById("myDiv");
         datadiv.innerHTML += "News Handler:  " + message + "<br />";
     }
 }
 function productshandler(productsmessageobj){
     //write appropriate logic here to handle data coming on products channel

     if (productsmessageobj.data != null) {
         var message = productsmessageobj.data;
         var datadiv = document.getElementById("myDiv");
         datadiv.innerHTML += "Product Handler:  " + message + "<br />";
     }
 }
 function subscribeMe(){
     var channel = document.getElementById("channeloption").value;
     switch (channel) {
```

© 2014 Adobe Systems Incorporated. All rights reserved.

```
            case "stocks":
                mysock.subscribe("stocks", {}, stockhandler);
                break;
            case "news":
                mysock.subscribe("news", {}, newshandler);
                break;
            case "products":
                mysock.subscribe("products", {}, productshandler);
                break;
        }
    }
    function mycbHandler(messageobj){
        var message = ColdFusion.JSON.encode(messageobj);
        var datadiv = document.getElementById("myDiv");
        datadiv.innerHTML += "Message Handler : " + message + "<br />";
    }
</script>
<cfwebsocket name="mysock" onmessage="mycbHandler"/>
<form>
 <select id="channeloption">
  <option>
   stocks
  </option>
  <option>
   news
  </option>
  <option>
   products
  </option>
 </select>
 <input id="subscribe" name="subscribe" value="Subscribe" type="button"
        onclick="subscribeMe();">
</form>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<div id="myDiv">
</div>
```

PublishMessage.cfm

```
<cfscript>
 if(isdefined("form.publish"))
  WsPublish(#form.channel#, #form.message#);
</cfscript>
<cfform method="post">
 <cfselect name="channel">
  <option>
   stocks
  </option>
  <option>
   news
  </option>
  <option>
   products
  </option>
 </cfselect>
 Message:
 <input id="message" name="message" type="text">
 <cfinput id="publish" name="publish" value="publish" type="submit">
</cfform>
```

Specifying custom information

The following JavaScript functions `subscribe`, `publish`, `invokeandpublish` and the serverside function `WSPublish` let you pass custom information (for example age or status) in the form of key-value pairs.

You can use these custom data in the `subscriberInfo` in the channel listener.

Example 1: Custom information using subscribe

In the following example,

1. In the `subscribe` function, specify the age of the client.
2. In the channel listener, specify the age restriction in the `allowSubscribe` method.

Application.cfc

```
component
{
this.name = "customoptionexample";
this.wschannels = [{name="Testchannel", cfclistener="TestListener"}];
}
```

Index.cfm

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<script type = "text/javascript">
       function  msgHandler(messageobj)
       {
              var ouputdiv=document.getElementById("myDiv");
              var message = ColdFusion.JSON.encode(messageobj);
              ouputdiv.innerHTML+=message +"<br >" +"<br>";
       }
       function subscribeMe()
       {
              var clientAge = document.getElementById("age").value;
    TestSocket.subscribe("Testchannel", {age: clientAge});       }</script>
<cfwebsocket name="TestSocket" onMessage="msgHandler"/>
<br>
Age <input id="age" name="age" type="text">
<br>
<input id="stocksSubscribe" type="button" value="Subscribe"
onclick="subscribeMe();">
<div id="myDiv"></div>
```

TestListener.cfc

```
component extends="CFIDE.websocket.ChannelListener"
{

       public boolean function allowSubscribe(Struct subscriberInfo)
       {
              if(structKeyExists(subscriberInfo, "age"))
                     if((subscriberInfo.age gt 18))                    {
                            return true;
                     }
                     else
                     {
                            return false;
                     }
              else
                     return false;
       }

}
```

Example 2: Custom information using publish

The following example illustrates how you communicate bid value data.

1. In the function `publish`, specify the bid value as custom information.
2. In the channel listener, specify a condition to validate the bid by comparing it with the current bid value. The message is broadcasted only if the bid is valid.

Application.cfc

© 2014 Adobe Systems Incorporated. All rights reserved.

```
component
{
 this.name = "customoptionexample1";
 this.wschannels = [{name="bidchannel", cfclistener="bidListener"}];

 boolean function onApplicationStart()
 {
  application.bidvalue = 1;
 }

}
```

Index.cfm

```
<script type="text/javascript">
 function msgHandler(messageobj){
     if (messageobj.data != null) {
         var ouputdiv = document.getElementById("myDiv");
         var message = messageobj.data;
         ouputdiv.innerHTML += message + "<br >" + "<br>";
     }
     if (messageobj.code == -1 && messageobj.reqType == "publish") {
         var ouputdiv = document.getElementById("myDiv");
         var message = "Bid amount is less than current bid value";
         ouputdiv.innerHTML += message + "<br >" + "<br>";
     }
 }
 function publishme(){
     var bidvalue = document.getElementById("amount").value;
     var clname = document.getElementById("name").value;
     var message = "Bid placed by " + clname + " Amount " + bidvalue;
     TestSocket.publish("bidchannel", message, {
         value: bidvalue
     });
 }
</script>
<cfwebsocket name="TestSocket" onmessage="msgHandler" subscribeto="bidchannel"/><br>

Bid Amount:
<input id="amount" name="amount" type="text">
Name:
<input id="name" type="text">
<input id="publishmessage" type="button" value="Publish" onclick="publishme();">
<div id="myDiv">
</div>
```

bidListener.cfc

© 2014 Adobe Systems Incorporated. All rights reserved.

```
component extends="CFIDE.websocket.ChannelListener"{ public boolean function
 allowPublish(Struct publisherInfo)
 {
 if(structKeyExists(publisherInfo, "value"))
 if((publisherInfo.value gt application.bidvalue))
 {
 application.bidvalue = publisherInfo.value;
 return true;
 }
 else
 { return false;
 }
 else
 return false;
 }
 }
```

Using selectors

Selectors provide filtering logic when you subscribe to a channel or publish a message.

While subscribing to a channel, a selector is provided as part of the subscriber information struct. For example, a client is subscribed to a channel with a selector `product eq abc`. Only the messages that contain the publisher information `product:abc` are published to the subscriber.

Similarly, you can include a selector as part of the publisher information struct while publishing a message. The messages are published to only the subscribers who satisfy the criteria specified in the selector.

> **Note**
>
> For a custom channel listener CFC, you cannot have the `canSendMessage` method (for the selector to work).

The following example shows how to subscribe to a channel with a selector:

1. Create a channel `selectorchannel` in the `Application.cfc`.

```
component
{
 this.name = "websocketApp1";
 this.wschannels = [{name="selectorchannel"}];
}
```

2. Create a CFM that allows the user to subscribe to the channel, with the selector condition value greater than the specified value, for example you want to see only the stock values above a particular threshold value.

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<script type="text/javascript">
 function  msgHandler(messageobj)
       {
              if (messageobj.data != null) {
                     var ouputdiv = document.getElementById("myDiv");
                     var message = messageobj.data;
                     ouputdiv.innerHTML += message + "<br >" + "<br>";
                     }
       }
       function subscribeMe()
       {
              var amt = document.getElementById("amount").value;
              var selectstring="value gt "+ amt;
    TestSocket.subscribe("selectorchannel", {selector : selectstring});
              document.getElementById("stocksSubscribe").style.display =
'none';
              document.getElementById("l1").style.display = 'none';
              document.getElementById("l2").style.display = 'block';
              document.getElementById("publisme").style.display = 'block';
       }
       function publishme()
       {
              var amt = document.getElementById("amount").value;
              TestSocket.publish("selectorchannel","Value is " +amt,{value:
amt})
       }
</script>
<cfwebsocket name="TestSocket" onMessage="msgHandler"/>
<br />
<lablel  id="l1">Stock value above which you want to recieve message</lablel>
<label id="l2" style="display:none;">Value</label>
<input id="amount" name="amount" type="text">
<br>
<input id="stocksSubscribe" type="button" value="Subscribe"
onclick="subscribeMe();">
<input id="publisme" type="button" value="Publish" onclick="publishme();"
style="display:none;">
<div id="myDiv"></div>
```

Specifying WebSocket authentication details

You can specify the authentication details for WebSocket at the application level. An application method `OnWSAuthenticate` has been added to support application level authentication.

1. In the Application.cfc, define the function `onWSAuthenticate`.
2. Call the JavaScript function `authenticate`. This function calls the onWSAuthenticate function.

> **Note**
>
> You cannot restrict authentication to a specific channel. If you do not want to authorize the client to any channels, it can be done using the allowSubscribe function of channel listener.

© 2014 Adobe Systems Incorporated. All rights reserved.

onWSAuthenticate

Description

Authenticates the user

Syntax

```
OnWSAuthenticate(username, password, connectionInfo)
```

Parameters

Parameter	Description
`username`	Name of the user that has to be authenticated.
`password`	Password for the user.
`connectionInfo`	A struct that contains the following keys: • `Authenticated`: `YES|NO` • `ConnectionTime`: Connection time stamp. • `clientID`: Unique ID of the client.

Custom keys are also supported. For example, you can specify the user's role, status, or age.The `connectionInfo` is shared across all the channels for a given WebSocket client. Also, modifications are persisted across all the subscriptions for that client.|

Example

The following example uses the function `onWSAuthenticate`, validates the user, and associates a user role.

> **Note**
>
> For this example to work, ensure that you implement the user-defined functions.

© 2014 Adobe Systems Incorporated. All rights reserved.

```
component
{
 this.name="websocketsampleapp23";
 this.wschannels=[{name="stocks",cfclistener="stocksListener"}];

 function  onWSAuthenticate(username, password, connectionInfo)
       {
           //write appropriate logic to fetch user password in funtion checkPassword


           If(checkPassword(username) eq password)
           {
              connectionInfo.authenticated="YES";
         //Role is the custom information that you provide
  connectionInfo.role= "admin";

   return true;
           }
           else{
              connectionInfo.authenticated="NO";
             return false;
           }
           writedump("#connectionInfo#","console");

         }

}
```

Enabling Single sign-on in WebSocket

If already authenticated to the ColdFusion server, you need not enter the credentials while subscribing to a WebSocket channel. You need only specify `useCFAuth = "true"` in the tag `cfwebsocket`.

In the following example, a login page is created using the tag `cflogin`. After authentication, the client is subscribed to a channel. That is, the client does not pass the login credentials while subscribing. Here, the function `onWSAuthenticate` need not be set up or called using the JavaScript function `authenticate`.

Example

1. Create a login page using the tag `cflogin`.

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<form name="form1" method="post" >
        User: <input name="username" type="text" id="username"
value="admin"><br>
        Pass: <input name="password" type="text" id="password"><br>
        <input type="submit" name="Submit" value="Submit">
 </form>
<cfif structKeyExists(form,"username")>
    <cflogout>
    <cfset r = "user">
    <cfif FORM.username is "admin">
     <cfset r = "admin">
    </cfif>
 <cflogin idletimeout="1800">
         <cfloginuser
              name = "#FORM.username#"
              password ="#FORM.password#"
              roles = #r#>
  </cflogin>
<cfoutput>Authorized user: #getAuthUser()#</cfoutput><br>
<cfoutput>Authorized role: #GetUserRoles()#</cfoutput><br>
<cflocation url="index.cfm">
</cfif>
```

2. Create the `Application.cfc` with a channel `shares`, and listener `channelListener`.

```
component
{
   this.name = "myssoexample";
   this.wschannels = [ {name = "sso", cfclistener="ChannelListener"}];
 }
```

3. Create a CFM that allows the user to subscribe to a channel and publish the message.

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<script>
var mycbHandler = function(msg)
 {
 if(msg.data)
  {
   var messageObject = msg.data;
   var txt=document.getElementById("myDiv");
         if((messageObject.indexOf("{")!=-1)){
     var jsonValue = (new Function( "return( " + messageObject + " );" ))();
    if(jsonValue){
           txt.innerHTML+="<br>Authenticated : " + jsonValue.authenticated;
           txt.innerHTML+="<br>UserName : " + jsonValue.username;
           txt.innerHTML+="<br>Role : " + jsonValue.roles;
           }
     }
  }
 }
var openHandler = function()
 {
  mysock.publish("sso","hii");
 }
</script>
<form name="f">
<cfwebsocket name="mysock" onMessage="mycbHandler" subscribeto="sso"
onOpen="openHandler" useCFAuth="yes"/>
<div id="myDiv"></div>
</form>
```

4. Create a ColdFusion listener file that returns the subscriber information while publishing.

```
component extends="CFIDE.websocket.ChannelListener"
{
public any function beforeSendMessage(any message, Struct subscriberInfo)
  {
   writedump(var=subscriberInfo.connectionInfo, output="console");
      return subscriberInfo.connectionInfo;
  }

}
```

Server- side APIs

WebSocket server-side APIs allow a CFC to communicate with a channel or a specific client. For example, when a message is received from the client, the server sends a specific message.

The following table details the server-side functions:

Function Name	Syntax	Description

© 2014 Adobe Systems Incorporated. All rights reserved.

`WSgetSubscribers`	`WSgetSubscribers (channel)`	Returns an array of struct with clientID and subcriberInfo as the keys.
`WSPublish`	`WSPublish(channel, message [, filterCriteria])`	Sends messages to a specific channel. Optionally, you can specify a filter criteria struct.
`WSGetAllChannels`	`WSGetAllChannels ()`	Provides all the channels defined in the Application.cfc as an array.

Interpreting the Server response

The following is a sample response that you receive from the server:

```
{"clientid":2077108630,"ns":"coldfusion.websocket.channels","reqType":"welcome","cod
e":0,"type":"response","msg":"ok"}
{"clientid":2077108630,"ns":"coldfusion.websocket.channels","reqType":"subscribe","c
ode":0,"type":"response","msg":"ok"}
```

The table explains the response keys:

Key	Description
`clientid`	Unique ID assigned to a client.
`ns`	ColdFusion WebSocket namespace
`reqType`	The type of request represented by the JavaScript functions (for example, `publish`, or `invokeandpublish`).
`code`	See <code section>
type	Either response or data. • Response: Conveys if the request is successful. • Data: The data that the channel recieves.
`msg`	Applies to responses; `ok` if successful. On failure, returns the cause of failure.
`subscriberCount`	An integer that stands for the subscriber count.
`channels`	Applies to `getSubscriptions`, list of all subscribed channels.

© 2014 Adobe Systems Incorporated. All rights reserved.

`data`	Message is conveyed through data.
`publisherID`	Client ID of the publisher. If the message is from `WSPublish`, the ID is `0`.
`channelname`	Name of the channel.

The following is a sample `subscribeto` response sent to the message handler:

```
{"clientid":2077108664,"ns":"coldfusion.websocket.channels","reqType":"welcome","cod
e":0,"type":"response","msg":"ok"}
{"clientid":2077108664,"ns":"coldfusion.websocket.channels","channelsnotsubscribedto
":"stocks.a","reqType":"subscribeTo","code":0,"type":"response","msg":"Subscription
failed for channel(s) 'stocks.a'.","channelssubscribedto":"stocks"}
```

In this case, you have the key `channelssubscribedto` and `channelsnotsubscribedto` which applies to scenarios related to the attribute `subsscribeTo` in the tag `cfwebsocket`.

© 2014 Adobe Systems Incorporated. All rights reserved.

Using WebSocket in point-to-point communication

Assume that the client need not send message to multiple clients. That is, the bi-directional communication is only between one client and server.

So unlike in the broadcast model, you do not need a channel for communication and therefore need not define it in the Application.cfc.

Do the following to set up a point-to-point communication:

1. Create the WebSocket object using the `cfwebsocket` tag. For details, see *Creating WebSocket object using cfwebsocket* in Using WebSocket to broadcast messages.
2. To send a message to the server, use the JavaScript method `Invoke`.
3. To send additional messages, use the function `WSSendMessage`.

invoke

Description

Calls a particular function in the CFC. The value returned by the function in CFC is sent back to the client that invokes the method.

Syntax

```
invoke(CFCName, functionName [, argumentsArray])
```

Parameters

Parameter	Description
`cfcName`	The CFC filename (from where a specific function is called).
`functionName`	The function name in the CFC file.
`argumentsArray`	The function arguments as an array.

Usage

Use the function `WSSendMessage` to send additional message back to the client inside the function. To continuously send messages to a client you have to create a thread in the method that you invoke using `invoke`. Further, you can keep sending messages inside a thread.

Example

```
mycfwebsocketobject.invoke("employee","getdept",["eid_2"]);
```

WSSendMessage

Description

Sends messages to a specific client that invokes the method. This can be included as a part of the function that is

© 2014 Adobe Systems Incorporated. All rights reserved.

called by the `invoke` WebSocket JavaScript method.

Returns

Nothing

Syntax

```
WSSendMessage(message)
```

Parameters

Parameters	Description
`message`	Required. The message object. This can be of type `Any`.

Example: Point-to-Point communication

The following example shows how to implement a point-to-point communication. In this example, you invoke three functions defined in mycfc.cfc.

- Method `f1` returns a value.
- Method `f2` expects an argument and returns a string. Also, client receives an additional message from the method `WSSendMessage` used in `f2`.
- In the method `f3`, you create a thread that sends messages to the client at particular intervals.

1. Create a CFM page index.cfm.

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<script type="text/javascript">
       function msgHandler(msgobj){
           var txt = document.getElementById("myDiv");
           var message = ColdFusion.JSON.encode(msgobj);
           txt.innerHTML += message + "<br >" + "<br>";
       }
       function invokecfcfn(){
              var fname= document.getElementById("fnname").value;
              if (fname == "f2") {
                     alert("f2 selected");
                     mysocket.invoke("mycfc", "f2", ["echo"]);
              }
              else
                     mysocket.invoke("mycfc", fname);

       }
</script>
<cfwebsocket name="mysocket" onmessage="msgHandler"/>
<form>
       <select id="fnname">
              <option>f1</option>
              <option >f2</option>
              <option>f3</option>
       </select>

       <input id="invokefn" name="invokefn" value="Invoke CFC function "
type="button"  onclick="invokecfcfn();">
       <div id="myDiv">
       </div>
</form>
```

2. Create a CFC mycfc.cfc that contains the function called from the client page.

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cfcomponent>

       <cffunction name="f1" >
              <cfreturn "Message returned from f1">
       </cffunction>

       <cffunction name="f2" returntype="string" >
              <cfargument name="arg1" type="string" required="true"  >
              <cfset msg= "Message from wsssendmessage of f2 which you called
with arg " & arg1>
              <cfset wssendMessage(msg)>
              <cfreturn "Message returned from f2">
       </cffunction>

       <cffunction name="f3" >
              <cfthread action="run" name="t1" >
                     <cfloop index="i" from="1" to="10">
                           <cfset sleep(20000)>
                           <cfset wssendMessage("Message #i# from
wsssendmessage of f3 #now()#")>
                     </cfloop>
              </cfthread>
              <cfreturn "Thread initiated in f3">
       </cffunction>
</cfcomponent>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

Error handling in ColdFusion WebSocket

The WebSocket errors are categorized as follows:

Category	Description	Response code
Channel Error	For example, channel not present or client has already subscribed to the channel	-1
Application Error	Runtime errors while invoking CFC	4001
Success	On successful completion of request	0

Note

If you have defined an Error Handler, it receives the responses -1 and 4001. If not defined, the response goes to Message Handler.

© 2014 Adobe Systems Incorporated. All rights reserved.

Using ColdFusion Administrator for WebSocket Configurations

Use the ColdFusion Administrator (Server Settings > WebSocket) to specify the following WebSocket-related details:

- Port that the WebSocket server listens to
- Socket timeout
- Data size of packets sent/received
- If to enable Flash fallback

© 2014 Adobe Systems Incorporated. All rights reserved.

Browser support for WebSocket

The following table lists the browser support for WebSocket.

Browser	WebSocket Support
Internet Explorer 6, 7, 8, and 9	Supported if Flash Player is installed.
Mozilla Firefox 3.x	Supported if Flash Player is installed
Mozilla Firefox 6, 7, and 8	Supported
Google Chrome 15.x and above	Supported
Safari 5.x	Supported
(Android) Default browser	Supported
(iPad) Safari	Supported

#back to top

© 2014 Adobe Systems Incorporated. All rights reserved.

Media Player enhancements-Developing guide

The enhancements in this release support:

- Play back capability for HTML 5 videos
- Fallback to HTML 5 video playback if Flash player is not installed
- Browser independent video controls
- Dynamic streaming of Flash videos
- Advanced skinning for media player
- Play list for Flash videos
- Extending media player using plug-ins built using Open Source Media Framework (OSMF), for example to:
- Play videos in the YouTube server
- Use stage video support by showing advertisements within the videos in linear and non-linear mode
- Adding title to the video
  For details, see Media Player enhancements.

#back to top

© 2014 Adobe Systems Incorporated. All rights reserved.

HTML 5 video playback capability

Apart from playing Flash videos, you can play videos in any format supported by HTML 5 compliant browsers.

Specify the source for HTML 5 videos in one of the following ways:

- If you feed only one video source, specify it as shown in the following example:

```
<cfmediaplayer name="Player" source="myvideo.mp4" width="200" height="200" />
```

- To feed multiple videos, specify the source as shown in the following example:

```
<cfmediaplayer name="Player" width=200 height=200>
<source src="movie.ogg" type="video/ogg" />
<source src="movie.mp4" type="video/mp4" />
<source src="movie.webm" type="video/webm" />
</cfmediaplayer>
```

Note

The `source` tag represents the HTML 5 video source tag. When source tag is specified, you are recommended to use the attribute type.

- Dynamically set the media player source using the JavaScript function `ColdFusion.MediaPlayer.setSource.`

For best results with the player, ensure that you specify doctype in your code as follows: `<!DOCTYPE html>`

© 2014 Adobe Systems Incorporated. All rights reserved.

Fallback plan for cfmediaplayer

Depending on the playback capability on the browser, `cfmediaplayer` offers alternate playback plan. `cfmediaplayer` supports both Flash to HTML and HTML to Flash fall back.

The fall back feature applies in the following scenarios:

Scenario	Fallback
Flash is not installed	Falls back to HTML 5 video playback, provided browser is HTML 5 compliant and video source is supported by the browser.If not, results in an error and displays the player container.
Browser does not have HTML 5 video playback capability	Checks for Flash player.If Flash player is available, plays the content in Flash player (for supported formats). If not, results in an error and displays the player container.
Flash is not installed and the browser does not support HTML 5 videos	Results in an error and displays the player container.
Both Flash and HTML playback are supported and Flash is the default.	From the given video source, media player tries to run the Flash video, if available. Else, tries to load the HTML 5 video unless you specify `html` as the value for `type` attribute in `cfmediaplayer`.

© 2014 Adobe Systems Incorporated. All rights reserved.

Specifying the playback type

You can set the preference of playback using the attribute `type` either as `Flash` or `HTML` as shown in the following examples:

- `<cfmediaplayer source="myvideo.mp4" type="html"/>`
- `<cfmediaplayer source="myvideo.mp4" type="flash"/>`

Assume that the playback is on Internet Explorer 9 or Google Chrome. By default, the value is set to `flash.**` `cfmediaplayer` honors the value you specify only if the browser supports the playback preference. If for example, you specify `type` as `flash` and the browser does not have Flash player installed, it falls back to HTML 5.

© 2014 Adobe Systems Incorporated. All rights reserved.

Specifying the video poster image, loop playback, and title using the tag cfmediaplayer

- `posterImage`: Specify this attribute to set a poster image for the video playback. Takes URL or relative address as value.
- `repeat`: Set this attribute to `true` to continue playback from first to last frame after the media player reaches the end of the video.The default value is `false`.
- `title`: Lets you set title on the media player. The title appears over the media player on upper-left corner. If `title` is specified and `hideTitle` is not specified, then `hideTitle` is set to `false`. If `title` is not specified and `hideTitle` is set to `false`, then the video filename is displayed as title.

Example

```
<!DOCTYPE html>
<html>
<head>
</head>
<body>
<cfmediaplayer
name="player_html"
title="End Game"
source="/videos/gizmo.ogv"
posterImage="/images/music.jpg"
repeat="true">
</cfmediaplayer>
</body>
</html>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

Extending the media player capabilities

> **Note**
>
> This feature applies only to Flash videos.

Using FlashVars, you can extend the media player to use capabilities of both OSMF plug-ins and Strobe media player.

As this section describes, you can use plug-ins in the OSMF market. Two popular plug-ins are demonstrated.

The general procedures to extend the media player capability are as follows:

1. Get the FlashVars and SWFs (from the plug-in vendor)
2. In the `cfmediaplayer`, insert a param tag with the FlashVars.

Playing videos on YouTube server

The following example shows how to play a video hosted on YouTube server using `cfmediaplayer`:

```
<!DOCTYPE html >
<html>
<head>
</head>
<body>
<cfmediaplayer
    name="player_youtube"
    source="http://www.youtube.com/watch?v=gv68hDObACk&feature=feedrec_grec_index"
type="flash"
    >
    <param name="flashvars" value="plugin_YoutubePlugin=YouTubePlugin.swf" />
</cfmediaplayer>
</body>
</html>
```

Here, source is the actual YouTube video link. You point to the YouTube plug-in SWF using FlashVars. You can get the YouTubePlugin.swf file here.

Playing linear and non-linear advertisements using stage video

You can extend the mediaplayer to play advertisements using a staged video. The advertisements play within the video in linear and non-linear mode.

Example

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<!DOCTYPE html>
<html>
<head>
<script type="text/javascript">
function nonLinear()
{
 var player = ColdFusion.MediaPlayer.getPlayer('player');
 player.displayNonLiniarAd (

"http://gcdn.2mdn.net/MotifFiles/html/1379578/PID_938961_1237818260000_women.flv", {

    right: 10 * Math.random(),
    bottom: 10 * Math.random(),
    scaleMode: "none"});
}
</script>
</head>
<body style="background:#FFFFFF">
<input name = "AdvertisementPlugin" value="Advertisement" type="button"
onClick="nonLinear('player')">
<cfmediaplayer
    name="player"
    source="/videos/cathy2_HD.mp4"
    type="flash"
    >
    <param name="flashvars"
value="plugin_ads=/CFIDE/scripts/ajax/resources/cf/assets/AdvertisementPlugin.swf"
/>
</cfmediaplayer>
</body>
</html>
```

In this example, using FlashVars, you point to the advertisement plug-in. Custom JavaScript function from the plug-in is used to play video in linear and non-linear mode.

References

- For details on the supported FlashVars, see *Chapter 2: Configuring the Player* in the documentation available at the following URL: http://www.osmf.org/downloads/pdf/using_fmp_smp.pdf
- To find more plug-ins that you can use with the mediaplayer, go to the following URL:http://www.osmf.com/partner.php
- See the section Using Plug-ins in *Chapter 4: Advanced Topics* for details of how to use the plug-ins inside `cfmediaplayer` in the document available at the following URL: http://www.osmf.org/downloads/pdf/using_fmp_smp.pdf
- For more information on Strobe media playback, see the presentation available at the following URL: http://www.osmf.com/downloads/pdf/Strobe_Media_Playback_Presentation.pdf

© 2014 Adobe Systems Incorporated. All rights reserved.

Embedding subtitle files to an HTML 5 video

Note

This option is not currently available for Flash videos.

You can embed subtitles using the HTML 5 `track` element. `track` element takes `src` as subtitle (SRT) file, with language type and label, and displays it in subtitle options.

This feature works only if the browser supports HTML 5 `track` element.

Example

```
<cfmediaplyer source="myvideo.ogv">
<track kind="subtitle" src="myvideo_en.srt" label="English" srclang="en" />
<track kind="subtitle" src="myvideo_fr.srt" label="French" srclang="fr" />
</cfmedialayer>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

Play list support for Flash videos

> **Note**
>
> This option is not available for HTML 5 videos.

The list of media that you have to play are embedded in play list file M3U format.

You can specify the play list as follows:

```
<cfmediaplayer source="playlist.m3u" />
```

© 2014 Adobe Systems Incorporated. All rights reserved.

Skinning

In ColdFusion 9, you have the option to skin the Flash player using the following styles (using the attribute `style`): `bgcolor/bgColorTheme`, `hideborder`, `titleColorTheme` , `controlsColorTheme`, `progressColorTheme`, and `progressbgColor`.

For HTML 5 playback, you can use these attributes.

> **Note**
>
> In ColdFusion 10, you might not be able to skin the progress bar and control bar using the `style` attributes. This is because, the new mediaplayer used in this release currently does not support them.

For Flash player, skin can be controlled using an XML by specifying the path for the `skin` attribute as follows:

```
<cfmediaplayer source="myvideo.mp4" skin="./skin/myskin.xml">
```

> **Note**
>
> Specifying styles using XML applies only to Flash videos.

Reference

Go to the following URL to see a sample skin for mediaplayer: http://www.rblank.com/2010/10/06/sample-skin-for-strobeflash-media-playback/

© 2014 Adobe Systems Incorporated. All rights reserved.

Player controls

ColdFusion 9 supports the following mediaplayer controls for Flash videos: Play/Pause, Stop, progress bar, Current and total playback time, volume, and full screen.

While the Stop control is not supported hereafter, enhancements in ColdFusion 10 let you use these controls for HTML 5 videos also.

In addition, when you use Flash Player, Next and Previous controls are supported. They appear when playing the play list. Currently, this control is available only for Flash player and appears when source specified is a play list file (M3U).

© 2014 Adobe Systems Incorporated. All rights reserved.

Dynamic streaming

> **Note**
>
> This feature applies only to Flash player.

At server side

The feature lets dynamic delivery of streaming videos by alternating among various playback streams. To set up dynamic streaming, point to the dynamic streaming video using the `source` attribute in `cfmediaplayer` as follows:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<script type="application/javascript">
functi     on getStreamData()
{
 var player = ColdFusion.MediaPlayer.getPlayer("player_ds");
 var dynamicStreams=player.getStreamItems();
 for (var idx = 0; idx < dynamicStreams.length; idx ++)
 {
  var streamLabel = dynamicStreams[idx]['width'] + "x" +
dynamicStreams[idx]['height'] + " @ " + dynamicStreams[idx]['bitrate'] + "kbps";
  var element = document.createElement("option");
  element.innerHTML = streamLabel;
  document.getElementById("streams").appendChild(element);
  if (idx==player.getCurrentDynamicStreamIndex())
  {
   document.getElementById("streams").selectedIndex = idx;
  }
 }
}
function selectionChange()
{
 var index = document.getElementById("streams").selectedIndex;
 var player = ColdFusion.MediaPlayer.getPlayer("player_ds");
 player.setAutoDynamicStreamSwitch(false);
 player.switchDynamicStreamIndex(index);
}
</script>
</head>
<body>
<div  align="center">
<br />
<br />
<br />
<button type="button" onclick="getStreamData()"  value="Stream Details" name=
"Stream Details"> Stream Details</button>
<br />
<br />
<select name="streams" id="streams" onchange="selectionChange()">
</select>
<br />
</div>
<cfmediaplayer
    name="player_ds"
    source="URL to the F4M file"  width=800 height=500 type="flash"
    autoplay="true"
 align="center"
    >
</cfmediaplayer>
</body>
</html>
```

At client side

© 2014 Adobe Systems Incorporated. All rights reserved.

At the client-side, you have the HD controls which can be used to toggle between different streaming videos available.

#back to top

© 2014 Adobe Systems Incorporated. All rights reserved.

Displaying geolocation - Developing guide

Displays user location on the map if the attribute `showUser` is specified in `cfmap`. This feature works only on HTML 5 compliant browsers.
For details, see Displaying geolocation.

#back to top

© 2014 Adobe Systems Incorporated. All rights reserved.

cfinput-Developing guide

- The attribute `type` now supports all HTML 5 input types, for example, email, range, or date. For details, see http://dev.w3.org/html5/spec/Overview.html#the-input-element
- The tag supports new attributes such as `max` and `min`.

#back to top

© 2014 Adobe Systems Incorporated. All rights reserved.

cfmap-Developing guide

- A new attribute `showUser` has been added. The default value is `false`. If `true`, on HTML-compliant browsers, user location is shown on the map.For browsers that are not HTML 5 compliant, the address falls back to the value you specify for `centerAddress`. If no value is specified, it falls back to the value specified for `centerLatitude` and `centerLongitude`.

> **Note**
>
> User has to authenticate the site so that it tracks user location. For example, in Google Chrome, you are prompted to Allow to track your Physical location.

© 2014 Adobe Systems Incorporated. All rights reserved.

cfmapitem-Developing guide

- A new attribute `showUser` has been added. The default value is `false`. If `true`, on HTML-compliant browsers, user location is shown on the map.

© 2014 Adobe Systems Incorporated. All rights reserved.

Client-side charting-Developing guide

ColdFusion 10 supports client-side charting. This is in addition to the existing server-side charting feature (which continues to serve the way it used to). Client-side charting supports the following:

- Dynamic and interactive charting: Modify the charts, add styles, and add new series or plots
- Popular chart formats with appropriate fallback functionality: Use HTML 5, Flash, SVG, or VML charts.
- If your browser does not support HTML 5 features relevant to charting, charts are rendered in Flash. Similarly, if
- Flash is not supported, charts are rendered in HTML.
- Features identical to server-side charting: Most of the server-side charting features are available with client-side charting.
- Old and new charts: In addition to the contemporary chart types, offers a new set of charts.
- Needs minimal trips to server: As compared to generating charts at server-level, for every user interaction. For details, see Client-side charting.

#back to top

© 2014 Adobe Systems Incorporated. All rights reserved.

Supported charts

Line Charts	Area Charts	Bar Charts	Scatter Charts
Bubble Charts	Horizontal Bar Charts	Pie Charts	Radar Charts
Bullet Charts	Nested Pie Charts	Piano Charts	Funnel Charts
Gauges	Horizontal Bullet Charts	Cone	3D Line Charts
3D Area Charts	3D Pie Charts	3D Horizontal Bar Charts	Pyramid
Cylinder			

© 2014 Adobe Systems Incorporated. All rights reserved.

How client side charting works

1. Use the `cfchart` tag.
2. Specify the chart details as you specify them for server-side charting in the previous releases.

© 2014 Adobe Systems Incorporated. All rights reserved.

Limitations

The following server-side charting features are not available with client-side charting:

- Linking charts to URL
- Writing charts to a variable

© 2014 Adobe Systems Incorporated. All rights reserved.

Charting examples

Example 1

The following is a basic example of using client side charting.

```
<cfchart format ="html" type="pie">
    <cfchartseries>
        <cfchartdata item="New car sales" value="50000">
        <cfchartdata item="Used car sales" value="25000">
        <cfchartdata item="Leasing" value="30000">
        <cfchartdata item="Service" value="40000">
    </cfchartseries>
</cfchart>
```

Example 2

This example showcases how you can create a simple bubble chart by specifying zcolumn.

```
<cfchart format ="html" type="bubble" query="ChartQuery" showlegend="false">
<cfchartseries query="ProdQuery" type="bubble" itemcolumn="title"
valuecolumn="total_days" zcolumn="rem_days" label="Total_Days">
</cfchart>
```

Example 3

This is an example that specifies labels as a struct.

```
<cfchart format ="html" type="bubble" query = " ChartQuery " showlegend="false"
labels=#[{"text"="Hello
Label","hook":"node:plot=0,index=2,offset-x=-10,offset-y=-90"}]#>
<cfchartseries type="bubble" label="Total_Days">
<cfchartdata item=1 value=10 zvalue=40>
<cfchartdata item=2 value=20 zvalue=30>
<cfchartdata item=3 value=30 zvalue=20>
<cfchartdata item=4 value=20 zvalue=35>
<cfchartdata item=5 value=40 zvalue=10>
</cfchartseries>
</cfchart>
```


- Example 1
- Example 2
- Example 3

© 2014 Adobe Systems Incorporated. All rights reserved.

Flex and AIR Integration in ColdFusion

This section explains about the effective Adobe Flex and Adobe AIR integration techniques with ColdFusion.

- *Using the Flash Remoting Service*
- *Using Flash Remoting Update*
- *Offline AIR Application Support*
- *Proxy ActionScript Classes for ColdFusion Services*
- *Using the LiveCycle Data Services ES Assembler*
- *Using Server-Side ActionScript*

#back to top

© 2014 Adobe Systems Incorporated. All rights reserved.

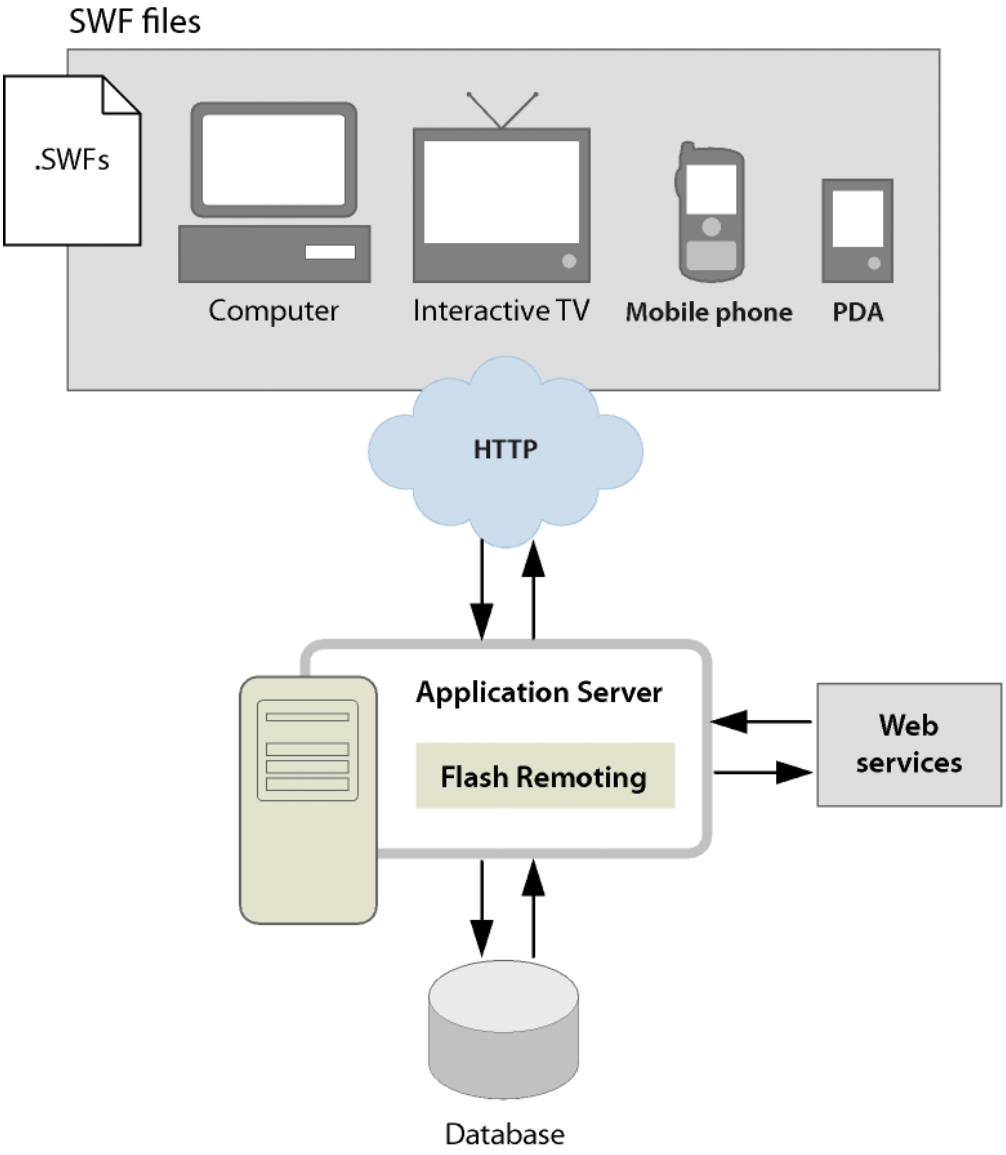
Using the Flash Remoting Service

Using the Flash Remoting service of Adobe ColdFusion, ColdFusion developers can work together with Flash designers to build dynamic Flash user interfaces for ColdFusion applications.
For a complete description of Flash Remoting capabilities, including how ColdFusion interacts with Flash Remoting, see *Using Flash Remoting MX 2004* and *Flash Remoting ActionScript Dictionary* in Flash Help. You can also access the Flash Remoting documentation on the Flash Remoting Developer Center at www.adobe.com/go/learn_cfu_flashremoting_en.

© 2014 Adobe Systems Incorporated. All rights reserved.

About using the Flash Remoting service with ColdFusion

Using the Flash Remoting service of ColdFusion, ColdFusion developers can work with Flash MX 2004 designers to build Flash user interfaces (UIs) for ColdFusion applications. Building Flash UIs requires the separation of user interface code from business logic code. You build user interface controls in Flash, and you build the business logic in ColdFusion.
The following is a simplified representation of the relationship between Flash and ColdFusion:



Planning your SWF application

When you are planning ColdFusion application development with Flash UIs, remember the importance of separating display code from business logic. Separating display code from business logic enables your ColdFusion applications to interact with multiple client types, such as SWF applications, web browsers, and web services.
When you build ColdFusion applications for multiple clients, your ColdFusion pages and components return common data types, including strings, integers, query objects, structures, and arrays. Clients that receive the results can process the passed data according to the client type, such as ActionScript with Flash, or CFML with ColdFusion.
To use the Flash Remoting service with ColdFusion, you build ColdFusion pages and components or deploy Java objects. In ColdFusion pages, you use the Flash variable scope to interact with SWF applications. ColdFusion components (CFCs) natively support Flash interaction. The public methods of Java objects are also available to the

© 2014 Adobe Systems Incorporated. All rights reserved.

Flash Remoting service.
The Flash Remoting ActionScript API has been updated to comply with ActionScript 2.0. The ActionScript 2.0 version of the API consists of the following significant features:

Flash Remoting MX 2004 ActionScript 2.0 API Features
Enforcement of strict data typing, which requires you to declare the data types of variables and prohibits you from assigning different types of data to them.
Enforcement of case sensitivity, which means that `myvar` and `myVar` are two different variables, though they were considered the same variable with different spellings in ActionScript 1.0.
A new Service class, which lets you create a gateway connection and at the same time obtain a reference to a service and its methods. It includes the `connection` property, which returns the connection and also lets you set credentials for authorization on the remote server.Note: The NetServices class is still supported but has been deprecated in favor of the new Service and Connection classes
A new Connection class that helps you create and use Flash Remoting connections.Note: The Connection class supersedes the former NetConnection class.
A new PendingCall object returned on each call to a service method that is run using the Service object. The PendingCall object contains the responder property, which you use to specify the methods to handle the results of the service call.
A new RelayResponder class, which specifies the methods to which the result and fault outcomes of a service call are relayed.
A RecordSet object that contains new properties (`columnNames`, `items`, and `length`), new methods (`clear()`, `contains()`, `editField()`, `getEditingData()`, `getIterator()`, `getLocalLength()`, `getRemoteLength()`, `isEmpty()`, and `sortItems()`), and the new `modelChanged` event.

For more information on the ActionScript 2.0 Flash Remoting API, see Flash Remoting ActionScript Dictionary Help.

© 2014 Adobe Systems Incorporated. All rights reserved.

Configuring the Flash Remoting Gateway

The following parameters in the ColdFusion web.xml file point the Flash Remoting gateway to the gateway-config.xml file.

```
<init-param>
 <param-name>gateway.configuration.file</param-name>
 <param-value>/WEB-INF/gateway-config.xml</param-value>
  </init-param>
<init-param>
 <param-name>whitelist.configuration.file</param-name>
 <param-value>/WEB-INF/gateway-config.xml</param-value>
</init-param>
<init-param>
 <param-name>whitelist.parent.node</param-name>
 <param-value>gateway-config</param-value>
</init-param>
```

Both the web.xml file and the gateway-config.xml file are located in the WEB-INF directory of your ColdFusion server. As a general rule, you do not have to change these web.xml settings.
ColdFusion MX 7 and later versions of ColdFusion configure Flash gateways differently from previous ColdFusion releases. Parameters that worked before this release are no longer supported, and you specify all configuration parameters in the gateway-config.xml file. Also, the Flash gateway now supports a whitelist, which specifies which remote sources can be accessed through the gateway. The two web.xml entries that identify the whitelist must specify your gateway-config.xml file and gateway-config as the parent node.
You can modify the gateway-config.xml file to configure service adapters, add service names to the whitelist, change the logging level, and specify how the gateway handles case sensitivity.
You can configure gateway features in the gateway-config.xml file as follows:

Feature	Description

© 2014 Adobe Systems Incorporated. All rights reserved.

service adapters	By default, the PageableResultSetAdapter, the ColdFusionAdapter, the CFCAdapter (for ColdFusion components), and the CFSSASAdapter (for server-side ActionScript) adapters are enabled in ColdFusion.You can also enable the JavaBeanAdapter, JavaAdapter, EJBAdapter, ServletAdapter, and CFWSAdapter (for web services) by removing their enclosing comment symbols (<!-- and -->). The following service adapter tags are defined as the default tag values:
	`<service-adapters>`
	`<adapter>flashgateway.adapter.resultset.PageableResultSetAdapter</adapter>`
	`<adapter>coldfusion.flash.adapter.ColdFusionAdapter</adapter>`
	`<adapter>coldfusion.flash.adapter.CFCAdapter</adapter>`
	`<adapter>coldfusion.flash.adapter.CFSSASAdapter</adapter>`
	`<!-- <adapter type="stateful-class">flashgateway.adapter.java.JavaBeanAdapter</adapter> -->`
	`<!-- <adapter type="stateless-class">flashgateway.adapter.java.JavaAdapter</adapter> -->`
	`<!-- <adapter type="ejb">flashgateway.adapter.java.EJBAdapter</adapter> -->`
	`<!-- <adapter type="servlet">flashgateway.adapter.java.ServletAdapter</adapter> -->`
	`<!-- <adapter>coldfusion.flash.adapter.CFWSAdapter</adapter> -->`
	`</service-adapters>`

security	You can edit security settings in child tags of the `<security>` tag.In the `<login-command>` tag, you can set the `flashgateway.security.LoginCommand` implementation for performing local authentication on a specific application server. By default, the <login-command> tag is set to the following values:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<login-command>

<class>flashgateway.security.JRunL
oginCommand</class>

<server-match>JRun</server-match>
</login-command>
```

In the `<show-stacktraces>` tag, you can enable stack traces. Stack traces are useful for debugging and product support, but they should not be sent to the client in production mode because they can expose internal information about the system. The following `<show-stacktraces>` tag is the default tag:

```
<show-stacktraces>false</show-stac
ktraces>
```

The `<whitelist>` tag specifies which remote sources can be accessed through the gateway. The * character can be used as a wildcard to imply ALL matches. The following `<whitelist>` tag shows the default value:

```
<whitelist>
  <source>*</source>
</whitelist>
```

When you deploy your application, ensure that you change this setting so that it specifies only the services that the gateway needs to access to run your application. Remember that for ColdFusion based services, directories are treated as "packages" and thus you specify a period delimited path from the web root to the directory or file containing the services you will allow access to. An asterisk wildcard allows access to all services in a particular directory. You can have multiple <source> tags. The following whitelist allows access to the *webroot*/cfdocs/exampleapps/ directory, which includes the flash1 through flash5 Flash Remoting example directories. It also allows access to a *webroot*/BigApp/remoting directory and its children.

© 2014 Adobe Systems Incorporated. All rights reserved.

	`<whitelist>`
	
	`<source>cfdocs.exampleapps.*</source>`
	
	`<source>BigApp.remoting.*</source>`
	
	`</whitelist>`

logger level	You can set the level of logging between `None`, `Error`, `Info`, `Warning`, and `Debug`. The following tag is the default logger level tag: `<logger level="Error">coldfusion.flash.ColdFusionLogger</logger>`
redirect URL	In the `<redirect-url>` tag, you can specify a URL to receive HTTP requests that are not sent with AMF data. By default, the `<redirect-url>` tag is set to {`context.root`}, which is the context root of the web application: `<redirect-url>{context.root}</redirect-url>`
case sensitivity	The `<lowercase-keys>` tag specifies how the gateway handles case sensitivity. ActionScript 1.0 and ColdFusion use case insensitive data structures to store associative arrays, objects and structs. The Java representation of these data types requires a case-insensitive Map, which the gateway achieves by forcing all keys to lowercase.ActionScript 2.0 is case sensitive and requires a <lowercase-keys> tag value of `false`. The following `<lowercase-keys>` tag is the default tag: `<lowercase-keys>true</lowercase-keys>`

© 2014 Adobe Systems Incorporated. All rights reserved.

Using the Flash Remoting service with ColdFusion pages

When you build a ColdFusion page that interacts with a SWF application, the directory name that contains the ColdFusion pages translates to the service name that you call in ActionScript. The individual ColdFusion page names within that directory translate to service functions that you call in ActionScript.

> **Note**
>
> Flash Remoting cannot interact with virtual directories accessed through a ColdFusion mapping.

In your ColdFusion pages, you use the Flash variable scope to access parameters passed to and from a SWF application. To access parameters passed from a SWF application, you use the parameter name appended to the `Flash` scope or the `Flash.Params` array. To return values to the SWF application, use the `Flash.Result` variable. To set an increment value for records in a query object to be returned to the SWF application, use the{{Flash.Pagesize}} variable.
The following table shows the variables contained in the Flash scope:

Variable	Description	For more information
Flash.Params	Array that contains the parameters passed from the SWF application. If you do not pass any parameters, `Flash.params` still exists, but it is empty.	See *Accessing parameters passed from Flash* in Using the Flash Remoting service with ColdFusion pages.
Flash.Result	The variable returned from the ColdFusion page to the SWF application that called the function.**`Note:`** Because ActionScript performs automatic type conversion, do not return a Boolean literal to Flash from ColdFusion. Return `1` to indicate `true`, and return `0` to indicate `false`.	See *Returning results to Flash* in Using the Flash Remoting service with ColdFusion pages.
Flash.Pagesize	The number of records returned in each increment of a record set to a SWF application.	See *Returning records in increments to Flash* in Using the Flash Remoting service with ColdFusion pages.

The following table compares the ColdFusion data types and their ActionScript equivalents:

ActionScript data type	ColdFusion data type
Number (primitive data type)	Number
Boolean (primitive data type)	Boolean (0 or 1)
String (primitive data type)	String

© 2014 Adobe Systems Incorporated. All rights reserved.

ActionScript Object	Structure
ActionScript Object (as the only argument passed to a service function)	Arguments of the service function. ColdFusion pages (CFM files): `flash` variable scope, ColdFusion components (CFC files): named arguments
Null	Null (`Asc( )` returns 0, which translates to not defined)
Undefined	Null (`Asc( )` returns 0, which translates to not defined)
Ordered array*Note:* ActionScript array indexes start at zero (for example: my_ASarray0).	Array*Note:* ColdFusion array indexes start at one (for example: my_CFarray1).
Named (or associative) array	Struct
Date object	Date
XML object	XML document
RecordSet	Query object (when returned to a SWF application only; you cannot pass a RecordSet from a SWF application to a ColdFusion application)

Also, remember the following considerations regarding data types:

- If a string data type on the server represents a valid number in ActionScript, Flash can automatically cast it to a number if needed.
- To return multiple, independent values to the SWF application, place them in a complex variable that converts to a Flash Object, Array, or Associative Array, that can hold all of the required data. Return the single variable and access its elements in the SWF application.
  For a complete explanation of using Flash Remoting data in ActionScript, see Using Flash Remoting MX 2004__Help.

Accessing parameters passed from Flash

To access variables passed from SWF applications, you append the parameter name to the Flash scope or use the `Flash.Params` array. Depending on how the values were passed from Flash, you refer to array values using ordered array syntax or structure name syntax. Only ActionScript objects can pass named parameters.
For example, if you pass the parameters as an ordered array from Flash{{,}} `array1` references the first value. If you pass the parameters as named parameters, you use standard structure-name syntax like `params.name`.
You can use most of the CFML array and structure functions on ActionScript collections. However, the `StructCopy` CFML function does not work with ActionScript collections. The following table lists ActionScript collections and describes how to access them in ColdFusion:

Collection	ActionScript example	Notes

© 2014 Adobe Systems Incorporated. All rights reserved.

Strict array	`var myArray:Array = new Array();`
	`myArray[0] = "zero";`
	`myArray[1] = "one";`
	`myService.myMethod (myArray);`
		The Flash Remoting service converts the Array argument to a ColdFusion array. All CFML array operations work as expected.
		`<cfset p1=Flash.Params[1][1]>`
		`<cfset p2=Flash.Params[1][2]>`

Named or associative array	`var myStruct:Array = new Array();`
	`myStruct["zero"] = "banana";`
	`myStruct["one"] = "orange";`
	`myService.myMethod (myStruct);`
		Named array keys are not case-sensitive in ActionScript.
		`<cfset p1=Flash.Params[1].zero>`
		`<cfset p2=Flash.Params[1].one>`

Mixed array	`var myMxdArray:Array = new Array();`
	`myMxdArray["one"] = 1;`
	`myMxdArray[2] = true;`
		Treat this collection like a structure in ColdFusion. However, keys that start with numbers are invalid CFML variable names. Depending on how you attempt to retrieve this data, ColdFusion sometimes throws an exception. For example, the following CFC method throws an exception:
		`<cfargument name="ca" type="struct">`
		`<cfreturn ca.2>`
		The following CFC method does not throw an exception:
		`<cfargument name="ca" type="struct">`
		`<cfreturn ca["2"]>`

© 2014 Adobe Systems Incorporated. All rights reserved.

Using an ActionScript object initializer for named arguments	`myService.myMethod ({ x:1, Y:2, z:3 });`	This notation provides a convenient way of passing named arguments to ColdFusion pages. You can access these arguments in ColdFusion pages as members of the Flash scope:
		`<cfset p1 = Flash.x>`
		`<cfset p2 = Flash.y>`
		`<cfset p3 = Flash.z>`
		Or, you can access them as normal named arguments of a CFC method.

The `Flash.Params` array retains the order of the parameters as they were passed to the method. You use standard structure name syntax to reference the parameters; for example:

```
<cfquery name="flashQuery" datasource="cfdocexamples">
 SELECT ItemName, ItemDescription, ItemCost
 FROM tblItems
 WHERE ItemName EQ '#Flash.paramName#'
</cfquery>
```

In this example, the query results are filtered by the value of `Flash.paramName`, which{{}}references the first parameter in the passed array. If the parameters are passed as an ordered array from the SWF application, you use standard structure name syntax; for example:

```
<cfset Flash.Result = "Variable 1:#Flash.Params[1]#, Variable 2: #Flash.Params[2]#">
```

Note

ActionScript array indexes start at zero. ColdFusion array indexes start at one.

Returning results to Flash

In ColdFusion pages, only the value of the `Flash.Result` variable is returned to the SWF application. For more information about supported data types between ColdFusion and Flash, see the data type table in Using the Flash Remoting service with ColdFusion pages. The following procedure creates the service function `helloWorld`, which returns a structure that contains simple messages to the SWF application.

Create a ColdFusion page that passes a structure to a SWF application

1. Create a folder in your web root, and name it helloExamples.

© 2014 Adobe Systems Incorporated. All rights reserved.

2. Create a ColdFusion page, and save it as helloWorld.cfm in the helloExamples directory.
3. Modify helloWorld.cfm so that the CFML code appears as follows:

```
<cfset tempStruct = StructNew()>
<cfset tempStruct.timeVar = DateFormat(Now ())>
<cfset tempStruct.helloMessage = "Hello World">
```

In the example, two string variables are added to a structure; one with a formatted date and one with a simple message. The structure is passed back to the SWF application using the `Flash.Result`variable.

```
<cfset Flash.Result = tempStruct>
```

4. Save the file.
Remember, the directory name is the service address. The helloWorld.cfm file is a method of the `helloExamplesFlash` Remoting service. The following ActionScript example calls the helloWorld ColdFusion page and displays the values that it returns:

```
import mx.remoting.*;
import mx.services.Log;
import mx.rpc.*;

// Connect to helloExamples service and create the howdyService service object

 var howdyService:Service = new Service(
   "http://localhost/flashservices/gateway",
   null,
   "helloExamples",
   null,
   null );
// Call the service helloWorld() method
var pc:PendingCall = howdyService.helloWorld();
// Tell the service what methods handle result and fault conditions
pc.responder = new RelayResponder( this, "helloWorld_Result",
"helloWorld_Fault" );

function helloWorld_Result(re:ResultEvent)
{
 // Display successful result
 messageDisplay.text = re.result.HELLOMESSAGE;
 timeDisplay.text = re.result.TIMEVAR;
}

function helloWorld_Fault(fe:FaultEvent)
{
 // Display fault returned from service
 messageDisplay.text = fe.fault;
}
```

© 2014 Adobe Systems Incorporated. All rights reserved.

> **Note**
>
> Due to ActionScript's automatic type conversion, do not return a Boolean literal to Flash from ColdFusion. Return `1` to indicate `true`, and return `0` to indicate `false`.

Returning records in increments to Flash

ColdFusion lets you return record set results to Flash in increments. For example, if a query returns 20 records, you can set the `Flash.Pagesize` variable to return five records at a time to Flash. Incremental record sets let you minimize the time that a SWF application waits for the application server data to load.

Create a ColdFusion page that returns an incremental record set to Flash

1. Create a ColdFusion page, and save it as getData.cfm in the helloExamples directory.
2. Modify getData.cfm so that the code appears as follows:

```
<cfparam name="pagesize" default="10">
<cfif IsDefined("Flash.Params")>
 <cfset pagesize = Flash.Params[1]>
</cfif>
<cfquery name="myQuery" datasource="cfdocexamples">
 SELECT *
 FROM tblParks
</cfquery>
<cfset Flash.Pagesize = pagesize>
<cfset Flash.Result = myQuery>
```

   In this example, if a single parameter is passed from the SWF application, the `pagesize` variable is set to the value of the `Flash.Params`1 variable; otherwise, the value of the variable is the default, 10. Next, a statement queries the database. After the querying, the `pagesize` variable is assigned to the `Flash.Pagesize` variable. Finally, the query results are assigned to the `Flash.Result` variable, which is returned to the SWF application.

3. Save the file.
   When you assign a value to the `Flash.Pagesize` variable, you are specifying that if the record set has more than a certain number of records, the record set becomes pageable and returns the number of records specified in the `Flash.Pagesize` variable. For example, the following code calls the `getData()`function of the CFMService and uses the first parameter to request a page size of 5:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
import mx.remoting.*;
import mx.services.Log;
import mx.rpc.*;
// Connect to helloExamples service and create the CFMService service object
 var CFMService:Service = new Service(
   "http://localhost/flashservices/gateway",
   null,
   "helloExamples",
   null,
   null );
// Call the service getData() method
var pc:PendingCall = CFMService.getData(5);
// Tell the service what methods handle result and fault conditions
pc.responder = new RelayResponder( this, "getData_Result", "getData_Fault" );

function getData_Result(re:ResultEvent)
{
 // Display successful result
 DataGlue.bindFormatStrings(employeeData, re.result, "#PARKNAME#, #CITY#,
#STATE#");
}
function getData_Fault(fe:FaultEvent)
{
 // Display fault returned from service
 trace("Error description from server: " + fe.fault.description);
}
```

In this example, `employeeData` is a Flash list box. The result handler, `getData_Result,` displays the columns PARKNAME, CITY, and STATE in the `employeeData` list box. After the initial delivery of records, the RecordSet ActionScript class assumes the task of fetching records. In this case, the list box requests more records as the user scrolls the list box. You can configure the client-side RecordSet object to fetch records in various ways using the `RecordSet.setDeliveryMode` ActionScript function.

© 2014 Adobe Systems Incorporated. All rights reserved.

Using Flash with CFCs

CFCs require little modification to work with a SWF application. The tag `cffunction` tag names the method and contains the CFML logic, the `cfargument` tag names the arguments, and the cfreturn tag returns the result to the SWF application. The name of the CFC file (*.cfc) translates to the service name in ActionScript.

> **Note**
>
> For CFC methods to communicate with SWF applications, set the `cffunction` tag's `access` att ribute to `remote`.

The following example replicates the `helloWorld` function that was previously implemented as a ColdFusion page. For more information, see Using the Flash Remoting service with ColdFusion pages.

Create a CFC that interacts with a SWF application

1. Create a CFC and save it as flashComponent.cfc in the helloExamples directory.
2. Modify the code in flashComponent.cfc so that it appears as follows:

```
<cfcomponent name="flashComponent">
 <cffunction name="helloWorld" access="remote" returnType="Struct">
  <cfset tempStruct = StructNew()>
  <cfset tempStruct.timeVar = DateFormat(Now ())>
  <cfset tempStruct.helloMessage = "Hello World">
  <cfreturn tempStruct>
 </cffunction>
</cfcomponent>
```

   In this example, the `helloWorld` function is created. The `cfreturn` tag returns the result to the SWF application.

3. Save the file.
   The `helloWorld` service function is now available through the `flashComponent`service to ActionScript. The following ActionScript example calls this function:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
import mx.remoting.*;
import mx.services.Log;
import mx.rpc.*;

// Connect to the Flash component service and create service object
 var CFCService:Service = new Service(
   "http://localhost/flashservices/gateway",
   null,
   "helloExamples.flashComponent",
   null,
   null );
// Call the service helloWorld() method
var pc:PendingCall = CFCService.helloWorld();
// Tell the service what methods handle result and fault conditions
pc.responder = new RelayResponder( this, "helloWorld_Result",
"helloWorld_Fault" );

function helloWorld_Result(re:ResultEvent)
{
 // Display successful result
 messageDisplay.text = re.result.HELLOMESSAGE;
 timeDisplay.text = re.result.TIMEVAR;
}

function helloWorld_Fault(fe:FaultEvent)
{
 // Display fault returned from service
 messageDisplay.text = fe.fault;
}
```

In this example, the `CFCService` object references the `flashComponent` component in the helloExamples directory. Calling the `helloWorld` function in this example executes the function that is defined in `flashComponent`. For ColdFusion components, the component filename, including the directory structure from the web root, serves as the service name. Remember to delimit the path directories with periods rather than backslashes.

© 2014 Adobe Systems Incorporated. All rights reserved.

Using the Flash Remoting service with ColdFusion Java objects

You can run various kinds of Java objects with ColdFusion, including JavaBeans, Java classes, and Enterprise JavaBeans. You can use the ColdFusion Administrator to add additional directories to the classpath.

Add a directory to ColdFusion classpath

1. Open the ColdFusion Administrator.
2. In the Server Settings menu, click the Java and JVM link.
3. Add your directory to the Class Path form field.
4. Click Submit Changes.
5. Restart ColdFusion.
   When you place your Java files in the classpath, the public methods of the class instance are available to your SWF movie. For example, assume the Java class `utils.UIComponents`exists in a directory in your ColdFusion classpath. The Java file contains the following code:

```
package utils;
public class UIComponents
{
 public UIComponents()
 {
 }
 public String sayHello()
 {
  return "Hello";
 }
}
```

Note

You cannot call constructors with Flash Remoting. Use the default constructor.

In ActionScript, the following `javaService` call runs the `sayHello` public method of the `utils.UIComponents` c lass:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
import mx.remoting.*;
import mx.services.Log;
import mx.rpc.*;

// Connect to service and create service object
 var javaService:Service = new Service(
   "http://localhost/flashservices/gateway",
   null,
   utils.UIComponents",
   null,
   null );
// Call the service sayHello() method
var pc:PendingCall = javaService.sayHello();
// Tell the service what methods handle result and fault conditions
pc.responder = new RelayResponder( this, "sayHello_Result", "sayHello_Fault" );

function sayHello_Result(re:ResultEvent)
{
 // Display successful result
 trace("Result is: " + re.result);
}

function sayHello_Fault(fe:FaultEvent)
{
 // Display fault returned from service
 trace("Error is: " + fe.fault.description);
}
```

Note

For more information about using Java objects with ColdFusion, see Using Java objects

© 2014 Adobe Systems Incorporated. All rights reserved.

Handling errors with ColdFusion and Flash

To help with debugging, use `cftry` and `cfcatch` tags in your ColdFusion page or component to return error messages to Flash Player. For example, the ColdFusion page, causeError.cfm, contains the code:

```
<cftry>
 <cfset dev = Val(0)>
 <cfset Flash.Result = (1 / dev)>
 <cfcatch type = "any">
  <cfthrow message = "An error occurred in this service: #cfcatch.message#">
 </cfcatch>
</cftry>
```

The second `cfset` tag in this example fails because it tries to divide by zero (0). The `message` attribute of the `cfthrow` tag describes the error; ColdFusion returns this attribute to the SWF application.
To handle the error in your SWF application, create a fault handler like `causeError_Fault` in the following example:

```
import mx.remoting.*;
import mx.services.Log;
import mx.rpc.*;

// Connect to service and create service object
 var CFMService:Service = new Service(
   "http://localhost/flashservices/gateway",
   null,
   "helloExamples",
   null,
   null );
// Call the service causeError() method
var pc:PendingCall = CFMService.causeError();
// Tell the service what methods handle result and fault conditions
pc.responder = new RelayResponder( this, "causeError_Result", "causeError_Fault" );

function causeError_Result(re:ResultEvent)
{
 // Display successful result
 messageDisplay.text = re.result;
}

function causeError_Fault(fe:FaultEvent)
{
 // Display fault returned from service
 trace("Error message from causeError is: " + fe.fault.description);
}
```

This example displays the trace message from the `causeError_Fault` function in the Flash Output panel. The portion of the message that is contained in `fe.fault.description` is the portion of the message that is contained in `#cfcatch.message#` in the causeError.cfm page.

© 2014 Adobe Systems Incorporated. All rights reserved.

Note

When you create a ColdFusion page that communicates with Flash, ensure that the ColdFusion page works before using it with Flash.

#back to top

© 2014 Adobe Systems Incorporated. All rights reserved.

Using Flash Remoting Update

You can use Flash Remoting Update to create Rich Internet Applications by using Adobe ColdFusion with Adobe Flash Builder or earlier versions of Flex Builder, with the advanced data retrieval features of ColdFusion, such as the `cfpop`, `cfldap`, and `cfquery` tags. In addition, you can use Flash Remoting Update to create Flash Forms and SWF files that contain features, such as server call backs and customized user interface.

© 2014 Adobe Systems Incorporated. All rights reserved.

Prerequisites for using Flash Remoting Update

You can use Flash Remoting Update with all configurations of ColdFusion (Server and JEE) on all the platforms that ColdFusion supports.

To use Flash Remoting Update, you must have the following installed:

- Flex 2 SDK or later, Flex Builder 2 or later, Flash Builder 4.
- Flash Player 8.5 or later

© 2014 Adobe Systems Incorporated. All rights reserved.

Configure Flex Compilation

You use Flash Builder or the Flex SDK to compile Flex applications into SWF files. To use the Flash Remoting Update, these programs must use the ColdFusion `services-config.xml` file when compiling the MXML. You need to configure Flash Builder to use the ColdFusion configuration file, or specify the file when you use the SDK to compile your application (as described in Compile and Run the application).

Configure Flex Builder 2 to use the ColdFusion configuration file

When you use the Flex Builder Project Setup Wizard and select ColdFusion as the server type, the wizard configures Flex Builder to use the services-config.xml file for you. Use the following steps to configure your project:

1. Select File> New> Flex Project to open the New Flex Project Wizard. and enter the appropriate information in the first sections of the Create a Flex project page.
2. Select one of the radio buttons, as follows:
   - Select ColdFusion Flash Remoting to compile in Flex Builder.
   - If you installed LiveCycle Data Services with ColdFusion and want to use messaging or data management, select Flex Data Services.
3. If you select Flex Data services, select whether to compile the application locally in Flex Builder or on the application server where the page is viewed. Do not select to compile code that you deploy on the server; this option is for development purposes only.
4. Click Next and complete creating the project, then click Finish.

If you select Basic on the first Create a Flex Project page, and decide later to compile the application for use with ColdFusion, configure Flex Builder manually, as follows:

1. Select Project > Properties.
2. Select Flex Complier in the right pane of the Properties dialog.
3. In the Additional Compiler arguments add `-services=`followed by the absolute path to the services-config.xml file in the local ColdFusion installation. For example, on a Windows system with a default ColdFusion stand-alone installation, specify the following argument string.

```
-services=C:/ColdFusion9/wwwroot/WEB-INF/flex/services-config.xml
```

Configure Flex Builder 3 to use the ColdFusion configuration file

When you use the Flex Builder Project Setup Wizard and select ColdFusion as the server type, the wizard configures Flex Builder to use the services-config.xml file for you. Use the following steps to configure your project:

1. Select File> New> Flex Project to open the New Flex Project Wizard. and enter the appropriate information in the first sections of the Create a Flex project page.
2. In the Server technology section of the Create a Flex project page, select ColdFusion as the Application server type, and select Use remote object access service.
3. Select one of the radio buttons, as follows:
   - Select ColdFusion Flash Remoting to compile in Flex Builder.
   - If you installed LiveCycle Data Services and want to compile the application on the server, select LiveCycle Data services in Flex Builder 3.
4. Click Next to open the Configure ColdFusion page, and enter the required information. If you selected LiveCycle Data services in step 3, you can select to compile the application locally or on the server. Select to compile on the server only when you are developing your application, for convenience. Do not select to compile on the server code that you deploy, because the MXML page is not compiled to a SWF file until the user requests it, and the compiler does not create an HTML wrapper page.
5. Click Finish to complete the configuration.

© 2014 Adobe Systems Incorporated. All rights reserved.

If you do not specify ColdFusion in the Server technology section of the Create a Flex project page, and decide later to compile the application for use with ColdFusion, configure Flex Builder manually, as follows:

1. Select Project > Properties.
2. Select Flex Complier in the right pane of the Properties dialog.
3. In the Additional Compiler arguments add `-services=`followed by the absolute path to the services-config.xml file in the local ColdFusion installation. For example, on a Windows system with a default ColdFusion stand-alone installation, specify the following argument string.

```
-services=C:/ColdFusion8/wwwroot/WEB-INF/flex/services-config.xml
```

© 2014 Adobe Systems Incorporated. All rights reserved.

Specify a CFC

To specify a CFC to connect to, you do one of the following:

- Specify the dot-delimited path from the web root to the CFC in the MXML.
- Create a named resource for the CFC. Creating this resource is like registering a data source; you then use the resource name in your XML.

Specify the CFC in the MXML

To specify the CFC in your MXML, use code such as the following:

```
<mx:RemotObject
 id="myCfc"
 destination="ColdFusion"
 source="myApplication.components.User"/>
```

ColdFusion 9 supports BlazeDS that allows messaging support for ColdFusion. When you install ColdFusion, the following files are added to the /WEB-INF/flex directory:

- remoting-config.xml
- messaging-config.xml
- services-config.xml
- proxy-config.xml
  The destination `ColdFusion` is preconfigured in remoting-config.xml. The default source value for this destination is the wildcard, `*`. For more information about the changes in Flash Remoting for ColdFusion 9, see *Changes in the XML configuration files for Flash Remoting in ColdFusion 9 and ColdFusion 9.0.1*.
  You do not have to use the ColdFusion destination if you have configured other valid destinations in the configuration file. In this case, the destination definition must specify `*` as the value of its `source` element. If you specify a source other than `*` in remoting-config.xml, then that source definition overrides the source specified in the MXML.

For details of defining a destination, see *Create a named resource for a CFC* below.

Create a named resource for a CFC

1. Edit the WEB-INF/flex/remoting-config.xml file by adding a destination entry for the CFC, for example:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<service id="remoting-service" class="flex.messaging.services.RemotingService"
messageTypes="flex.messaging.messages.RemotingMessage">
 <adapters>
 <adapter-definition id="cf-object"
 class="coldfusion.flash.messaging.ColdFusionAdapter" default="true" />
 <adapter-definition id="java-object"
 class="flex.messaging.services.remoting.adapters.JavaAdapter" />
 </adapters>

  <default-channels>
  <channel ref="my-cfamf" />
  </default-channels>
   <destination id="ColdFusion">
  <channels>

  <channel ref="my-cfamf" />
  </channels>
  <properties>
  <source>*</source>
  </properties>
 </destination>
</service>
```

The `source`attribute specifies the dot notation to the CFC from the web root (the classpath to the CFC). The channel-ref tag refers to the channel-definition in the services-config.xml file. In the preceding sample, the my-cfamf channel-definition has been referenced, which looks similar to the following:

```
<channel-definition id="my-cfamf" class="mx.messaging.channels.AMFChannel">
<endpoint uri="http://{server.name}:{server.port}{context.root}/flex2gateway/"
class="coldfusion.flash.messaging.CFAMFEndPoint" />
 <properties>
  <polling-enabled>false</polling-enabled>
  <serialization>
   <enable-small-messages>false</enable-small-messages>
  </serialization>
  <coldfusion>
   <access>
    <use-mappings>true</use-mappings>
    <method-access-level>remote</method-access-level>
   </access>
   <use-accessors>true</use-accessors>
   <use-structs>false</use-structs>
   <property-case>
    <force-cfc-lowercase>false</force-cfc-lowercase>
    <force-query-lowercase>false</force-query-lowercase>
    <force-struct-lowercase>false</force-struct-lowercase>
   </property-case>
  </coldfusion>
 </properties>
</channel-definition>
```

2. Restart the ColdFusion server.

© 2014 Adobe Systems Incorporated. All rights reserved.

The following table describes the XML attributes for remoting-config.xml:

Element	Description
`destination id`	The `destination` attribute that the MXML mx:RemoteObject tag must specify to access the CFC.
`channels`	A container for one of more child `channel` attributes specifying the AMF channels to use to access the ColdFusion server.
`channel-ref`	Reference to the channel-definition id specified in the services-config.xml file.
`source`	The dot-delimited file path to the CFC, from the *cfWeb Root*, or, if the use-`mappings` property is `true,` an entry in the ColdFusion Administrator Mappings page.
`access`	Properties that control how the CFC is accessed on the ColdFusion server.

The following table lists the XML attributes for services-config.xml:

Elements	Description
`channel-definition id`	Channel definition
`coldfusion`	Contains tags to set access levels, mappings to find CFCs, access to public or remote methods.
`access`	define the resolution rules and access level of the CFC being invoked
`use-mappings`	Use the ColdFusion mappings to find CFCs, by default only CFC files under your web root can be found.
`method-access-level`	Allow "public and remote" or just "remote" methods to be invoked
`use-accessors`	Whether the Value Object CFC has getters and setters. Set the value of use-accessors to true if there are getters and setters in the Value Object CFC.
`use-structs`	Set the value of use-structs to true if you don't require any translation of ActionScript to CFCs. The assembler can still return structures to Flex, even if the value is false. The default value is false.

© 2014 Adobe Systems Incorporated. All rights reserved.

`force-cfc-lowercase`
`force-query-lowercase`
`force-struct-lowercase`
	Whether to make property names, query column names, and structure keys lowercase when converting to ActionScript. Query column names must precisely match the case of the corresponding ActionScript variables. The default value is `false`.

© 2014 Adobe Systems Incorporated. All rights reserved.

Use the CFC

1. In the MXML file, you use the `<mx:RemoteObject>` tag to connect to your CFC named resource. With this connection, you can call any remote method on the CFC.
2. If you created a destination for the CFC in the remoting-config.xml file, specify the destination name in the `mx:RemoteObject`tag; for example:

```
<mx:RemoteObject
 id="a_named_reference_to_use_in_mxml"
 destination="CustomID"
 result="my_CFC_handler(event)"/>
```

   If you did not create a destination for the CFC, specify the ColdFusion destination and the CFC path in the `mx:RemoteObbject`tag; for example:

```
<mx:RemotObject
 id="myCfc"
 destination="ColdFusion"
 source="myApplication.components.User"/>
```

3. Call a CFC method, for example, as the following example shows:

```
<mx:Button label="reload" click="my_CFC.getUsers()"/>
```

   In this example, when a user presses a button, the Click event calls the CFC method getUsers.

4. Specify a handler for the returned result of the CFC method call for the `<mx:RemoteObject>`tag, as the following example shows.

```
private function my_CFC_handler( event:ResultEvent )
{
// Show alert with the value that is returned from the CFC.
mx.controls.Alert.show(ObjectUtil.toString(event.result));
}
```

© 2014 Adobe Systems Incorporated. All rights reserved.

Compile and Run the application

You can compile and run your application from Flash Builder or without using Flash Builder. The techniques you can use also depend on whether you have installed LiveCycle Data Services.

Compile and run the application using Flash Builder

To compile and run an application using Flash Builder, make sure that Flash Builder is configured as described in Configure Flex Compilation. Compile your application normally to create a SWF file. When you configure your Flash Builder project you can specify the location in which to place it. By default, Flash Builder attempts to place the SWF file and an HTML wrapper page under the web root. You can then run the application as appropriate, for example, by requesting the HTML wrapper for the SWF file in a browser.

Compile and Run the application without Flash Builder

To compile the application directly using the SDK, set the Flex compiler to use the ColdFusion services-config.xml file. Set the Flex compiler by adding to the mxml command line " `-services=`" followed by the absolute path to the services-config.xml file in the local ColdFusion installation. For example, on a Windows system with a default ColdFusion standalone installation, specify the following argument string.

```
-services=C:/ColdFusionCentuar/wwwroot/WEB-INF/flex/services-config.xml
```

© 2014 Adobe Systems Incorporated. All rights reserved.

Lazy loading across client and server

This release supports need-based loading of related entities for applications that use ColdFusion ORM in the back end and Flex as the front end. Your application can now fetch the main entity and not return the related entities. Only when the client application tries to access the related entities, they are loaded.

Example

Maria uses ColdFusion ORM and is building a Flex application. She wants to list all employees and projects that each employee in her company is part of. Using lazy loading, users in her company can fetch the employee information first and the number of projects the employee has worked on. And later when the users click on a particular project, the feature lets them load the project information of a particular employee rather than having all the data loaded initially when the application is loaded (which was the behavior in ColdFusion 9).

Setting up lazy loading

For lazy loading enhancements to take effect, you need to perform various configurations on both client-side and server-side.

Server-side configuration

1. In the services-config.xml, go to the section that specifies the channel-definition (for the channel that you use) and search the following:**`<serialize-array-to-arraycollection>*false`**`*`
2. Change the value to `true`.
3. (Optional) If you want to change the name of the method that is invoked while related entities are loaded, modify the name of the method from `loadProxy`.**`<proxy-load-method>*loadProxy`**`</proxy-load-method>*`
4. Add the loadProxy method to your service CFC as shown in the following sample code. The service CFC should be in the same directory as your ORM components:service.cfc

```
component {
remote function loadProxy(any proxyKey, any fullyqualifiedname)
 {
 //writeDump(var="#proxyKey#,,,#fullyqualifiedname#",output="console" );
 writedump(var="#proxykey#",output="console");
 if(fullyqualifiedname contains "cproducts"){
 return EntityLoadByPK("cproducts",proxykey );
 }else{
 return EntityLoadByPK("ccategories",proxyKey );
 }
 }
}
```

   The `loadProxy` method gets `proxyKey` (primary key of the entity instance) and `fullyqualifiedname` of the entity as arguments.The `fullyqualifiedname` sent by `dphibernate` is checked and the required object (in this case `cproducts` or `ccategories`) is returned.

5. Set `remotingFetch` to `lazy`. `lazy` is the new value added to `remtoingFetch` (in addition to `true` and `false`) in the tag `cfproprety`:

- `true`: The value of the property is sent to Flash by way of Flash Remoting.
- `false`: Null is sent to Flash.
- `lazy`: Proxy object for the related entities is sent with only the primary key value.Only when any property on the proxy object is accessed, another remoting call reaches the load method defined on the CFC to which the

© 2014 Adobe Systems Incorporated. All rights reserved.

primary key is passed. The load method returns the object with all the values populated.

Sample configuration

employee.cfc

```
<cfcomponent persistent="true" table="employees">
 <cfproperty name="employeeID" fieldtype="id" generator="native"/>
 <cfproperty name="personalObj" fieldtype="one-to-one" cfc="cpersonal" cascade="all"

             remotingfetch="lazy"/>
 <cfproperty name="lastName"/>
 <cfproperty name="firstName"/>
</cfcomponent>
```

personal.cfc

```
<cfcomponent persistent="true" table="personal" >
 <cfproperty name="personalID" generator="foreign" params="{property=EmployeeObj}">
    <cfproperty name="employeeObj" fieldtype="one-to-one" cfc="employee
constrained="true">
 <cfproperty name="SSN">
 <cfproperty name="fatherName">
</cfcomponent>
```

service.cfc

```
component {
remote function loadProxy(any proxyKey, any fullyqualifiedname)
 {
 //writeDump(var="#proxyKey#,,,#fullyqualifiedname#",output="console" );
 writedump(var="#proxykey#",output="console");
 if(fullyqualifiedname contains "cproducts"){
 return EntityLoadByPK("cproducts",proxykey );
 }else{
 return EntityLoadByPK("ccategories",proxyKey );
 }
 }
}
```

Client-side configurations

1. Add dpHibernate.swc to your Flex project. The file can be found in the following directory: /CFIDE/scripts/.
2. Use the `HibernateRemoteObject` instead of the default `RemoteObject` to make remoting calls.
3. Set `HibernateManaged.defaultHibernateService` to the Remote Object instance. The dpHibernate.swc uses this remote object instance to make load calls to the server for lazy loading.
4. Ensure the following:
5. ActionScript class is mapped to the CFC using the attribute `alias` in the `RemoteClass`.For example, `RemoteClass(alias="orm.employees")`

© 2014 Adobe Systems Incorporated. All rights reserved.

6. ActionScript class extends from org.dphibernate.core.HibernateBean.
7. Managed metadata is added to the ActionScript class.
8. Perform Flash Remoting to fetch the entities.The related entities are loaded only when specifically accessed on the client.

Example

Category.as

```
package model.beans
{
      import mx.collections.ArrayCollection;

      import org.dphibernate.core.HibernateBean;

      [RemoteClass(alias="lazyloading.o2m.ccategories")]
      [Managed]
      public class Category extends org.dphibernate.core.HibernateBean
      {
            public function Category()
            {
            }

            public var categoryID:Number;
            public var categoryName:String;
            public var description:String;
            public var products:ArrayCollection;
      }
}
Product.as
package model.beans
{
      import mx.collections.ArrayCollection;

      import org.dphibernate.core.HibernateBean;
      [RemoteClass(alias="lazyloading.o2m.cproducts")]
      [Managed]
      public class Product extends org.dphibernate.core.HibernateBean
      {
            public function Product()
            {
            }

            public var productID:Number;
            public var productName:String;
            public var categoryID:Category;
      }
}
LazyLoading.MXML
<?xml version="1.0" encoding="utf-8"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
                     xmlns:s="library://ns.adobe.com/flex/spark"
                     xmlns:mx="library://ns.adobe.com/flex/mx"
                     xmlns:dp="org.dphibernate.rpc.*"
                     applicationComplete="onAppComplete(event)" minWidth="955"
minHeight="600" xmlns:dphibernate="http://www.dphibernate.org/2010/mxml">
```

© 2014 Adobe Systems Incorporated. All rights reserved.

```
      <fx:Declarations>
            <!-- Place non-visual elements (e.g., services, value objects) here -->
            <dp:HibernateRemoteObject destination="ColdFusion"
source="newmanual.apollounit.orm.lazyloading.o2m.service"

endpoint="http://localhost:8500/flex2gateway/lazyloading" id="BasicService"

result="BasicService_resultHandler(event)" fault="BasicService_faultHandler(event)">

            </dp:HibernateRemoteObject>
      </fx:Declarations>
      <fx:Script>
            <![CDATA[
                  import model.beans.*;

                  import mx.binding.utils.BindingUtils;
                  import mx.collections.ArrayCollection;
                  import mx.rpc.AsyncToken;
                  import mx.rpc.events.FaultEvent;
                  import mx.rpc.events.ResultEvent;

                  import org.dphibernate.rpc.HibernateManaged;

                  [Bindable]
                  public var categories:ArrayCollection;
                  [Bindable]
                  public var ProdsList:ArrayCollection;
                  [Bindable]

                  public var displaytext:String;
                  public var category:Category;
                  public var product:Product;
                  public var employee:Employee;
                  public var employee_self123:Employee_self;

                  protected function
BasicService_resultHandler(event:ResultEvent):void
                  {
                        if( event.token.operation == "getCategories"){
                              categories = event.result as ArrayCollection;
                              var len:Number = categories.length;
                              allintext.text = event.result.length;
                        }else if(event.token.operation == "getProducts"){
                              ProdsList = event.result as ArrayCollection;
                        }
                  }


                  protected function
BasicService_faultHandler(event:FaultEvent):void
                  {
                        allintext.text="There is an error";
                  }

                  protected function onAppComplete(event:Event):void
                  {
                        HibernateManaged.defaultHibernateService=BasicService;
                        //var token:AsyncToken = BasicService.getCategories();
                        //token.operation="getCategories";
```

© 2014 Adobe Systems Incorporated. All rights reserved.

```
                    }

                    protected function load_clickHandler(event:MouseEvent):void
                    {
                          var token:AsyncToken = BasicService.getCategories();
                          token.operation="getCategories";
                    }
              ]]>
      </fx:Script>
      <s:TextArea x="10" y="39" id="allintext" text="hello"/>
      <s:Button x="240" y="10" label="Load Categories" id="load"
click="load_clickHandler(event)"/>
      <mx:DataGrid x="240" y="39" id="CategoryList" dataProvider="{categories}">
              <mx:columns>
                    <mx:DataGridColumn headerText="CategoryName"
dataField="categoryName"/>
                    <mx:DataGridColumn headerText="CategoryID"
dataField="categoryID"/>
              </mx:columns>
      </mx:DataGrid>
      <mx:DataGrid x="239" y="198" id="ProductList"
dataProvider="{Category(CategoryList.selectedItem).products}">
              <mx:columns>
                    <mx:DataGridColumn headerText="ID" dataField="productID"/>
                    <mx:DataGridColumn headerText="Name" dataField="productName"/>
              </mx:columns>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

```
        </mx:DataGrid>
  </s:Application>
```

ccategories.cfc

```
<cfcomponent persistent="true" table="categories">
 <cfproperty name="categoryID" fieldtype="id" generator="native"/>
 <cfproperty name="categoryName"/>
 <cfproperty name="description"/>
 <cfproperty name="products" fieldtype="one-to-many" cfc="cproducts" cascade="all"
             fkcolumn="categoryid" lazy="true" remotingfetch="lazy"/>
</cfcomponent>
```

cproducts.cfc

```
<cfcomponent persistent="true" table="products">
 <cfproperty name="productID" fieldtype="id" generator="native"/>
 <cfproperty name="productName"/>
 <cfproperty name="categoryID" fieldtype="many-to-one" cfc="ccategories" lazy="true"

             remotingfetch="lazy"/>
</cfcomponent>
```

Note on using debugger/network monitor in FlashBuilder

When you inspect the results returned by the server in debugging mode, debugger fetches the related entities (and defies the purpose of lazy loading). This can also occur when you use network monitor.
However, this issue does not occur when you run the application.

#back to top

© 2014 Adobe Systems Incorporated. All rights reserved.

Offline AIR Application Support

ColdFusion provides offline AIR application support, which includes data persistence and synchronization. These features let an AIR application use a local SQLite database that represents data on the ColdFusion server.
You cannot use these features in applications built with Flash, which run in a browser or Flash Player. These features only support AIR applications with intermittent connectivity to the ColdFusion data provider. They enable users to run the AIR application offline and then synchronize data with the ColdFusion application the next time the application runs online.
To support offline AIR data access, you code ActionScript elements on the client side and CFML on the server side.

Note

Some of the code in the following discussion uses an AIR application that displays and updated an Employee database that ColdFusion manages for its sample code. However, the snippets below are not all from this example, and do not make up a complete or consistent application.

© 2014 Adobe Systems Incorporated. All rights reserved.

ColdFusion server side

The ColdFusion application uses a CFC to represent the data being exchanged and synchronized. For example, you could have an ORM employee component with a structure as follows:

```
<cfcomponent persistent="true" displayname="EMP">
 <cfproperty name="id" type="numeric" fieldtype="id" generator="native">
 <cfproperty name="firstName" type="string">
 <cfproperty name="lastName" type="string">
 ...
 <cfproperty name="countryCode" type="string">
</cfcomponent>
```

You can also use a traditional non-ORM CFC. In this case, the Fetch and Sync Methods use the `cfquery` tag and related tags and function for database operations.
To manage interactions with the AIR application and keep the data synchronized, ColdFusion application uses a component called the SyncManager. The SyncManager implements the CFIDE.AIR.ISyncManager interface. The component has two functions:

- A `fetch` function that the AIR application calls to get data from ColdFusion. This function is not part of the ISyncManager interface, but is required. The function can have any arbitrary name, but is called `fetch` by convention.
- A sync function that the AIR application calls to synchronize the ColdFusion and AIR data sources when the application updates or changes data. This function takes three parameters:

- **`operations`** An array of operations to perform INSERT, UPDATE, or DELETE.
- **`clientObjects`** An array of objects, where each item in the array represents the client's current view of the data to be synchronized.
- **`originalObjects`** An array of objects, each item in this Array represents the corresponding original data from the server (if any), such as an existing employee record that a user is updating. For an INSERT operation, this object is null. For a DELETE operation, this object is normally the same as the current data. Incase of Conflict during Sync process, the sync function returns to the AIR client an Array of "CFIDE.AIR.Conflict.cfc" objects. Each of this Conflict object consists of a single serverObject element. The sync function sets the element to equal the server copy of the record that is in conflict. The client application can then handle the conflict as described in Conflict management.

Server-Side notes

- When the `sync` function performs a DELETE operation, it gets the primary key ID from the OriginalObject of the Sync method, as the ClientObject is NULL. For update and insert operations, use the ClientObject key value.
- When you do an INSERT operation, the CFC checks whether the OriginalObject parameter of the sync method is a simple value, as in the following code:

  ```
  {NOT IsSimpleValue(OriginalObject)}
  ```

  In an INSERT operation, OriginalObject passed to the Sync function is null. So if you attempt to retrieve any of its properties, you get a Method NOT Found error. For Example, OriginalObject.GetID results in a Method GetID() not found error. So, for Insert operation, use ClientObject to access various fields.

- While a ColdFusion application can use cfquery to directly manage the database, most AIR applications are

© 2014 Adobe Systems Incorporated. All rights reserved.

expected to use the ORM feature. The discussion here uses ColdFusion ORM for server-side data management.

- You may see the following kind of error message if you are using ColdFusion 8 Remoting with AIR offline applications, which have server side "Sync" method using ORM `EntitySave()`/`EntityDelete()`methods .

```
Error handling message: flex.messaging.MessageException: Unable to invoke CFC
- a different object with the same identifier value was already associated
with the session: [address#1].
Root cause:org.hibernate.NonUniqueObjectException: a different object with the
same identifier value was already associated with the session: [address#1]
```

You may also encounter this error with ColdFusion 9 Remoting but only for `EntityDelete` method.To resolve this sort of error, call your `EntitySave`/`EntityDelete` method in following way in "Sync" method.

```
<cfif operation eq "INSERT" OR operation eq "UPDATE">
 <cfset obj = ORMGetSession().merge(clientobject)>
 <cfset EntitySave(obj)>
<cfelseif operation eq "DELETE">
 <cfset obj = ORMGetSession().merge(originalobject)>
 <cfset EntityDelete(obj)>
</cfif>
```

- In case of a conflict, the `sync` function returns an array of `"CFIDE.AIR.Conflict"` objects to the client. There are four properties a conflict object can have: `operation`,`serverobject`,`clientobject`,`originalobject`. The `serverobject`property of the conflict object must be a user-defined CFC type that represents the server-side database table. The following example generates a conflict object with a valid ServerObject property of type employee.cfc, which represents the Employee table:

```
<cfset serverobject = EntityLoadByPK("employee",originalobject.getId())>
<cfset conflict = CreateObject("component","CFIDE.AIR.conflict")>
<cfset conflict.serverobject = serverobject>
<cfset conflict.clientobject = clientobject>
<cfset conflict.originalobject = originalobject>
<cfset conflict.operation = operation>
<cfset conflicts[conflictcount++] = conflict>
<cfreturn conflicts>
```

If you are using ColdFusion ORM, you can replace the preceding example with the following code.

```
<cfset conflict = CreateObject("component","CFIDE.AIR.Conflict")
<cfset serverobject = EntityLoadByPK("employee",#res.IDENTITYCOL#)>
<cfset conflict.SetServerobject(serverobject)>
```

- When an AIR client with stale data tries to update an already deleted record from the database, server throws the conflict, and the client's conflict handle, which has the `KeepAllServerObjects` or `KeepServerObjec`

© 2014 Adobe Systems Incorporated. All rights reserved.

tmethod accepts the changes from the server. However, the client method does not delete the stale record, which no longer exists in the server database, from the client database.To prevent this issue: The serverObject property of the conflict object returned by the server must be null, if the record that the client requests for updating is no longer in the database. For example:

```
<cfset serverobject = EntityLoadByPK("employee",originalobject.getId())>
<!----If the operation is INSERT, serverObject is also NULL.hence NEQ
condition---->
<cfif not isdefined('serverobject') and operation NEQ "INSERT" >
<cflog text="CONFLICT::SERVER OBJECT NOT FOUND, RECORD MAY BE DELETED
ALREADY">
<cfset conflict = CreateObject("component","CFIDE.AIR.conflict")>
<cfset conflict.clientobject = clientobject>
<cfset conflict.originalobject = originalobject>
<cfset conflict.operation = operation>
<cfset conflicts[conflictcount++] = conflict>
<cfcontinue>
</cfif>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

Offline AIR application code constructs

To code an AIR application that synchronizes with ColdFusion, include the cfair.swc file in your AIR project. This file contains all the ColdFusion client-side code to support interactions between AIR and ColdFusion. The file is installed with ColdFusion in the_ cf_webroot_/CFIDE/scripts/AIR directory. In Flash Builder, specify the swc file location in the Project > Properties > FlexBuildPath > Library Path > Add SWClibrary dialog.

Data object

The AIR application represents the managed data in an ActionScript object that corresponds to the ColdFusion-side data CFC. For example, in the Employee example, the AIR application has an Employee.as file containing an Employee ActionScript class that corresponds to ColdFusion employee.cfc:

```
package test.basic
 {
  [Bindable]
  [RemoteClass(alias="AIRIntegration.employee")]
  [Entity]
  public class Employee
  {
  /** The user id of the employee **/
  [Id]
  public var id:int;
  public var firstName : String;
  public var lastName : String;
  public var displayName : String;
  ....
  public var countryCode : String = 'US';
  ....
  }
```

Note

You do not need to create any SQLite databases or tables; they are created automatically. For example, once the Employee class is defined as above, the first time you invoke the class, the equivalent SQLite table is created for server data persistence.

Data object metadata

You use the following metadata to define the data object:

Metadata Element	Purpose
Entity	Specifies that instances of this class can be persisted to the SQLite database. This element is required.
Table(name="tableName")	The name of the SQLite table in which the object is to be stored. Defaults to the name of the class.

© 2014 Adobe Systems Incorporated. All rights reserved.

Id	Precedes a field definition. Indicates that the field is a primary key in the table. For composite keys, use Id tags on all the primary key fields.
Transient	A Boolean value specifying whether the property or field is persistent. A True value indicates that the field is not persistent and so it is not a part of the client side Sqlite table.
Column(name="name", columnDefinition="TEXT),	Specifies the SQLite database column that contains data for this field. **name:** Column name. If not specified, defaults to the property name. **columnDefinition:** The SQL Datatype to use for the column. **nullable:** Specifies whether a field can have a null value. **unique:** Specifies whether the value for each field must be unique within the column.
RemoteClass	Used for all remote objects, not just ColdFusion. The alias attribute identifies the corresponding class on the remote server. This information is used to map between ActionScript data types and the remote data types.It is mandatory that you specify the RemoteClass metadata tag for the ActionScript classes or entities that map with the server-side CFC. If you do not specify this metadata tag, you get a runtime error. For example, you specify the alias name for the Address entity as follows: RemoteClass(alias="myFolder.AIRIntegration.Address")The alias name must be unique within the AIR application.

Note

For private properties in a class, set Column metadata on the accessor functions (getxxx and setxxx) and not on the private property, as shown in the following code:

```
private var name:String; // Private property
[Column(name="FNAME",columnDefinition="VARCHAR")]
public function set fname(name:String):void // accessor function
{
  this.name = name;
}
public function get fname():String // accessor function
{
  return name;
}
```

© 2014 Adobe Systems Incorporated. All rights reserved.

Client Side

ColdFusion 9 extends offline application support to the client side of the application by letting you code ActionScript elements on the client side. The data that is exchanged and synchronized on the client side is managed through persistent objects in the local or offline database.

Managing relationships

The ActionScript persistent framework lets you define the following relationship types between two persistent objects.

- one-to-one
- one-to-many
- many-to-one
- many-to-many
  To understand how the persistent framework handles relationships, let us consider an example of the Employee and Department objects in a database.
  If you do not specify attribute values, the default values are taken as follows:

- The default table name is the class name.
- The default value for `columnDefinition` is the ActionScript type of the field.
- The default value for `referencedColumnName` is the primary key of the target entity.
- The default value for `targetEntity` is the ActionScript type of the referring field.

> **Note**
>
> In case you are using ORM CFCs, the remotingFetch attribute in the <cfproperty> tag is set to false by default for all relationships. You must set this attribute to true to retrieve data on the client side.

One-to-one relationship

Consider a one-to-one relationship where one employee belongs to a single department. You can use code like the following to define a one-to-one mapping between the Department and Employee objects with DEPTID as the foreign key column name.

```
[Entity]
[Table(name="employee")]
public class Employee
 {
  [Id]
  var id:uint;
  [OneToOne(targetEntity="Department"|fetchType="EAGER(default)|LAZY")]
  [JoinColumn(name="DEPTID", referencedColumnName="DEPT_ID")]
  var dept:Department;
 }
```

The `JoinColumn}}tag specifies the foreign key column and all the attributes of the column tag. Do not specify {{JoinColumn` for both the entities in the relationship. For example, in the one-to-one relationship between the Department and Employee objects, specify {{JoinColumn}}only for one of the entities depending on the direction of the relationship.

© 2014 Adobe Systems Incorporated. All rights reserved.

`referencedColumnName` specifies the primary key column that it refers to. `Class` indicates the target entity, which is `Department` in this example.
The default `fetchType` value is `EAGER`. See *Lazy loading and fetch type* below for information on fetch types.

One-to-many relationship

Consider a one-to-many relationship where one employee belongs to many departments. You can use code like the following to define a one-to-many mapping between the Department and Employee objects.

```
public class Employee
 {
  [Id]
  var id:uint;
  [OneToMany(targetEntity="Department",mappedBy="department",
  fetchType="EAGER|LAZY(default)")]
  var depts:ArrayCollection;
 }
```

There is no column specified in the Employee table but refers to the field in the Department entity that points to the Employee entity.
The default `fetchType` value is `LAZY`. See *Lazy loading and fetch type* below for information on fetch types

Many-to-one relationship

Consider a many-to-one relationship where many employees belong to a single department. You can use code like the following to define a many-to-one mapping between the Department and Employee objects.

```
public class Employee
 {
  [Id]
  var id:uint;
  [ManyToOne(targetEntity="Department",fetchType="EAGER(default)|LAZY")]
  [JoinColumn(name="deptId",referencedColumnName="DEPT_ID")]
  var dept:Department;
 }
```

The default `fetchType` value is `EAGER`. See *Lazy loading and fetch type* below for information on fetch types

Many-to-many relationship

Consider a many-to-many relationship where many employees belong to many departments. You can use code like the following to define a many-to-many mapping between the Department and Employee objects.

© 2014 Adobe Systems Incorporated. All rights reserved.

```
public class Employee
 {
  [Id]
  [Column(name="ID")]
  var id:uint;
  [ManyToMany(targetEntity="Department",
  fetchType="EAGER|LAZY(default)")]
  [JoinTable(name="EMP_DEPT")]
  [JoinColumn(name="EMPID", referencedColumnName="ID")]
  [InverseJoinColumn(name="DEPID", referencedColumnName="DEPTID")]
  var depts:ArrayCollection;
 }
```

The default `fetchType` value is `LAZY`. See *Lazy loading and fetch type* below for information on fetch types.
For a many-to-many relationship, you specify metadata like the following only on one of the entities and not both.

```
[JoinTable(name="ORDER_PRODUCT")]
[JoinColumn(name="ORDER_ID",referencedColumnName="oid")]
[InverseJoinColumn(name="PRODUCT_ID",referencedColumnName="pid")]
```

The `JoinColumn` tag specifies the foreign key column and all the attributes of the column tag. The `InverseJoinColumn` tag specifies the reference to the joining entity in the `JoinTable` tag. In this example, join table "EMP_DEPT" has a column named "DEPID", which refers to the "DEPTID" column of the Department table.
The `JoinTable` tag defines the join table for the many-to-many relationship specifying the join column and inverse join column. In this example, a join table named "EMP_DEPT" is created in the Offline SQLite DB with a many-to-many relationship between the Employee and Department tables.

Lazy loading and fetch type

The ActionScript persistent framework supports lazy loading although it may not be as intuitive because of the asynchronous connection with the database.
The fetch type `EAGER` or `LAZY` determines the fetch type for the relationship that is loaded. An EAGER fetch type loads the relationship and fetches data when the call is first made. A LAZY fetch type loads the relationship and fetches data only when an explicit fetch call is made. The default value for the fetch type is `EAGER` and the default value for `ignoreLazyLoad` is `false`.
When you specify `fetchType="EAGER"` at the class-definition level, the `loadByPk` function always loads the related object, irrespective of the value you specify for the ignoreLazyLoad parameter.
When you specify `fetchType="LAZY"` at the class-definition level, you can have two possibilities:

- When you specify the `ignoreLazyLoad` parameter as `true`, the related object is also loaded. For example, if you have two related objects Address and Customer, and specify `loadByPK(Customer,{id:3},true)`, the Address object is also loaded.
- When you do not specify any value for the `ignoreLazyLoad` parameter, it takes the default value that is `false`, and the related object is not loaded. For example, if you have two related objects Address and Customer, and specify `loadByPK(Customer,{id:3})`, the Address object is not loaded.

© 2014 Adobe Systems Incorporated. All rights reserved.

Cascading options

Cascading lets you specify the operations to be cascaded from the parent object to the associated object. The supported operations are INSERT, UPDATE, and DELETE. The `cascadeType` attribute lets you set any of the following values.

- **ALL** If The source entity is inserted, updated, or deleted, the target entity is also inserted, updated, or deleted.
- **PERSIST** If The source entity is inserted or updated, the target entity is also inserted or updated.
- **REMOVE** If The source entity is deleted, the target entity is also deleted.
  The one-to-one, one-to-many, many-to-one, and many-to-many relationships are all supported by cascading. You can use code like the following to specify the cascading options:

```
ManyToMany(cascadeType="ALL or PERSIST or REMOVE")
```

If you do not specify the `cascadeType` option, only the source entity is persisted or updated.
When you specify `cascadeType='ALL or REMOVE'}}to remove the parent object and the associated child objects, load the parent object using {{load***()` method and pass it through `session.remove(parentObj)`. If you do not use this method of loading, only the parent object gets deleted from SQLite database and the child objects remain.

Note

If you have enabled lazy loading by specifying `fetchType='LAZY'` at the entity level, when you load a parent object using `load**()` **method, the child objects are not loaded. When you specify `cascadeType='ALL or REMOVE'` and try to delete the parent object by passing it through `session.remove(parentObj)`, it does not delete the child objects. To overcome this limitation, use the `load**()` method with `ignorelazyloading='true'`.**

© 2014 Adobe Systems Incorporated. All rights reserved.

Using the AIR SyncManager class to manage data

The AIR application uses the SyncManager class to fetch data from the server and synchronize the local data with the ColdFusion data source. The SyncManager uses a coldfusion.air.Session object to manage the session between the client and the local SQLite database, and uses calls to the following methods in the ColdFusion sync manager CFC:

- fetch to get data from the remote system
- sync to synchronize the local and remote data
  The following text describes basic functionality that you must implement. For details on the SyncManager and Session classes, and other classes in the coldfusion.air package, see ActionScript 3.0 Reference. Alternatively, you can see the standalone *Adobe ColdFusion ActionScript Language Reference,* which is accessible through the Documentation link on the Resources page of the ColdFusion Administrator.
  The AIR application init() function creates and configures a SyncManager instance, and fetches the initial data from ColdFusion as shown in the following code:

```
private function init():void
{
syncmanager = new SyncManager();
//The ColdFusion server and port. Port without double quotes as it is

//expected to be integer and IP is taken as String.
syncmanager.cfPort = CFServerPort;
syncmanager.cfServer = "CFServerIP";

//The CFC that implements the ColdFusion sync manager. Here
//AIRIntegration is the user defined folder under webroot.
syncmanager.syncCFC = "AIRIntegration.empManager";

//Specify a user-defined CF destination,if not specified, default destination
//'ColdFusion' will be used
syncmanager.destination = 'USerDefinedCFDestination'

//The event listener for conflict events returned byt the CFC
syncmanager.addEventListener(ConflictEvent.CONFLICT, conflictHandler);

//The local database file
var dbFile:File = File.userDirectory.resolvePath("EmpManager.db");

//Create a session object, which handles all interactions between the
//AIR application and the SQLite database.
var sessiontoken:SessionToken = syncmanager.openSession(dbFile, 999);

//Add a responder for handling session connection results.
sessiontoken.addResponder(new mx.rpc.Responder(connectSuccess,
connectFault));
}
```

Fetching data from the server

Use the SyncManager fetch method to fetch data from the ColdFusion server by calling the fetch method of the server data object. The `syncManager.fetch` method takes the name of the CFC fetch method (typically fetch) as its first parameter, followed by any CFC fetch method parameters.
The `syncManager.fetch` method returns an AsyncToken object that provides access to the data. The function

© 2014 Adobe Systems Incorporated. All rights reserved.

returns the token synchronously. The ColdFusion CFC response returns asynchronously. Therefore, call the token's addResponder method to specify a responder that handles the responses for successful and failed fetches.
To fetch the initial data from the server, you can include the following code in the Application `init()` method.

```
// fetch the data.
var token:AsyncToken = syncmanager.fetch("fetch");
//Specify the responder to handle the fetch results.
token.addResponder(new mx.rpc.Responder(fetchSuccess, fetchFault));
```

Managing the local database

You use a Session object to manage the data in the local SQLite database. You call the syncmanager.openSession method to create a session with a specific database file. The method returns a SessionToken token, and the SessionToken.session property provides access to the session. You use code in the token's openSessionSuccess responder event handler to provide access to the session object. This way, you do not access the session, and therefore the database, until it is successfully opened.
The following code expands on the session initialization code that was shown above. and shows an openSessionSuccess event handler that uses the session to save the contents of the remote database in the local image. In this example, users is the array collection fetched from server:

```
var dbFile:File = File.userDirectory.resolvePath("basic.db");
var token:SessionToken = syncmanager.openSession(dbFile, 113);
token.addResponder(new mx.rpc.Responder(openSessionSuccess, openSessionFault));
function openSessionSuccess(event:SessionResultEvent):void
{
  //Initialize a variable for access to the session.
  var session:Session = event.sessionToken.session;
  //Save the remote database data in the local database.
  //users is the array collection fetched from server
  var saveToken:SessionToken = session.saveCache(users);
  //Add responder event handlers for successful and failed saves.
  saveToken.addResponder(new mx.rpc.Responder(saveSuccess, saveFault));
}
```

If `Begintransaction()` function doesnt have a corresponding `Committransaction()` function, then the AIR side SQLite DB file is locked. To avoid this, use the following code at the end of an event flow.

```
var closetoken:SessionToken = session.close();
closetoken.addResponder(new mx.rpc.Responder(closesuccess, closefault));
```

Once you have access to the session, you can get (load) data from the SQLite database, and insert, delete, and update data database by calling the session objects methods. For details on the session object methods, see Action Script 3.0 Reference. Alternatively, you can see the standalone *Adobe ColdFusion ActionScript Language Reference,* which is accessible through the Documentation link on the Resources page of the ColdFusion Administrator.

Notes:

- The SQLite database doesn't validate column types when it creates a table. If you give it an invalid value for

© 2014 Adobe Systems Incorporated. All rights reserved.

column data type, it creates the column with that type.

- When you pass a unique integer ID parameter (one that is not used in the application) to the `OpenSession` method, the method creates an intermediate database file, which tracks the client changes to be committed on the server. If you use more than one database in a single application, use a unique ID for each database. Using a single ID ensures that you use the correct database for each client-side transaction.
- For asynchronous calls (such as SaveCache) that save fetched data in the local database, the call result can be available by using the session token when the call returns, before the responders are added. This situation occurs if the SaveCache operation tries to save null data. That is, if the fetch operation returned null data. In such cases, a responder is not required. There are two ways to handle this situation:

1. Check whether the result property of the session token returned by the function is null:

```
if (token.result == null) {
  Add the responder here
}
else {
  Directly use the token.result
 }
```

2. Check that the ArrayCollection that is input to the SaveCache function is not null. The null response indicates that the fetch operation did not get a result from the server:

```
If (ArrayCollection != null) {
  Call SaveCache() function
  Add Responder to the SaveCache Token
 }
 else {
  Handle the condition here.
 }
```

- If you call the `SaveUpdate` Method and a record with the specified primary key doesn't exist, the function inserts the record. The method updates an existing record only if the primary key exists in the client database.
- After you fetch data from the server, use only the `SaveCache` and `SaveCacheUpdate` methods to save the fetched data into client side database. If you use the Save function to store the fetched data, the data is marked for insert on server on commit, and the data you just got is written back to the server. In this case, a conflict occurs for the server database primary key. If the server-side logic handles this conflict by ignoring the primary key ID from the client, and lets the server generate a new ID, then the records are inserted, resulting in multiple copies of the data with different IDs.
- For AIR integration offline support, if you do not globally declare the variables for client side ActionScript classes but attempt to save the data records fetched from server using `session.saveCache()` or `session.saveUpdateCache()`, you might encounter AIR side error stackstrace with a message similar to the following:

```
"Error: The object of type Object does not have the Entity metadata tag at
coldfusion.air::EntityCache/addMetadata()[D:\p4\depot\ColdFusion\cf_main\tools
\AIRIntegration\OfflineSupport\src\coldfusion\air\EntityCache.as:228]"
```

Sending data to the server

© 2014 Adobe Systems Incorporated. All rights reserved.

The client SyncManager object tracks all the changes to the local data that happens through the session, so that the local data and server data can be synchronized. The SyncManager also tracks the old data instances when data that is already on the server is updated locally.
When the AIR application calls the `Session.commit` method, all changes that happened on the session are passed to the CFC sync function. The sync function determines if there are any conflicts between the returned information. If there are no conflicts, it updates the server data source. Otherwise, it handles the data as described in Conflict management.

Note

When you call the session.commit method, if the server does not throw an error, the commit method dispatches a CommitSucces event. This event indicates that the session.commit method has executed successfully, and the client data has been handed to the server CFC sync method. (If an error occurs while sending data to the sync method, the client receives a CommitFault event.) Therefore, the CommitSucces event does not mean that the server has saved the client data, but only that it has received the data. For example, the server does not save the data if there is a conflict, but the CommitSucces event is still dispatched. You handle conflicts separately by adding an event listener on the SyncManager. The event Flow is the first to get the CommitSucces event, and then a ConflictEvent event.

Setting Remote Credentials for SyncManager

To authenticate the AIR client that is connecting to the ColdFusion server, you can send the remoting credentials, which can be used on the server side under the `<cflogin>` tag. This is the same as setting the remoting credential for a normal flash remoting object.
The following code contains the getRemoteObject() method in SyncManager, which gets the underlying flash remoting object, so that you have full and same control over this as a normal flash remoting object.

```
syncmanager.getRemoteObject().SetRemoteCredentials("username","password");
```

© 2014 Adobe Systems Incorporated. All rights reserved.

Conflict management

Conflicts can happen in an offline application when the client modifies data that is already modified on the server. To identify such a conflict, the session.Commit method passes the following data to the ColdFusion server sync method:
`operations`: An array of operations to perform INSERT, UPDATE, or DELETE.
`clientobjects`: An array of new data changes.
`originalobjects`: An array of data that was in the client database before the change. There is no conflict in the following circumstances:

- If your are updating a record and the data on the server is same as the data in the originalobject. The client before the change had the same data as the server. In this case, the server updates its data source. If the old client data differs from that on the server, the application must handle the conflict.
- If you are inserting a new record. In this case, there is no originalobject value and ColdFusion can insert the record in the data store.
  You use the ColdFusion ObjectEquals function to identify conflicts. Pass the function the new instance of cfc from the client and the original instance to check if they are equal. If they are equal, the client has been working with the latest data. If it is not, the server can raise a conflict by returning the sever version of the instance present on the server from the sync method by creating an instance of `CFIDE.AIR.conflict.cfc`, setting its `serverobject` property (its only property) to the server value of the data, and returning the array of conflict objects to the AIR client.
  The following code is an ideal example of sync method that uses ORM methods for syncing operations and also handles conflicts.

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cffunction name="sync" returntype="any">
<cfargument name="operations" type="array" required="true">
<cfargument name="clientobjects" type="array" required="true">
<cfargument name="originalobjects" type="array" required="false">
<cfset conclits = ArrayNew(1)>
<cfset conflictcount = 1>

<cfloop index="i" from="1" to="#ArrayLen( operations )#">
 <cfset operation = operations[i]>
 <cfset clientobject = clientobjects[i]>
 <cfset originalobject = originalobjects[i]>
 <cfif operation eq "INSERT">
    <cfset obj = ORMGetSession().merge(clientobject)>
    <cfset EntitySave(obj)>
 <cfelseif listfindnocase("UPDATE,DELETE",operation) neq 0>
  <cfset serverobject = EntityLoadByPK("employee",originalobject.getId())>
  <cfif not isdefined('serverobject') >
   <cflog text="CONFLICT::SERVER OBJECT NOT FOUND, RECORD MAY BE DELETED ALREADY">
     <cfset conflict = CreateObject("component","CFIDE.AIR.conflict")>
     <cfset conflict.clientobject = clientobject>
     <cfset conflict.originalobject = originalobject>
     <cfset conflict.operation = operation>
     <cfset conflicts[conflictcount++] = conflict>
     <cfcontinue>
   </cfif>
   <cfset isNotConflict = ObjectEquals(originalobject, serverobject)>
   <cfif isNotConflict>
     <cfif operation eq "UPDATE">
        <cfset obj = ORMGetSession().merge(clientobject)>
        <cfset EntitySave(obj)>
     <cfelseif operation eq "DELETE">
        <cfset obj = ORMGetSession().merge(originalobject)>
        <cfset EntityDelete(obj)>
     </cfif>
   <cfelse><!----Conflict--->
     <cflog text = "is a conflict">
     <cfset conflict = CreateObject("component","CFIDE.AIR.conflict")>
     <cfset conflict.serverobject = serverobject>
     <cfset conflict.clientobject = clientobject>
     <cfset conflict.originalobject = originalobject>
     <cfset conflict.operation = operation>
     <cfset conflicts[conflictcount++] = conflict>
     <cfcontinue>
   </cfif>
 </cfif>
   </cfloop>
<cfif conflictcount gt 1>
<cfreturn conflicts>
</cfif>
</cffunction>
```

The CFC handling of the conflict depends on your application. In some cases, it can be appropriate to ignore the conflict and update the server data source with the new client data. In many cases, as in the preceding example, the

© 2014 Adobe Systems Incorporated. All rights reserved.

CFC informs the client about the conflict by returning the server value of the data.
On the client side, you use code such as the following to register the method that handles the conflict that the server returns.

```
syncmanager.addEventListener(ConflictEvent.CONFLICT, conflictHandler);
function conflictHandler(event:ConflictEvent):void
{
var conflicts:ArrayCollection = event.result as ArrayCollection;
var token:SessionToken = session.keepAllServerObjects(conflicts);
token.addResponder(new mx.rpc.Responder(conflictSuccess, conflictFault));
}
```

The conflictevent object contains an array of conflict objects that contain the clientinstance, originalinstance and the serverinstance. To accept the server's data, the application calls keepAllServerObjects, which takes an ArrayCollection that was passed to the conflict handler, or call the keepServerObject that takes an individual Conflict instance as shown in the following code. This conflict handler simply accepts any returned server object.

```
function conflictHandler(event:ConflictEvent):void
{
var conflicts:ArrayCollection = event.result as ArrayCollection;
var conflict:Conflict = conflicts.getItemAt(0);
var token:SessionToken = session.keepServerObject(conflict);
token.addResponder(new mx.rpc.Responder(conflictSuccess, conflictFault));
}
```

Conflicts can happen in the following cases:

- When the client does an update after the server data was updated. In this case, the client was using an old instance of data and not the latest data on the server. The server can inform the client by creating an instance of conflict.cfc in the sync method and setting the server instance on it. On the client side, you can call the keepServerObject function in the conflict handler to resolve the conflict by updating the client database with the server instance.
- When the client does an update but that record no longer exists on the server. Again, a conflict can be passed to the client from the server by creating an instance of Conflict.cfc and returning it. There is no need to set a serverobject property, as there is no server instance of the inserted data.
- When the client did an insert, but for example, the server data uses an autoincrement primary key field. The server, therefore, does not use the primary key inserted by the client. To inform the client of the correct key field value, the server returns the conflict cfc instance with the server instance. The ActionScript Calling keepServerObject method can then update the local data with the new primary key value from the server.

Note

After a commit or conflict resolution, it is recommended to synchronize the client database with the server data source, because the server can have new data available from other clients.

ActionScript has a few reserved keywords. When you name the Class/SQLite table, ensure that you do not use any of the reserved keywords. For example, `Order` is an ActionScript reserved keyword. If you name a table or class as `Order`, the table creation fails. To avoid this name conflict, use the `Table(name="OrderTable")` metadata tag to override the default name. Your code for the Order.as class could look something like the following:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
package test
{
[Entity]
[Table(name="OrderTable")]
public class Order
{
public function Order()
{
}
[Id]
public var oid:uint;
public var name:String;
[ManyToMany(targetEntity="test::Product",cascadeType='ALL')]
public var products:Array;
}
}
```

© 2014 Adobe Systems Incorporated. All rights reserved.

Offline AIR application example

The example here describes how to build an offline AIR application that has a one-to-one relationship between the Customer and Address objects in the database. You can use this example as a basis to build offline AIR applications for the other relationship types.

Client-side (AIR application) code

Create a folder called "onetoone" in your AIR project and add the ActionScript class files: Customer.as and Address.as with code that could be something like the following:

Customer.as

```
package onetoone
 {
  [Bindable]
  [RemoteClass(alias="AIRIntegration.custome r")]
  [Entity]

 public class  Customer
  {
  [Id]
   public var cid:int;
   public var name: String;
   [OneToOne(cascadeType='ALL',fetchType="EAGER")]
   [JoinColumn(name="add_id",referencedColumnName="aid")]

 public var address:Address;
 }
}
```

Address.as

```
package onetoone
{
  [Bindable]
  [RemoteClass(alias="AIRIntegration.address")]
  [Entity]

 public class Address
 {
  [Id]
   public var aid:int;
   public var street:String;
 }
}
```

MainApplication.mxml

Add code like the following in the MainApplication.mxml file to perform CRUD operations on the database.

© 2014 Adobe Systems Incorporated. All rights reserved.

Note

For Customer.as and Address.as ActionScript classes, global variables have been declared.

```
<?xml version="1.0" encoding="utf-8"?>
<mx:WindowedApplic ation xmlns:mx="http://www.adobe.com/2006/mxml"
layout="absolute" creationComplete="init()">
<mx:Script>

<![CDATA[
import mx.collections.ArrayCollection;
import mx.rpc.AsyncToken;
import mx.controls.Alert;

import coldfusion.air.events.*;
import coldfusion.air.*;

import onetoone.Address;
import onetoone.Customer;

private var session:Session;
private var dbFile:File;

private var cusColl:ArrayCollection;
private var syncmanager:SyncManager;
private var add:Address;//global variable for address.as
private var cus:Customer; //global variable for customer.as

private function init():void
 {
  // Provide Credentials for Server side Connection and CFC
  syncmanager = new SyncManager();
  syncmanager.cfPort = 80;
  syncmanager.cfServer = "localhost";
  // Path of the Server side CFC from CF webroot
  syncmanager.syncCFC = "AIRIntegration.cusManager";
  // This handler is called when any conflict occurs while
  writing back changes on the server side
  syncmanager.addEventListener(ConflictEvent.CONFLICT, conflictHandler);
  // Fetch Server side DB data onto Client SQLite DB while
  starting the App
  var token:AsyncToken= syncmanager.fetch("fetch");
  token.addResponder(new mx.rpc.Responder(fetchSuccess, fetchFault));
 }

private function conflictHandler(event:ConflictEvent):void
 {
  Alert.show("conflict man!");
  var conflicts:ArrayCollection = event.result as ArrayCollection;
  // Accept Server data and write it to client side SQLite DB
  var token:SessionToken
  = session.keepAllServerObjects(conflicts);
  token.addResponder(new
  mx.rpc.Responder(conflictSuccess, conflictFault));
 }
```

© 2014 Adobe Systems Incorporated. All rights reserved.

```
private function conflictSuccess(event:SessionResultEvent):void
 {
  Alert.show("conflict resolved");
 }

private function fetchSuccess(event:SyncResultEvent):void
 {
  var cus:Array = event.result as Array;
  cusColl = new ArrayCollection(cus);
  // Open a Session for the client side SQLite DB
  dbFile = File.userDirectory.resolvePath("onetoonesync.db");
  var sessiontoken:SessionToken
  =syncmanager.openSession(dbFile,017915);
  sessiontoken.addResponder(new
  mx.rpc.Responder(connectSuccess,connectFault));
 }

private function connectSuccess(event:SessionResultEvent):void
 {
  session = event.sessionToken.session;
  if(cusColl.length > 0)
  {
  // This operation saves it to the AIR SQLite DB
  var savetoken:SessionToken
  = session.saveCache(cusColl);
  savetoken.addResponder(new
  mx.rpc.Responder(saveCacheSuccess, savefault));
 }
 else
  {
   Alert.show("No data available from Server to save on local DB");
  }
 }

private function saveCacheSuccess(event:SessionResultEvent):void
 {
  Alert.show("Data saved on client Sqlite DB from Server");

  /*
  A new Insert is tried here. Note that this is not a complete user interface
  application. Otherwise, typically, users need to provide inputs to populate
  the Customer/Address Objects
  */
 var cus:Customer = new Customer();
 cus.cid=12;
 cus.name="New Customer";
 var add:Address = new Address();
 add.aid = 14;
 add.street = 'New Address';
 cus.address = add;

 /*
 INSERT the new Records, this is first saved in client side SQLite DB.
 On the Commit operation this new record is saved in the Server side DB
 Notice that although you are saving the Customer object here,
 this operation saves even the binded Address Object also,
 as both the entities are CASCADED inside Customer Class
 */
 var savetoken:SessionToken = session.save(cus);
```

© 2014 Adobe Systems Incorporated. All rights reserved.

```
 savetoken.addResponder(new mx.rpc.Responder(savesuccess, savefault));
 }

private function savesuccess(event:SessionResultEvent):void
 {
 Alert.show("Customer was Inserted Successfully");
 // Load some otehr Customer(ex: id=11) so that we can perform Update on
 that Customer
 var loadtoken:SessionToken =session.loadByPK(Customer,{cid:11});
 loadtoken.addResponder(new mx.rpc.Responder(loadCustomer,loadFault))
 }
private function loadCustomer(event:SessionResultEvent):void
 {
 var cus1:Customer = event.result as Customer;
 cus1.name = "UpdateCustomerName";
 var add1:Address = new Address;
 add1.aid = 22;
 add1.street = 'UpdatedCustomerAddress';
 cus1.address = add1;
 /*
 Let's call update now and save it to Client side SQLite DB
 */
 var savetoken:SessionToken = session.update(cus1);
 savetoken.addResponder(new mx.rpc.Responder(updateSuccess,updatefualt));
 }

private function updateSuccess(event:SessionResultEvent):void
 {
 Alert.show("Customer was updated Successfully");
 /*
 Let's Load another Customer(for example, with id 128) to perform a Delete operation
on that
 */
 var loadtoken:SessionToken = session.loadByPK(Customer,{cid:128});
 loadtoken.addResponder(new mx.rpc.Responder(loadCustomerForDelete,loadFault));
 }

private function loadCustomerForDelete(event:SessionResultEvent):void
 {
 // pass the loaded customer to remove function
 var removetoken:SessionToken = session.remove(event.result);
 removetoken.addResponder(new mx.rpc.Responder(removeSuccess,removeFault));
 }
private function removeSuccess(event:SessionResultEvent):void
 {
 Alert.show("Customer was deleted Successfully");
 }

private function commit():void
 {
 /*
 Until now, you have performed Insert/Update/Delete operation on Customer/Address
 entities on the client side SQLite DB. Now use the Commit opertaion to
 send them to the Server.
 */
 var committoken:SessionToken = session.commit();
 committoken.addResponder(new mx.rpc.Responder(commitSuccess, commitFault));
 }
```

© 2014 Adobe Systems Incorporated. All rights reserved.

```
private function commitSuccess(event:SessionResultEvent):void
 {
 Alert.show("Server has been updated with local changes");
 /*
 Now that you have completed all the operations, you can close the SQLite DB
 connection/session. It is a good practice to Close the session,
 after you complete all the operations.
 */
 var closetoken:SessionToken
 = session.close();
 closetoken.addResponder(new
 mx.rpc.Responder(sessionclosesuccess, sessionclosefault));
 }
private function sessionclosesuccess(event:SessionResultEvent):void
 {
 Alert.show("Session Close Success");
 }

 // Fault Handlers
 private function fetchFault(event:SyncFaultEvent):void
 {
 Alert.show("fetch fault" + event.toString());
 }
 private function conflictFault(event:SessionFaultEvent):void
 {
 Alert.show("conflict not resolved");
 }
 private function connectFault(event:SessionFaultEvent):void
 {
 Alert.show("connect failure" + event.toString());
 }
 private function sessionclosefault(event:SessionFaultEvent):void
 {
 Alert.show("Session Close Failed::"+event.error);
 }
 private function removeFault(event:SessionFaultEvent):void
 {
 Alert.show("Delete Operation Failed::"+event.error);
 }
 private function commitFault(event:SessionFaultEvent):void
 {
 Alert.show("Commit Failed::"+event.error);
 }
 private function loadFault(event:SessionFaultEvent):void
 {
 Alert.show("Load Failed::"+event.error);
 }
 private function updatefualt(event:SessionFaultEvent):void
 {
 Alert.show("update fault"+event.error);
 }
 private function savefault(event:SessionFaultEvent):void
 {
 Alert.show("Save Fault::"+event.error);
 }
]]>
</mx:Script>
<mx:Button click="commit()" name="commitbutton"
label="Commit/write local data to Server">
```

© 2014 Adobe Systems Incorporated. All rights reserved.

```
</mx:Button>
</mx:WindowedApplication>
```

Server-side code

Create the following cfc files - Application.cfc, Customer.cfc, Address.cfc, and Cusmanager.cfc with code like the following. The AIR client interacts with the Cusmanager.cfc file, in which you specify the code to fetch and sync the data back to the server.

Application.cfc

```
<cfcomponent>
  <cfset this.name = "OneTonOneExample">
  <cfset this.datasource="testorm">
  <cfset this.ormenabled="true">
  <cfset this.ormsettings={dialect = "MicrosoftSQLServer"}>
</cfcomponent>
```

Customer.cfc

```
<cfcomponent persistent="true">
  <cfproperty name="cid" fieldtype="id" >
  <cfproperty name="name" >
  <cfproperty name="address" fieldType='one-to-one'
  CFC="address" fkcolumn='aid' cascade='all' >
</cfcomponent>
```

Address.cfc

```
<cfcomponent persistent="true">
  <cfproperty name="aid" fieldtype="id" >
  <cfproperty name="street" >
</cfcomponent>
```

Cusmanager.cfc

```
<cfcomponent implements="CFIDE.AIR.ISyncManager">
<!----Fetch method--->
<cffunction name="fetch" returnType="Array" access="remote">
<cfset cus = ArrayNew(1)>
<cfset cus = EntityLoad("customer")>
<cfreturn cus>
</cffunction>
<!----SYNC method--->
<cffunction name="sync" returntype="any">
<cfargument name="operations" type="array" required="true">
<cfargument name="clientobjects" type="array" required="true">
<cfargument name="originalobjects" type="array" required="false">
```

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cfset conclits = ArrayNew(1)>
<cfset conflictcount = 1>

<cfloop index="i" from="1" to="#ArrayLen( operations )#">
 <cfset operation = operations[i]>
 <cfset clientobject = clientobjects[i]>
 <cfset originalobject = originalobjects[i]>
  <cfif operation eq "INSERT">
    <cfset obj = ORMGetSession().merge(clientobject)>
    <cfset EntitySave(obj)>
 <cfelseif listfindnocase("UPDATE,DELETE",operation) neq 0>
  <cfset serverobject = EntityLoadByPK("employee",originalobject.getcid())>
  <cfif not isdefined('serverobject') >
  <cflog text="CONFLICT::SERVER OBJECT NOT FOUND, RECORD MAY BE DELETED ALREADY">
     <cfset conflict = CreateObject("component","CFIDE.AIR.conflict")>
     <cfset conflict.clientobject = clientobject>
     <cfset conflict.originalobject = originalobject>
     <cfset conflict.operation = operation>
     <cfset conflicts[conflictcount++] = conflict>
     <cfcontinue>
   </cfif>

   <cfset isNotConflict = ObjectEquals(originalobject, serverobject)>
   <cfif isNotConflict>

     <cfif operation eq "UPDATE">
        <cfset obj = ORMGetSession().merge(clientobject)>
        <cfset EntitySave(obj)>
     <cfelseif operation eq "DELETE">
        <cfset obj = ORMGetSession().merge(originalobject)>
        <cfset EntityDelete(obj)>
     </cfif>

   <cfelse><!----Conflict--->
     <cflog text = "is a conflict">
     <cfset conflict = CreateObject("component","CFIDE.AIR.conflict")>
     <cfset conflict.serverobject = serverobject>
     <cfset conflict.clientobject = clientobject>
     <cfset conflict.originalobject = originalobject>
     <cfset conflict.operation = operation>
     <cfset conflicts[conflictcount++] = conflict>
     <cfcontinue>
   </cfif>
 </cfif>
  </cfloop>
<cfif conflictcount gt 1>
<cfreturn conflicts>
</cfif>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

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

© 2014 Adobe Systems Incorporated. All rights reserved.

Offline AIR application support in ColdFusion 9.0.1

The AIR integration feature introduced in ColdFusion 9 has an ActionScript ORM for persisting entities in the SQLite database present within Adobe Integrated Runtime (AIR). This release has the following enhancements for this ActionScript ORM:

- Support for auto-generating primary keys
- Support for encrypted database (introduced in AIR 1.5).
- Cache file used by ActionScript ORM to track the operations on SQLite database is now in the applicationStoragedirectory instead of applicationDirectory. You can specify the location of the cahceDirectory in openSession API on syncmanager.
- Supports Self Join relationships for one-to-one, one-to-many, and many-to-many database relationships.
- Supports both Array and ArrayCollection for use in ActionScript Entity to represent a collection in a database relationship.
- ActionScript ORM logs all the SQL statements that ORM uses to persist entities into the SQLite database.
- New APIs `keepClientObject` and `keepAllClientObjects` to ensure that the server updates are not retained when ColdFusion server raises conflict.
- The class `SessionToken` is dynamic and therefore, data can be stored on the token returned from the ORM APIs.
- Supports autocommit mode

Auto-generating primary keys

This release supports primary key generation for the ActionScript ORM using the metadata tag `GeneratedValue`.

GeneratedValue

Description

Adding this tag on an ActionScript primary key file auto-generates primary key.

Parameters

Parameter	Description
`strategy`	UUID uses the Flash UUID API to generate the ID (used for primary key of type `string`) or INCREMENT (for primary key of type `int`).
`initialValue`	Applies only for INCREMENT strategy. Specifies the initial value of the primary key. The default value is 0.
`incrementBy`	Applies only for INCREMENT strategy. Specifies the integer with which the value must be incremented to generate the primary key.

If the ID value is not present in the object, the value is generated and is assigned the primary key value. If the key value is already present on the object instance, then the key generation is ignored.For integer primary keys, the database table is checked for the presence of existing primary keys. If the highest key value is greater than the `initialValue`, then the key that is generated next will be an increment of the highest key value. For example, if the `initialValue` that you specify is `1`, and the database (already) has a key value `5`, then the next key is generated with the value `6` (5+1, if `incrementBy` is set to `1`).

© 2014 Adobe Systems Incorporated. All rights reserved.

Example

```
//Integer Primary Keys
===========
package test.apollo.CFSQLiteSupport.INCREMENTPK
{
 [Entity]
 [RemoteClass(alias="Customer")]
 public class Customer
 {
  public function Customer()
  {
  }
  [Id]
  [GeneratedValue(strategy="INCREMENT",initialValue=5,incrementBy=2) ]
        public var cid:int;
        public var name:String;

        [OneToOne(mappedBy="customer")]
        public var ord:Order;
 }
}
//String Primary Keys
===========
package test.apollo.CFSQLiteSupport.UUIDPK
{
 [Entity]
 [RemoteClass(alias="Customer")]
 public class Customer
 {
  public function Customer()
  {
  }
  [Id]
  [GeneratedValue(strategy="UUID") ]
        public var cid:String;
        public var name:String;

        [OneToOne(mappedBy="customer")]
        public var ord:Order;
 }
}
```

Encrypting the database

You can protect the database used by ActionScript ORM with an encryption key.
Use the `ByteArray` encryption key for `syncmanager.openSession` method to encrypt the database. The user-specified database file and the cache database file (used by the ActionScript ORM) are both encrypted using the encryption key you specify.
The key is optional.

Example

© 2014 Adobe Systems Incorporated. All rights reserved.

```
dbFile = File.userDirectory.resolvePath("customerManger.db");
    dbDir = File.applicationStorageDirectory;
 var keyGenerator:EncryptionKeyGenerator = new EncryptionKeyGenerator();
    var encryptionKey:ByteArray = keyGenerator.getEncryptionKey("UserPassword");

 var sessiontoken:SessionToken
=syncmanager.openSession(dbFile,179176,encryptionKey,dbDir);
```

For details on `EncryptionKeyGenerator,` see the section Using the EncryptionKeyGenerator class to obtain a secure encryption key in *Developing Adobe AIR 1.5 Applications with Flex.*

Specifying the cache directory

The cache directory where you store the cache file can be specified using the `cacheDirectory` (instance of `flash.filesystem.File`) for the `syncmanager.openSession` method.
The `cacheDirectory` is optional.

> **Note**
>
> By default, the cache file used by the ActionScript ORM is stored in the File.applicationStorageDirectory (in ColdFusion 9, it was stored in File.applicationDirectory).

For example, see *Encrypting the database* in this page.

Support for self joins

Database table can be related to itself through a foreign key. A typical example is an Employee table with a manager relationship containing the employee id of the managers (who manage the employee).
The manager id refers to another row in the same table. This is an example of one-to-one self join.
There can be one-to-many self join and many-to-many self joins with an intermediate join table.
ColdFusion 9 Update 1 has self join support for all the relationships in the ActionScript ORM.
The following ActionScript class definition for customer entity illustrates how all the self-join relationships are defined:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
package
{

 [Bindable]
 [RemoteClass(alias="AIRIntegration.customer")]
 [Entity]
 public class Customer
 {
  [Id]
  [GeneratedValue(strategy="INCREMENT",initialValue=5,incrementBy=2)]
        public var cid:int;
        public var name:String;

        [OneToOne(cascadeType='ALL',fetchType="EAGER")]
        [JoinColumn(name="add_id",referencedColumnName="aid")]
        public var address:Address;

        // Many-to-One self Join
        [ManyToOne(targetEntity="onetoone::Customer",fetchType="EAGER")]
  [JoinColumn(name="managerId",referencedColumnName="cid")]
  public var manager:Customer;

  // One-to-one Self Join
  [OneToOne(targetEntity="onetoone::Customer",fetchType="EAGER")]
  [JoinColumn(name="spouseId",referencedColumnName="cid",unique="true")]
  public var spouse:Customer;

       // Many-to-Many self Join
       [ManyToMany(targetEntity="onetoone::Customer",fetchType="EAGER")]
       [JoinTable(name="CUSTOMER_PARENTS_MAPPINGS")]
       [JoinColumn(name="CUST_ID",referencedColumnName="cid")]
       [InverseJoinColumn(name="PARENT_ID",referencedColumnName="cid")]
       public var parents:Array;


       // Many-to-Many self Join
       [ManyToMany(targetEntity="onetoone::Customer",fetchType="EAGER")]
       [JoinTable(name="CUSTOMER_CHILDREN_MAPPINGS")]
       [JoinColumn(name="CUST_ID",referencedColumnName="cid")]
       [InverseJoinColumn(name="CHILD_ID",referencedColumnName="cid")]
  public var children:Array;

[OneToMany(targetEntity="onetoone::Order",cascadeType='REMOVE',mappedBy="customer",f
etchType="EAGER")]
  public var orders:Array;

 }
}
```

ArrayCollection to hold multiple entities

In addition to Array, you can now use ArrayCollection to hold multiple entities in a database relationship. ArrayCollection can also be used in the ActionScript entities as Arrays are used to represent the related entities.

Example

© 2014 Adobe Systems Incorporated. All rights reserved.

```
package
{
 import mx.collections.ArrayCollection;

 [Bindable]
 [RemoteClass(alias="AIRIntegration.customer")]
 [Entity]
 public class Customer
 {
  [Id]
  [GeneratedValue(strategy="INCREMENT",initialValue=5,incrementBy=2)]
        public var cid:int;
        public var name:String;

        [OneToOne(cascadeType='ALL',fetchType="EAGER")]
        [JoinColumn(name="add_id",referencedColumnName="aid")]
        public var address:Address;

[OneToMany(targetEntity="onetoone::Order",cascadeType='REMOVE',mappedBy="customer",f
etchType="EAGER")]
  public var orders:ArrayCollection;

 }
}
```

Server-side configuration

See the section *Changes in the XML configuration files for Flash Remoting in ColdFusion 9 and ColdFusion 9.0.1*.

Logging SQL statements

The ActionScript ORM logs all SQL statements that it executes.
The log can be configured as follows:

1. Add a log target for the AIR applications as shown in the following example:

```
var logTarget:TraceTarget = new TraceTarget();
logTarget.filters = "*";
logTarget.level = LogEventLevel.ALL;
Log.addTarget(logTarget);
```

The log target is the TraceTarget where all the trace statements appear. The log target can be set to any other log using the Flash APIs.

ColdFusion ActionScript APIs

The following two APIs have been introduced to the session class in the `coldfusion.air` package:

keepAllClientObjects

Description

© 2014 Adobe Systems Incorporated. All rights reserved.

Takes an `ArrayCollection` of conflict instances and keeps the client object for every conflict instance in the ArrayCollection.

Returns

An instance of `coldfusion.air.SessionToken` (which is the token for `keepAllClientObjects` call).

Syntax

```
public function keepAllClientObjects(conflicts:ArrayCollection):SessionToken
```

Parameters

Parameter	Description
`mx.collections.ArrayCollection`	An `ArrayCollection` of conflicts raised by the server.

Example

```
private function conflictHandler(event:ConflictEvent):void
{
   // Alert.show("Server returned a Conflict !");
    var conflicts:ArrayCollection = event.result as ArrayCollection;
    // Ignore Server data and retain client Data in  SQLite DB
    var token:SessionToken = session.keepAllClientObjects(conflicts);
    token.addResponder(new mx.rpc.Responder(conflictSuccess, conflictFault));
}
```

keepClientObject

Description

Ensures that the client object is retained instead of the one from the server (despite server raising data conflict).The API also ensures that the retained client object is not sent to the server as a new operation on sync.

Returns

An instance of `coldfusion.air.SessionToken` associated with `keepClientObject` call.

Syntax

```
public function keepClientObject(conflict:coldfusion.air.Conflict):SessionToken
```

Parameters

Parameter	Description
`coldfusion.air.Conflict`	The conflict that the server raises.

© 2014 Adobe Systems Incorporated. All rights reserved.

Example

See the example in the section *keepAllClientObjects*. For `keepClientObject`, the only difference is that you must iterate over each conflict in the conflictarray collection.

Offline AIR SQLite API enhancements

The following new parameters for `openSession`:

New Parameters	Type	Required/Optional	Description
`encryptionKey`	`ByteArray`	Optional	Used to encrypt offline SQLite database. For details, see *Encrypting the database* in this page.
`cacehDirectory`	`File`	Optional	Used to specify a custom cache directory. For details, see *Specifying the cache directory* in this page.

SessionToken class is dynamic

A class is dynamic in ActionScript if you can add additional key-value pairs to the instance of the class.
In this release, `sessionToken` is dynamic class. Therefore, you can add additional information that can be passed from where the API is called to the success or fault handlers.
Example

```
private function fetchData():void
{
var token:AsyncToken= syncmanager.fetch("fetch");
token.addResponder(new mx.rpc.Responder(fetchSuccess, fetchFault));

                // Test For SessionToken class to be Dynamic, so that Dynamic
Properties could be added
                token.userdefined_key = "value";
   }
        public function fetchSuccess(event:SyncResultEvent):void
            {

              if(event.token.userdefined_key == "value")
                { .... }
             }
```

Support for AutoCommit

SyncManager supports a Boolean property `autoCommit`.
The default value is `false`.
If `true`, the changes in the local database are committed to the server when the `save`, `saveUpdate`, and `remove` methods are used as shown here:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
private var syncmanager:SyncManager = new SyncManager();
syncmanager.autoCommit = true;
```

This functionality helps you minimize the conflicts during the synchronization with the server, particularly in the case of auto-generation of primary key on client and serverside.

New attribute for SessionResultEvent and SessionToken

The classes `SessionResultEvent` and `SessionToken` have a new attribute `autoGeneratedId` that gets populated with the auto-generated ID used by ActionScript ORM. `autoGeneratedId` is populated only when a key is generated by the ActionScript ORM in that specific call.

Example

```
private function connectSuccess(event:SessionResultEvent):void
{
  // Generate an Order Obect
   .
   .
   .
   .
  // Save the Order
  var savetoken:SessionToken = session.save(ord);
  savetoken.addResponder(new mx.rpc.Responder(savesuccess, savefault));
}
private function savesuccess(event:SessionResultEvent):void
{
   // This is how, you can access autogenrated PK
   RememberINTPK = event.autoGeneratedId.toString();
   var loadtoken:SessionToken = session.loadByPK(Order,{oid:RememberINTPK},true);

   loadtoken.addResponder(new mx.rpc.Responder(loadsuccess,loadfailure))
}
```

Note

Assume that the server database generates primary keys and you choose to generate primary key on client SQL Lite table (as shown in the example). This scenario results in a conflict which the application developer must resolve. An option is to design your application in such a way that you minimize conflicts between client and server objects. In this case, you can set client object primary keys as null or empty string before saving data to the database server using serverside ORM EntitySave function.

#back to top

© 2014 Adobe Systems Incorporated. All rights reserved.

Proxy ActionScript Classes for ColdFusion Services

Flex-based applications in AIR and Flash can access several ColdFusion services by using ColdFusion proxy ActionScript classes. This feature is available in all Flex-based applications that run on Flash and AIR. ColdFusion provides services corresponding to the following tags and their child subtags: `cfchart`, `cfdocument`, `cfimage`, `cfmail`, `cfpdf`, `cfpop`. Using ColdFusion you can also upload files from the application to the server.
ColdFusion provides the following Flex proxy classes and related support classes:

- Config (configures the application for using ColdFusion services)
- Util (includes file upload support)
- Chart (cfchart)
- Document (cfdocument)
- Image (cfimage)
- Mail (cfmail)
- PDF (cfpdf)
- Pop (cfpop)
  These classes are part of coldfusion.service.mxml package, distributed in the cfservices.swc file. You normally use these classes in MXML tag format, using the cf namespace identifier, as in the following line:

```
{{<cf:Image id="image" action="AddBorder" source="Uploaded Image server URL"
thickness="5" color="Blue"/>}}
```

To use a ColdFusion service in an application built with Flex, you use the Config class to establish the connection, and then use the other classes to access the ColdFusion services.
Since ColdFusion 9, you can also specify the remoting destination in the Config class as well as all the proxy tags.

Note

To use ColdFusion services from Flex and AIR, you must enable access to the services as described in "Enable ColdFusion Services" in the ColdFusion Web Services section.

© 2014 Adobe Systems Incorporated. All rights reserved.

About the cfservices.swc file

To use any ColdFusion service class in an application built with Flex/AIR, do the following:

1. Include the cfservices.swc file, located at /CFIDE/scripts/AIR/cfservices.swc, in your application
2. In Flash Builder, add the cfservices.swc file to your project by right-clicking Flex/AIR Project > Properties >Flex BuildPath >Library Path >Add SWC.
   For details on all ColdFusion service classes, and other classes in the coldfusion.air package, see [ActionScript 3.0 Reference]. Alternatively, you can see the standalone *Adobe ColdFusion ActionScript Language Reference,* which is accessible through the Documentation link on the Resources page of the ColdFusion Administrator.
   For information about attributes for specific services, such as Mail or Pop, see the attributes of corresponding ColdFusion tags and functions.

© 2014 Adobe Systems Incorporated. All rights reserved.

Using the Chart class

The Chart class is the proxy for the ColdFusion Chart service, which provides the functionality of the `cfchart` tag and its child `chartdata` and `chartseries` tags.

- You specify the `cfchart` attributes as Chart object properties.
- You represent chart series in the `chartSeries` element of the chart object. The `chartSeries` element is an arrays of objects, each of which represents a single chart (`chartseries` tag) document section. These objects include a type element for the chart type, a chartdata element for the chart data, and elements for any other series attributes.
- You represent each chart's data as an array of objects, each of which contains an item element and a value element. You use these arrays as the `chartdata` elements of the chartSeries object.
- You call the document object execute() function to run the service.
  The following example shows how you can use the chart service:

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
layout="absolute" xmlns:cf="coldfusion.service.mxml.*" creationComplete="init()">
<mx:Script>
 <![CDATA[
  import mx.controls.Alert;
  import mx.rpc.events.FaultEvent;
  import mx.rpc.events.ResultEvent;

 [Bindable]
 public var chaSer:Array;
 public var chaDat:Array;
  function init():void
  {
  chaDat =
  [{item:"Facilities",value:"35000"},
  {item:"Facilities1",value:"32000"}];
  chaSer = [{type:"bar",chartdata:chaDat},
  {type:"line",chartdata:chaDat}];chartest.execute();
  }
   function handleResult(event:ResultEvent):void
   {
   Alert.show("success" + event.result.toString());
   }
   function handleFault(event:FaultEvent):void
   {
   Alert.show("failure" + event.toString());
   }
 ]]>
</mx:Script>
 <cf:Config id="configid" cfServer="localhost"
 cfPort="80" servicePassword="service"
 serviceUserName="service" >
 </cf:Config>
<cf:Chart id="chartest"
  action="generate"
  format="jpg"
  xAxisTitle="Department"
  yAxisTitle="Salary Average"
  chartSeries="{chaSer}"
  result="handleResult(event)"
```

© 2014 Adobe Systems Incorporated. All rights reserved.

```
fault="handleFault(event)"
backgroundColor = "Black"
chartHeight = "500"
chartWidth = "600"
dataBackgroundColor = "yellow"
font = "ariel"
fontBold = "yes"
fontItalic = "yes"
fontSize = "12"
foregroundColor = "red"
gridLines = "2"
labelFormat = "number"
markerSize = "10"
showBorder = "yes"
showLegend = "yes"
showMarkers = "yes"
showxGridLines="yes"
showyGridLines="yes"
tipBgColor="blue"
tipStyle = "MouseOver"
```

© 2014 Adobe Systems Incorporated. All rights reserved.

```
    title = "AIR Integration testing"/>
  </mx:Application>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

Using the Config class

Use the Config class to set the configuration parameters for ColdFusion Services, including the connection details and event handlers. Therefore, use the Config class before using any of the service classes. The parameters set on the Config class are global and can be overridden by the individual service proxy classes.
The following table lists the Config class parameters, normally used as attributes of a Config tag.

Attribute	Description
serviceUserName	The userName set in the ColdFusion Administrator with the permission to access the specific service being requested.
servicePassword	The password set in the ColdFusion Administrator for the user name.
cfServer	The server name or IP address of the CF server.
cfPort	The port on which the CF Server is running.
cfContextRoot	The context root if any for the CF server.
secureHTTP	Boolean value specifying whether to use http or https to run the service.
destination	The destination attribute can be used to specify a user defined remoting destination in WEB-INF/flex/remoting-config.xml. If not specified, default ColdFusion destination is used.

You normally specify the config class using an MXML tag as follows:

```
{{<cf:Config id="conf" cfServer="CF Server IP Address/HostName" CFPort="HTTP Port on
Which CF accepts request" destination="UserDefinedRemotingDestination" >}}
```

You can also override the server settings directly in the service tag, for example:

```
<cf:Image id="image" action="AddBorder" cfServer="IP Address" cfPort="Port number"
source="Uploaded Image server URL" thickness="5" color="Blue"
destination="UserDefinedRemotingDestination" />
```

© 2014 Adobe Systems Incorporated. All rights reserved.

Using the Document class

The Document class is the proxy for the ColdFusion Document service, which provides the functionality of the `cfdocument` tag and its child `cfdocumentsection` and `cfdocumentitem` tags.

- You specify the `cfdocument` attributes as Document object properties. You specify document content that is not in a section as a content element of the document object.
- You represent document sections in the documentSection element of the document object. The `documentSection` element is an arrays of objects, each of which represents a single document section. These objects include a content element for the section content, an optional `documentItem` element for the document items, and elements for any other section attributes.
- You represent all document items in a document section (or the document object) as an array of objects with type and content elements. The array element type field specifies whether the item is a header, footer, or page break. You specify the document item array as a `documentItem` element of the document object or a `documentSection` object.
- You call the `document object execute()` function to run the service.
  The following excerpt from the full example shows how to create sections and items and add them to a document:

```
[Bindable]
var docItem:Array = [{type:"header",content:"<font size='-3'>
<i>Salary Report</i></font>"},{type:"footer",
content:"<font size='-3'>
Page #cfdocument.currentpagenumber#</font>"}];

[Bindable]var docSectionItem:Array = [{content:"<table width='95%'
border='2' cellspacing='2' cellpadding='2' >
<tr><th>Salary</th></tr><tr>
<td><font size='-1'>John</font></td>
<td align='right'><font size='-1'>Guess What</font></td></tr>
<tr><td align='right'><font size='-1'>Total</font></td>
<td align='right'><font size='-1'>Peanuts</font></td></tr>",
documentitem:docItem},{content:"content2",documentitem:docItem}];
.
.
.
cfDoc.documentSection = docSectionItem;
```

The following example shows some typical document use:

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
layout="vertical" xmlns:cf="coldfusion.service.mxml.*"
creationComplete="init()">
<mx:Script>
 <![CDATA[
 import mx.controls.Alert;
 import mx.rpc.events.ResultEvent;
 import coldfusion.service.PdfParam;

 [Bindable]
 var docItem:Array = [{type:"header",content:"
 <font size='-3'><i>Salary Report</i></font>"},
```

© 2014 Adobe Systems Incorporated. All rights reserved.

```
  {type:"footer",content:"<font size='-3'>
  Page <cfoutput>#cfdocument.currentpagenumber#
  </cfoutput></font>"}];

  [Bindable]
  var docSection:Array =
  [{content:"content1"},{content:"content2"},
  {content:"content3"}];

  [Bindable]
  var docSectionItem:Array =
  [{content:"content1",documentitem:docItem},
  {content:"content2",documentitem:docItem},
  {content:"content3",documentitem:docItem}];
  [Bindable]
  var res:String = new String();
  private function init():void
   {
    doctestnow.execute();
   }
  private function handleResult(event:ResultEvent):void
   {
    res=event.result.toString();
    //Alert.show("httpurl= "+event.result.toString());
   }
   private function handleError(event:Event):void
   {
    mx.controls.Alert.show(event.toString());
   }
   ]]>
</mx:Script>
<cf:Config id="configid" cfServer="localhost"
cfPort="80" servicePassword="service" serviceUserName="service" />

  <!-- simple case-->
  <cf:Document id="doctestnow" action="generate"
  format="flashpaper" result="handleResult(event)"
  fault="handleError(event)"
  content="&lt;table&gt;&lt;tr&gt;&lt;td&gt;bird&lt;/td&gt;&lt;td&gt;
  1&lt;/td&gt;&lt;/tr&gt;&lt;tr&gt;&lt;td&gt;fruit&lt;/td&gt;&lt;
  td&gt;2&lt;/td&gt;&lt;/tr&gt;&lt;tr&gt;&lt;td&gt;rose&lt;/td&gt;
  &lt;td&gt;3&lt;/td&gt;&lt;/tr&gt;&lt;/table&gt;"/>
  <!--doc item case -->
  <!--<cf:Document id="doctestnow" action="generate"
  format="flashpaper" result="handleResult(event)"
  fault="handleError(event)" documentItem="{docItem}"/>-->
  <!-- doc section case-->
  <!--<cf:Document id="doctestnow" action="generate"
  format="flashpaper" result="handleResult(event)"
  fault="handleError(event)" documentSection="{docSection}"/>-->

  <!-- doc section and doc item case
  <cf:Document id="doctestnow" action="generate"
  format="flashpaper" result="handleResult(event)"
  fault="handleError(event)" documentSection="{docSectionItem}" />-->
```

© 2014 Adobe Systems Incorporated. All rights reserved.

```
    <mx:SWFLoader source="{res}"/>
  </mx:Application>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

Using the Image class

The Image class is the proxy for the ColdFusion Image service, which provides the functionality of the cfimage tag. You specify the required cfimage attributes for the action as Image tag attributes. The following examples show typical usage, in this case adding a border:

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" layout="vertical"
xmlns:cf="coldfusion.service.mxml.*"  creationComplete="init()">
<mx:Script>
<![CDATA[
import mx.controls.Alert;
import coldfusion.service.events.*;

function init():void
 {
  img.execute();
 }

function handleResult(event:ColdFusionServiceResultEvent):void
 {
  mx.controls.Alert.show("result=" + event.result.toString());
  retImage.source = event.result.toString();
 }

function handleError(event:ColdFusionServiceFaultEvent):void
 {
  mx.controls.Alert.show(event.toString());
 }
]]>
</mx:Script>

<cf:Config
 id="configid"
 cfServer="127.0.0.1"
 cfPort="80"
 servicePassword="service"
 serviceUserName="service"/>

<!-- Add border-->
<cf:Image
 id="img"
 action="addborder"
 color="red"
 thickness="5"
 source="http://127.0.0.1:80/GetExifmetaData.jpg"
 result="handleResult(event)"
 fault="handleError(event)"/>

<mx:Image id="retImage"/>
</mx:Application>
```

Batch operation

A batch operation lets you perform multiple image manipulations on an image as part of one operation, instead of performing the operations individually. You upload an image to the server, use the batch operation to perform

© 2014 Adobe Systems Incorporated. All rights reserved.

multiple Image operations on it, and get the modified image back from the server. To use this action, use an associative array of actions and corresponding attributes. It is easier to create an associative array in ActionScript than MXML.

The following example shows code to create the associative array with the action information and perform the batch operation:

ActionScript Part:

```
[Bindable]
public var attributes:Array =
[{AddBorder:{color:"Red",thickness:"50"}},
{Resize:{width:"50%",height:"50%",interpolation:"blackman",blurfactor:"2"}},
{Flip:{transpose:"270"}}]
```

MXML Part:

```
<!-- batch operation --
<cf:Image id="img" action="batchoperation"
source="http://localhost:8500/cat.jpg"
attributes="{attributes}"/>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

Using the Mail class

The Mail class is the proxy for the ColdFusion Mail service, which provides the functionality of the cfmail tag. You specify the required cfmail and child tag attributes for the action as Mail tag attributes. The default Mail action on this tag is `send`.

The following AIR application uses the Mail Service and file upload functionality. It refers to the CFCredentials.mxml file to reference the credentials of the user for authentication.

Following is the CFCredentials.mxml file being used for the example:

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Panel xmlns:mx="http://www.adobe.com/2006/mxml" currentState="collapsed"
toolTip="Double Click to Collapse/Expand" creationComplete="retrieveCredential()"
headerHeight="5" layout="vertical" mouseDown="check Collapse(event )"
resizeEffect="Resize">
<mx:Script>
 <![CDATA[
 import mx.controls.Alert;
 import mx.rpc.http.HTTPService
 import mx.rpc.events.Result Event;
 import mx.rpc.events.FaultEvent;
 import flash.data.EncryptedLocalStore;
 import flash.utils.ByteArray;
 public var service:HTTPService = new HTTPService();
    private function testConnection () :void
  {
      var CF_HTTP URL:String;
      if(cfip.text == null)
      {
        Alert.show("IP Not provided or Invalid IP Address");
      }
      else if(cfprt.text!= "" && cfcnxtrt.text != "")
      {CF_HTTPURL="http://"+cfip.text+":"+cfprt. text+"/"
+cfcnxtrt.text+"/flex2gateway/";
      }
         else if(cf prt.text == "" && cfcnxtrt.text == "")
      {
         CF_HTTPURL ="http://"+cfip.text+"/flex2gateway/";
      }
      else if(cf prt.text!= "" && cfcnxtrt.text == "")
      {
         CF_HTTPURL="http://"+cf ip.text+":"+cfprt.text+"/flex2gateway/";
      }

         if(cfserviceusername.text == "" && cfservicepassword.text == "")
          {
           Alert.show("CF Service UserName and Password are not required to test
    CF server connectivity but they will be required while using the CF
       services","Note");

          }

          service.url = CF_HTTPURL;
          service.method = "POST";
          service.addEventListener(ResultEvent.RESULT,httpResult);
          service.addEventListener(FaultEvent.FAULT,httpFault);
             service.send();
```

© 2014 Adobe Systems Incorporated. All rights reserved.

```
        }

    public function httpResult(event:ResultEvent):void
   {
   Alert.show("Connection with ColdFusion server Successful","Connection Status");
   }
    public function httpFault(event:FaultEvent):void
   {
   Alert.show("ColdFusion server could not be reached, Make sure credentials are
   correct and CF server is running","Error");
    }
    private function checkCollapse(event:MouseEvent):void
   {
   if( event.clickCount == 2)
    {
     currentState = currentState == "collapsed" ? "":"collapsed";
    }
   }
   private function rememberCredential():void
   {
     var data:ByteArray = new ByteArray();
   data.writeUTFBytes(cfip.text);
   En cryptedLocalStore.setItem('IP', data );
     var data:ByteArray = new ByteArray();
   data.writeUTFBytes(cfprt.text);
   Encr yptedLocalStore.setItem('PORT', data );
     var data:ByteArray = new ByteArray();
   data.writeUTFBytes(cfcnxtrt.text);
   Encrypt edLocalStore.setItem('CONTEXT', data );
   var data:ByteArray = new ByteArray();
   data.writeUTFBytes(cfserviceusername.text);
   Encr yptedLocalStore.setItem( 'USER', data );
   var data:ByteArray = new ByteArray();
   data.writeUTFBytes(cfservicepassword.text);
   Encrypted LocalStore.setItem('PASS', data );
   }
   private function retrieveCredential():void
   {
   try{
   cfip.text = EncryptedLocalStore.getItem('IP').toString();
   cfprt.text = EncryptedLocalStore.getItem('PORT').toString();
   cfcnxtrt.text = EncryptedLocalStore.getItem('CONTEXT').toString();
   cfserviceusername.text = EncryptedLocalStore.getItem('USER').toString();
   cfservicepassword.text = EncryptedLocalStore.getItem(' PASS').to String();
   }
   catch(e:Error)
   {
   }
 }
        private function resetCredential():void
         {
    EncryptedLocalStore.reset();
    cfip.text = "";
    cfprt.text = "";
    cfcnxtrt.text  = "";
    cfserviceusername.text  = ""  ;
    cfservicepassword.text = "";
        }
```

© 2014 Adobe Systems Incorporated. All rights reserved.

```
 ]]>
</mx:Script>
<mx:ControlBar>
 <mx:Label text="CFServer IP"/>
 <mx: TextInput id="cfip" text="" width="   70"/>
 <mx:Label text="CFServer Port"/>
 <mx:TextInput id="cfprt" text="" width="40"/>
 <mx:Labeltext="CFServer Context Root (if any)"/>
 <mx:TextInput id="cfcnxtrt" text="" width="70"/>
 <mx:Label text="CFService UserName"/>
 <mx:TextInput id= "cfserviceusername" text="" width="70"/>
 <mx:Label text="CFService Password"/>
 <mx:TextInput displayAsPassword="true" id="cfservicepassword" text="" width="70"/>

 <mx:Button id="testconn" label="Test Connection" click="testConnection()"/>
 <mx:Button id="save" label=" Remember"click="rememberCredential()"/>
 <mx:Button id="reset" label ="Reset" click = "resetCred e n tial()"/>
</mx:ControlBar>
<mx:states>
 <mx:State name="collapsed">
  <mx: SetProperty  name="height" value="10"/>
 </mx:State>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

```
</mx:states>
</mx:Panel>
```

The AIR application example is as follows:

```
<?xml version="1.0" encoding="utf-8"?>
<mx:WindowedApplication xmlns:mx="http://www.adobe.com/2006/mxml" layout="absolute"
xmlns:local="com.*" creationComplete="init()" xmlns:cf="coldfusion.service.mxml.*">
<mx:Script>
<![CDATA[
import coldfusion.service.events.*;
import com.CFCredential;
import mx.collections.ArrayCollection;
import mx.binding.utils.BindingUtils;
import mx.rpc.events.FaultEvent;
import mx.rpc.events.ResultEvent;
import mx.controls.Alert;
import flash.events.*;
import flash.net.FileReference;
import flash.net.FileReferenceList;
import flash.net.URLRequest;
import flash.net.URLVariables;
import coldfusion.service.Util;
 public var fileTypes:Array = new Array();
 public var imageTypes:FileFilter = new FileFilter("Images (*.jpg; *.jpeg; *.gif;
*.png)" ,"*.jpg; *.jpeg; *.gif; *.png");
 public var documentTypes:FileFilter = new FileFilter("Documents (*.pdf), (*.doc),
(*.rtf), (*.txt)",("*.pdf; *.doc; *.rtf, *.txt"));
 [Bindable]
  private var fileslist:ArrayCollection;
  private var filereflist:FileReferenceList
  public var fileRef:FileReference = new FileReference();
 [Bindable]
 public var mailPartArray:Array = [{type:"text",content:"Plain text only"},
{type:"html",content:"<B>bold text man!!</B>"}];
 public var uploadURL:URLRequest = new URLRequest();
 [Bindable]
 public var attachCollection:Array = new Array();
 public var urlcnt:int=0;
 public function init():void
 {
 fileslist = new ArrayCollection();
 filereflist = new FileReferenceList;
 fileRef = new FileReference;
 uploadURL.url =
"http://"+conf.cfServer+":"+conf.cfPort+"/"+conf.contextRoot+"/"+Util.UPLOAD_URL;
 var variables:URLVariables = new URLVariables();
 variables.serviceusername = conf.serviceUserName;
 variables.servicepassword = conf.servicePassword;
 uploadURL.data = variables;
 uploadURL.method = "POST"; //this can also be set to "POST" depending on your needs

   uploadURL.contentType = "multipart/form-data";
   fileTypes.push(imageTypes);
   fileTypes.push(documentTypes);
  //Add Event Listeners to UI
```

© 2014 Adobe Systems Incorporated. All rights reserved.

```
  attachbutton.addEventListener(MouseEvent.CLICK, browseFiles);
  sendbutton.addEventListener(MouseEvent.CLICK,uploadFiles);
  filereflist.addEventListener(Event.SELECT, selectHandler);
  //mailtest.send();
  }

//Browse for files
 private function browseFiles(event:Event):void
  {
   filereflist.browse(fileTypes);
  }

//  called after user selects files form the browse dialogue box.
 private function selectHandler(event:Event):void
  {var i:int;
   for (i=0;i < event.currentTarget.fileList.length; i ++)
   {
    fileslist.addItem(event.currentTarget.fileList[i]);
    attachList.text +=  event.currentTarget.fileList[i].name + ",  ";
   }
  }
 private function uploadFiles(event:Event):void
 {
  if (fileslist.length > 0)
  {
   fileRef = FileReference(fileslist.getItemAt(0));
   fileRef.addEventListener(Event.COMPLETE, completeHandler);
   fileRef.addEventListener(DataEvent.UPLOAD_COMPLETE_DATA,dataHandler);
   fileRef.upload(uploadURL);
  }
  else if (fileslist.length == 0)
   {
    sendmail();
   }
  }

// called after a file has been successfully uploaded | We use this as well to check
if there are any files left to upload and how to handle it
 private function completeHandler(event:Event):void
  {
   // Alert.show("File Uploaded successfully");
   fileslist.removeItemAt(0);
   if (fileslist.length > 0)
   {
    uploadFiles(null);
   }
  }
//called after file upload is done and Data has been returned from Server
 private function dataHandler(event:DataEvent):void
 {
  attachCollection[urlcnt++] = {"file":
  Util.extractURLFromUploadResponse(event.data.toString())};
  if (fileslist.length == 0)
   sendmail();
 }
 private function sendmail():void
 {
  mailtest.execute();
 }
```

© 2014 Adobe Systems Incorporated. All rights reserved.

```
  public function handleResult(event:ColdFusionServiceResultEvent):void
  {
   from.text="";too.text=""; cc.text="";bcc.text="";
   subject.text="";mailbody.text="";
   attachList.text=""; fileslist.removeAll();
   Alert.show("Mail was delivered Successfully");
  }
  public function handleError(event:ColdFusionServiceFaultEvent):void
  {
   Alert.show("Failure"+ event.toString());
  }
  ]]>
 </mx:Script>
 <mx:Panel width="100%" height="100%">
  <local:CFCredential id="cred" />
  <mx:ControlBar>
  <mx:Spacer width="100%"/>
  <mx:HBox>
   <mx:Button label="Send Mail" id="sendbutton"/>
   <mx:Button label="Attachment" id="attachbutton"/>
  </mx:HBox>
  </mx:ControlBar>
 <mx:VBox width="100%" height="100%">
  <mx:HBox width="100%">
  <mx:Text text="From" width="100" />
  <mx:TextInput width="100%" id="from"/>
  </mx:HBox>
  <mx:HBox width="100%">
  <mx:Text text="To" width="100"/>
  <mx:TextInput width="100%" id="too"/>
  </mx:HBox>
  <mx:HBox width="100%">
  <mx:Text text="CC" width="100"/>
  <mx:TextInput width="100%" id="cc"/>
  </mx:HBox>
  <mx:HBox width="100%">
  <mx:Text text="Bcc" width="100"/>
  <mx:TextInput width="100%" id="bcc"/>
  </mx:HBox >
  <mx:HBox width="100%">
  <mx:Text text="Subject" width="100"/>
  <mx:TextInput width="100%" id="subject"/>
  </mx:HBox>
  <mx:HBox width="100%">
  <mx:Text text="Attachments" width="100"/>
  <mx:TextInput width="100%" id="attachList"  enabled="false"
  backgroundDisabledColor="white"/>
  </mx:HBox>
  <mx:TextArea width="100%" height="100%" id="mailbody"/>
 </mx:VBox>
 </mx:Panel>

  <!--Provide your CF server credentials here-->
   <cf:Config id="conf"
   cfServer="{cred.cfip.text}"
   cfPort="{int(cred.cfprt.text)}"
   cfContextRoot="{cred.cfcnxtrt.text}"
   servicePassword="{cred.cfservicepassword.text}"
   serviceUserName="{cred.cfserviceusername.text}"
```

© 2014 Adobe Systems Incorporated. All rights reserved.

```
/>

<cf:Mail id="mailtest"
 server="xx.xxx.xx.xxx"
 to="{too.text}" bcc="{bcc.text}" cc="{cc.text}"
 failTo="jayesh@adobe.com" replyTo="jayesh@adobe.com"
 subject="{subject.text}" content="{mailbody.text}"
 from="{from.text}"
 attachments="{attachCollection}"
 type="text" charset="utf-8" mailerId="CF" priority="1"
 timeout="60" useTLS="true" wrapText="5"
 result="handleResult(event)"
 fault="handleError(event)"
```

© 2014 Adobe Systems Incorporated. All rights reserved.

```
  />
</mx:WindowedApplication>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

Using the PDF class

The `Pdf` class is the proxy for the ColdFusion PDF service, which provides the functionality of the cfpdf tag. You specify the required cfpdf attributes for the action as `Pdf` tag attributes. The following examples show each supported action:

```
<!-- Get Info-->
<cf:Pdf id="pdftest" action="GETINFO"
source="http://localhost:8500/lcds26_devguide_040908.pdf"/>

<!-- Delete Pages-->
<cf:Pdf id="pdftest1" action="deletepages"
source="http://localhost:8500/lcds26_devguide_040908.pdf"
pages="1" resultHandler="handleDeleteResult"
errorHandler="handleDeleteError"/>

<!-- Merge files--><cf:Pdf id="pdftest" action="mergeFiles"
source="http://localhost:8500/lcds26_devguide_040908.pdf,
http://localhost:8500/EC205W_JoelGeraci.pdf"
resultHandler="handleMergeResult"
errorHandler="handleMergeError"/>

<!-- extract pages-->
<cf:Pdf id="pdftest" action="extractpages"
source="http://localhost:8500/lcds26_devguide_040908.pdf"
pages="2" keepBookmark="true"
resultHandler="handleExtractResult"
errorHandler="handleExtractError"/>

<!-- addwatermark-->
<cf:Pdf id="pdftest" action="addwatermark"
source="http://localhost:8500/1page.pdf"
image="http://localhost:8500/IMG_8680.JPG"
resultHandler="handleExtractResult"
errorHandler="handleExtractError"/>

<!-- removewatermark-->
<cf:Pdf id="pdftest" action="removewatermark"
source="http://localhost:8500/CFFileServlet/
_cfservicelayer/_cf3466030530070122606.pdf"
resultHandler="handleExtractResult"
errorHandler="handleExtractError"/>

<!-- protect-->
<cf:Pdf id="pdftest" action="protect"
source="http://localhost:8500/1page.pdf"
newUserPassword="test" permissions="All"
resultHandler="handleExtractResult"
errorHandler="handleExtractError"/>

<!-- mergespecificpages-->
<cf:Pdf id="pdftest" action="mergespecificpages"
pdfParam="{pdfparams}" keepBookmark="true"
resultHandler="handleExtractResult"
errorHandler="handleExtractError"/>

<!-- set info-->
```

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cf:Pdf id="pdftest" action="setinfo"
source="http://localhost:8500/1page.pdf" info="{elements}"
resultHandler="handleExtractResult"
errorHandler="handleExtractError"/>

<!-- thumbnail-->
<cf:Pdf id="pdftest" action="thumbnail"
source="http://localhost:8500/EC205W_JoelGeraci.pdf"
resultHandler="handleThumbnailResult"
errorHandler="handleThumbnailError"/>

<!-- ProcessDDX-->
<cf:Pdf id="pdftest" action="processddx"
```

© 2014 Adobe Systems Incorporated. All rights reserved.

```
ddxString="{ddx}" outputFiles="{outputFiles}" result="handleProcessDDXResult"
fault="handleProcessDDXError"/>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

Using the Pop class

The Pop class is the proxy for the ColdFusion Pop service, which provides the functionality of the `cfpop` tag. You specify the cfpop action and required attributes as Pop object properties and call the `object execute()` function to run the service. The following example shows the user each supported action:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" layout="absolute"
xmlns:cf="coldfusion.service.mxml.*" creationComplete="init()">
<mx:Script>
<![CDATA[
import mx.rpc.events.ResultEvent;
import mx.rpc.events.FaultEvent;
import mx.controls.Alert;
import coldfusion.service.events.*;
 public function init():void
  {
   poptest.execute();
  }
 public function handleResult(event:ResultEvent):void
  {
   Alert.show("Success" + event.toString());
  }
 public function handleError(event:FaultEvent):void
  {
   Alert.show("Failure");
  }
 public function handleGetAll(event:ResultEvent):void
  {
   var res:Array = event.result as Array;
   for(var i:uint = 0; i < res.length; i++)
   {
    var key:String;
    for(key in res[i])
    {
     trace("object key = " + key.toString());
     if(res[i][key] != null)
     {
      trace("object value = " + res[i][key].toString());
     }
    }
   }
  }
 ]]>
</mx:Script>

<cf:Config id="configid" cfServer="localhost" cfPort="8500"
servicePassword="service" serviceUserName="service"  />

<!--<cf:Pop id="poptest" action="getall" result="handleGetAll(event)"
host="xx.xxx.xx.xxx" userName="failoveruser" password="password"
fault="handleError(event)"/>-->

<!--<cf:Pop id="poptest" action="getheaderonly" result="handleGetAll(event)"
host="xx.xxx.xx.xxx" userName="failoveruser" password="password"
fault="handleError(event)"/>-->

<cf:Pop id="poptest" action="delete" messageNumber="25" host="xx.xxx.xx.xxx"
userName="failoveruser" password="password" result="handleResult(event)"
fault="handleError(event)" />
</mx:Application>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

#back to top

© 2014 Adobe Systems Incorporated. All rights reserved.

Using the Util class

The Image, PDF, and Mail services typically act on a file that has been uploaded to the server. To upload the file to your CF server, use the Util class to run the ColdFusion Upload service. The Util class consists of two elements:

- The `UPLOAD_URL` constant contains the URL on the ColdFusion server of the Upload service, relative to *cf_webroot*.
- The `extractURLFromUploadResponse()` method takes response returned by the Upload service as input and returns the path of the uploaded file on the ColdFusion server.
  You use the `UPLOAD_URL` constant and the `extractURLFromUploadResponse()` function in the following workflow to upload a file and use the file in a ColdFusion service.

Event flow of the ColdFusion service

1. Use the ActionScript `flash.net.FileReference` APIs and the `Util.UPLOAD_URL` variable to upload an Image, PDF, or mail attachment to the server on which the action has to be performed. The upload URL to supply to the `flash.net.FileReference`APIs can be constructed as follows in the ActionScript part of the application:

   ```
   uploadURL.url =
   "http://"+conf.cfServer+":"+conf.cfPort+"/"+conf.contextRoot+"/"+Util.UPLOAD_U
   RL;
   var variables:URLVariables = new URLVariables();
    variables.serviceusername = conf.serviceUserName;
    variables.servicepassword = conf.servicePassword;
    uploadURL.data = variables;
    uploadURL.method="POST";
   ```

   Here, specify "conf.cfServer", "conf.cfPort" and "conf.ContextRoot" in the <cf:Config> tag. Specify "conf.ContextRoot" only if ColdFusion is deployed as a J2EE application.

   > **Note**
   >
   > The ActionScript FileUpload functionality is out of the scope for this feature hence it is not explained in detail, but an example of the usage is provided in code in MAIL class section. For further information on FileUpload functionality see the ActionScript documentation.

2. Once the file is uploaded, the server returns an XML response containing URL of the uploaded file. Use the Util class function `extractURLFromUploadResponse()` to extract the URL from the XML
3. Use the file URL in the source attribute of the service tags.
4. When the required service tag attributes are set, run the service action by calling the following method:

   ```
   serviceObject.execute()
   ```

5. If the action succeeds, the server returns the result. If there is an exception, it returns the fault. Handle the `ResultEvent` and `FaultEvent` objects in the service-specific result handler and fault handler that you specify in the service tag, or in global handlers that you specify in the <cf:config>. The ResultEvent object contains the URL of the File on which the operations have been performed. Users can save this file by downloading it on their machine or rendering it in the application. The FaultEvent object contains the exception details that occurred on the server while performing the operation.
   For granular control over proxy classes, you can get hold of the underlying RemotObject by using

© 2014 Adobe Systems Incorporated. All rights reserved.

getRemoteObject() method on the proxy class object. For example, for <cf:Mail id="mailId">, you can get it using the following code in ActionScript.

```
var mailobject:RemoteObject = mailId.getRemoteObject();
```

© 2014 Adobe Systems Incorporated. All rights reserved.

Using the LiveCycle Data Services ES Assembler

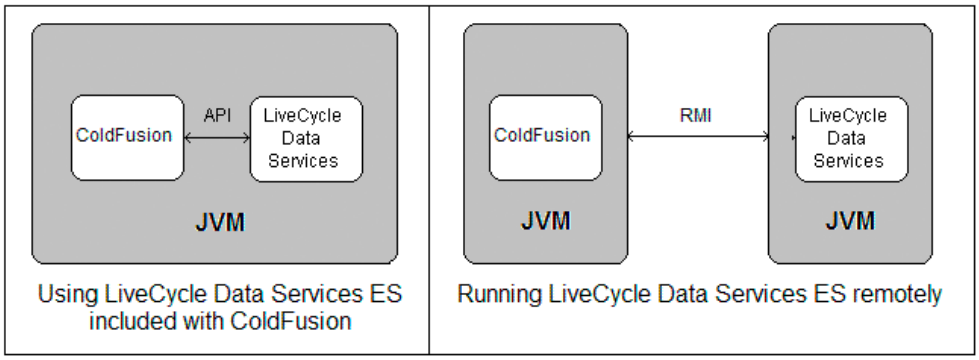
To use Adobe ColdFusion as the back-end data manager for an Adobe Flex application, you use the Adobe LiveCycle Data Services ES assembler. You configure the LiveCycle Data Services ES assembler and write an application that uses the assembler.

To use LiveCycle Data Services ES with ColdFusion, you have to be familiar with ColdFusion components; accessing and using data in ColdFusion applications; and using LiveCycle Data Services ES.

© 2014 Adobe Systems Incorporated. All rights reserved.

About ColdFusion and Flex-Developing guide

ColdFusion 9 supports LiveCycle Data Services 2.6.1 however the ColdFusion installation does not provide the option to install LiveCycle Data Services. You need to manually install LiveCycle Data Services to use it with ColdFusion.
By default, ColdFusion installs BlazeDS which provides messaging support in ColdFusion.
For more information about the changes in using Flash Remoting, see *Changes in the XML configuration files for Flash Remoting in ColdFusion 9 and ColdFusion 9.0.1*.
The LiveCycle Data Services ES assembler lets you use ColdFusion components (CFCs) to provide the back-end data management for a Flex application that uses the Data Management Service. You can run LiveCycle Data Services ES as part of ColdFusion or remotely. If you are running LiveCycle Data Services ES as part of ColdFusion, LiveCycle Data Services ES and ColdFusion communicate directly. If you are running LiveCycle Data Services ES remotely, LiveCycle Data Services ES and ColdFusion communicate by using RMI. The following diagram shows how ColdFusion and LiveCycle Data Services ES interact in both cases:



Note

To use the LiveCycle Data Services ES assembler, the Flex application must be running on Flex Data Services 2.0.1 or LiveCycle Data Services 2.5, although not every feature is supported in Flex Data Services 2.0.1.

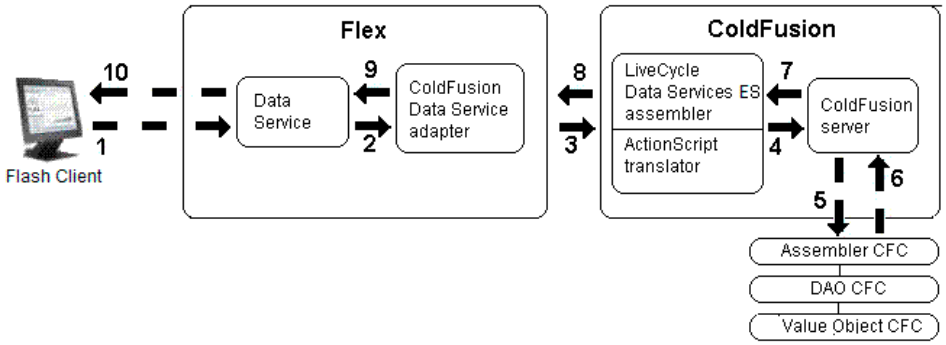
The Flex server includes a ColdFusion Data Service adapter. The adapter processes changes to data to ensure that data on the client is synchronized with back-end data and conversely; it executes the `sync`, `fill`, `count`, and `get` operations, identifies conflicts, and passes results to LiveCycle Data Services ES.
ColdFusion includes the LiveCycle Data Services ES assembler; along with the ActionScript translator, it converts the input arguments where necessary and translates the return values.

Note

If you install LiveCycle Data Services ES, ColdFusion does not map .SWF files. This means that all .SWF files are served through the ColdFusion web application instead of the web server.

The following diagram shows the process that LiveCycle Data Services ES and ColdFusion use when a Flex application calls a method in a ColdFusion component:

© 2014 Adobe Systems Incorporated. All rights reserved.



1. A Flash client requests data that the LiveCycle Data Management Service adapter handles.
2. Flex calls a `fill`, `sync`, `get`, or `count` method in the Data Service.
3. If you are running LiveCycle Data Services ES remotely, copy the lcds2.6_install/resources/lib/flex-*.jar files to the <ColdFusion_home>/lib/ directory for a standalone deployment or WEB-INF/cfusion/lib directory for J2EE deployment.
4. The ColdFusion Data Service adapter sends the request to the LiveCycle Data Services ES assembler. If you are running LiveCycle Data Services ES remotely, the adapter sends the request by using Java Remote Method Invocation (Java RMI).
5. The LiveCycle Data Services ES assembler and the ActionScript translator convert ActionScript 3.0 data types to the appropriate ColdFusion values.
6. The ColdFusion server runs the `fill`, `sync`, `get`, or `count` method of the assembler CFC, which runs the appropriate methods in the DAO CFC.
7. The ColdFusion application creates an array of Value Objects or appropriate return value, which it sends to the ColdFusion server.
8. The ColdFusion server sends the results to the LiveCycle Data Services ES assembler.
9. The LiveCycle Data Services ES assembler and the ActionScript translator convert ColdFusion values to the appropriate ActionScript 3.0 data types, and then the assembler sends the results to the ColdFusion Data Service adapter.
10. The ColdFusion Data Service adapter sends the results to the LiveCycle Data Management Service.
11. The LiveCycle Data Management Service passes the results to the Flash client.

Note

The RMI registry, which facilitates communication between the ColdFusion Data Service assembler and the remote LiveCycle Data Management Service uses port 1099, which is the default port for Java RMI. You can change the port number by adding -Dcoldfusion.rmiport=1234 to the Java JVM arguments on both the ColdFusion server and the Flex server.

© 2014 Adobe Systems Incorporated. All rights reserved.

Application development and deployment process-Developing guide

The following is a typical process for developing and deploying a Flex application that uses the ColdFusion Data Service adapter and LiveCycle Data Services ES assembler to manage back-end database tasks:

1. Design your application.
2. Create the Flex application, in which you define a DataService component in MXML or ActionScript. The DataService component calls methods on a server-side Data Management Service destination to perform activities such as filling client-side data collections with data from remote data sources and synchronizing the client and server versions of data.
3. Configure a destination for the ColdFusion Data Service adapter so that the Flex application to connect to the ColdFusion back-end application. For more information, see Configuring a destination for the Data Service Adapter.
4. Write your ColdFusion CFCs. For more information, see Writing the ColdFusion CFCs.

> **Note**
>
> To make creating the CFCs easier, ColdFusion includes wizards that you can use in Flash Builder. For more information, see Using the ColdFusion Extensions for Eclipse.

1. Test your application by using Flex.

© 2014 Adobe Systems Incorporated. All rights reserved.

Configuring a destination for the Data Service Adapter

To provide the information necessary for the Flex application to connect to the ColdFusion back-end application, you configure a destination. In the destination, you specify the ColdFusion Data Service adapter, the channels to use to transport messages to and from the destination, the CFC that contains the `fill`, `get`, `sync`, and `count` methods, and other settings.
To provide configuration information, you edit the following files:

1. services-config.xmlYou specify channel definitions and enable ColdFusion-specific debugging output in the Flex console in this file. Change the port numbers in the services-config.xml file for the RTMP channels if you run more than one ColdFusion instance with the integrated LiveCycle Data Services ES.
2. data-management-config.xmlThis file is added only when you install LiveCycle Data Services 2.6.1 manually. You specify adapters and destinations in this file.
   To ensure that Flex recognizes the LiveCycle Data Services ES assembler and can transport messages to and from the destination, by doing the following:

- Specifying ColdFusion-specific channel definitions
- Specifying the ColdFusion Data Service adapter
- Specifying a destination
- Enabling ColdFusion-specific debugging output

Enhanced Flash Remoting

ColdFusion 10 has introduced enhanced Flash Remoting.

- Enhanced Flash Remoting supports Circular references for objects which are not supported in old Flash Remoting (ColdFusion 8).
- Enhanced Flash Remoting is significantly faster than the old one
  The default ColdFusion 10 installation makes use of this Enhanced Flash Remoting. The structure inside the xml configuration files under WEB-INF/flex directory has been changed to support Enhanced Flash Remoting. ColdFusion 10 supports Old XML configurations files as well, but with these files it will not be using Enhanced Flash Remoting. To take advantage of Enhanced Flash Remoting, while migrating your old XML files to ColdFusion10, you should make sure that they comply with new XML structural changes.
  If you are using LCDS with ColdFusion 10, then Flash Remoting works on LCDS 2.6.1. Ensure that you are on LCDS 2.6.1 to make use of the enhanced Flash Remoting.
  ColdFusion 10 is backward compatible with old Flash Remoting as well, to support LCDS2.5 and prior releases.
  Now in a case, where you require to continue with LCDS 2.5 or prior versions, it will not be possible for you to use enhanced Flash Remoting offered by ColdFusion 10. In this scenario, you can continue working with LCDS2.5 and prior versions with old style XML configuration files.
  To use the old style flash remoting with LCDS 2.5, first remove the existing ColdFusion 10 flex-**.jar files from `cfusion/lib` by taking a backup of the files and placing the `LCDS2.5 flex-`**`.jar` files in the `cfusion\lib` directory. After this, you can continue to use the old style (ColdFusion 8) XML configuration files by placing them under `WEB-INF\Flex` directory. Also make sure that `WEB-INF\flex\jar\cfdataserviceadapter.jar` is also present.
  For detailed steps on integrating LCDS2.5 with ColdFusion, see the technote on www.adobe.com.
  Another case here could be that you have integrated LCDS 2.6.1 with ColdFusion 10 and still you want to use old Flash Remoting. You can implement this scenario, but in this case you cannot take advantages offered by enhanced Flash Remoting.

Changes in the XML configuration files for Flash Remoting in ColdFusion 10

For ColdFusion 9, the structure of the services-config.xml file has changed. These structural changes are:

- A new `<coldfusion>` tag has been added under the <properties> in `<channel-definition>`, where the

© 2014 Adobe Systems Incorporated. All rights reserved.

`<access>`, `<use-accessor>`, `<use-structs>`, and `<property-case>` tags are defined.In old ColdFusion 8 style remoting, these tags used to be present in the destination, defined in data-management-config.xml file.

- Previously, the `<serialization>`tag included:

```
<serialization>
 <instantiate-types>false</instantiate-types>
</serialization>
```

However, now you need to either set the `<instantiate-types>` to `true` or remove it from the services-config.xml file.

- The `<enable-small-messages>` flag must be set to `false`under the serialization property.

> **Note**
>
> In case you create custom channel definition on your client side by overriding the XML-based channel configurations, you still need to set "enableSmallMessages" flag to false. This is shown in the following code example:

```
<mx:ChannelSet id="myChannelSet">
<mx:channels>
<mx:AMFChannel enableSmallMessages="false"
url="http://localhost:8500/flex2gateway/cfamfpolling" id="cfAMFPolling"
pollingEnabled="true" pollingInterval="8"/>
</mx:channels>
</mx:ChannelSet>
```

- In ColdFusion 9, the endpoint class names have been changed from the endpoint classes in ColdFusion 8. The following table provides a list of channel-definitions and their corresponding endpoint classes:

Channel-definition ID	ColdFusion 10 Endpoint Class	ColdFusion 8 Endpoint Class
`my-cfamf`	`coldfusion.flash.messaging.CFAMFEndPoint`	`flex.messaging.endpoints.AMFEndpoint`
`cf-polling-amf`	`coldfusion.flash.messaging.CFAMFEndPoint`	`flex.messaging.endpoints.AMFEndpoint`
`my-cfamf-secure`	`coldfusion.flash.messaging.SecureCFAMFEndPoint`	`flex.messaging.endpoints.SecureAMFEndpoint`
`cf-rtmp`	`coldfusion.flash.messaging.CFRTMPEndPoint`	`flex.messaging.endpoints.RTMPEndpoint`

© 2014 Adobe Systems Incorporated. All rights reserved.

> **Note**
>
> For LiveCycle Data Services ES, the cf-polling-amf and cf-rtmp channel definitions are used.

- ColdFusion 10 supports BlazeDS 4. The following table details the endpoint classes for BlazeDS:

Channel-definition ID	Endpoint Class
my-streaming-amf	coldfusion.flash.messaging.CFStreamingAMFEndPoint
secure-streaming-amf	coldfusion.flash.messaging.SecureCFStreamingAMFEndPoint

- ColdFusion 10 supports LCDS 3 and LCDS 3.1.The following table details the endpoint classes for LCDS:

Channel-definition ID	Endpoint Class
my-nio-amf	coldfusion.flash.messaging.CFNIOAMFEndPoint secure-nio-amf
secure-nio-amf	coldfusion.flash.messaging.SecureCFNIOAMFEndPoint
secure-cf-rtmp	coldfusion.flash.messaging.SecureCFRTMPEndPoint
my-nio-amf-stream	coldfusion.flash.messaging.CFStreamingNIOAMFEndPoint
secure-nio-amf-stream	coldfusion.flash.messaging.SecureCFStreamingNIOAMFEndPoint

- In ColdFusion 10, a channel-definition construct has been introduced in services-config.xml (*CF_root*/wwroot/WEB-INF/flex/) named `serialize-array-to-arraycollection`.This construct provides more flexibility and control for users to decide whether to serialize the ColdFusion array to ActionScript Array or ArrayCollection.To serialize, in the XML, set the value to `true` for the following:`<serialize-array-to-arraycollection>false</serialize-array-to-arraycollection>`

  > **Note**
  >
  > This construct is not considered when ColdFusion Array is sent to LCDS Flex Client. In this case, ColdFusion Array always gets translated to ActionScript ArrayCollection.

- All the other files that need to be referenced from services-config.xml are now included in services-config.xml. In ColdFusion 8, the services-config.xml files looked similar to the following:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<channel-definition id="cf-polling-amf" class="mx.messaging.channels.AMFChannel">
<endpoint url="http://{server.name}:{server.port}{context.root}/
 flex2gateway/cfamfpolling" class="flex.messaging.endpoints.AMFEndpoint"/>
   <properties>
   <polling-enabled>true</polling-enabled>
    <serialization>
     <instantiate-types>false</instantiate-types>
    </serialization>
   </properties>
</channel-definition>

<channel-definition id="cf-rtmp" class="mx.messaging.channels.RTMPChannel">
<endpoint url="rtmp://{server.name}:2048"
class="flex.messaging.endpoints.RTMPEndpoint"/>
<properties>
 <idle-timeout-minutes>20</idle-timeout-minutes>
  <serialization>
   <!-- This must be turned off for any CF channel -->
    <instantiate-types>false</instantiate-types>
   </serialization>
</properties>
</channel-definition>
```

The new services-config.xml file looks similar to this:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<channel-definition id="cf-polling-amf "class="mx.messaging.channels.AMFChannel">
<endpoint uri="http://{server.name}:{server.port}{context.root}
/flex2gateway/cfamfpolling" class="coldfusion.flash.messaging.CFAMFEndPoint" />
<properties>
 <polling-enabled>true</polling-enabled>
 <polli ng-interval-seconds>8</polling-interval-seconds>
 <serialization>
  <enable-small-messages>false</enable-small-messages>
 </serialization>
 <coldfusion>
  <access>
   <use-mappings>true</use-mappings>
   <method-access-level>remote</method-access-level>
  </access>
  <use-accessors>true</use-accessors>
  <use-structs>false</use-structs>
  <property-case>
   <force-cfc-lowercase>false</force-cfc-lowercase>
   <force-query-lowercase>false</force-query-lowercase>
   <force-struct-lowercase>false</force-struct-lowercase>
  </property-case>
 </coldfusion>
</properties>
</channel-definition>
<channel-definition id="cf-rtmp" class="mx.messaging.channels.RTMPChannel">
<endpoint uri="rtmp://{server.name}:2048"
class="coldfusion.flash.messaging.CFRTMPEndPoint"/>
 <properties>
   <idle-timeout-minutes>20</idle-timeout-minutes>
 <serialization>
  <enable-small-messages>false</enable-small-messages>
 </serialization>
   <coldfusion>
    <access>
     <use-mappings>true</use-mappings>
     <method-access-level>remote</method-access-level>
    </access>
    <use-accessors>true</use-accessors>
    <use-structs>false</use-structs>
    <property-case>
     <force-cfc-lowercase>false</force-cfc-lowercase>
     <force-query-lowercase>false</force-query-lowercase>
     <force-struct-lowercase>false</force-struct-lowercase>
    </property-case>
   </coldfusion>
 </properties>
</channel-definition>
```

Specifying ColdFusion-specific channel definitions

LiveCycle Data Services ES transports messages to and from destinations over message channels that are part of the Flex messaging system. When you configure a destination, you reference the messaging channels to use. To connect to a ColdFusion back-end application, ensure that the `services-config.xml` file contains definitions for the `cf-polling-amf` channel and the `cf-rtmp` channel in the channels section. If you are running LiveCycle Data Services ES in ColdFusion, the services-config.xml file is in the `wwwroot\WEB-INF\flex` directory and

© 2014 Adobe Systems Incorporated. All rights reserved.

contains the channel definitions by default. If you are running LiveCycle Data Services ES remotely, the services-config.xml file is located in the under `\WEB-INF\flex` directory when you install LiveCycle Data Services ES in the default location.
The new channel definitions include the following:

```
<channel-definition id="cf-polling-amf" class="mx.messaging.channels.AMFChannel ">
<endpoint uri="http://{server.name}:{server.port}{context.root}
/flex2gateway/cfamfpolling"class="coldfusion.flash.messaging.CFAMFEndPoint" />
<properties>
 <polling-enabled> true </polling-enabled>
 <polling-interval-seconds>8</polling-interval-seconds>
 <serialization>
  <enable-small-messages>false</enable-small-messages>
 </serialization>
 <coldfusion>
  <access>
   <use-mappings>true</use-mappings>
   <method-access-level>remote</method-access-level>
  </access>
  <use-accessors>true</use-accessors>
  <use-structs>false</use-structs>
  <property-case>
   <force-cfc-lowercase>false</force-cfc-lowercase>
   <force-query-lowercase>false</force-query-lowercase>
   <force-struct-lowercase>false</force-struct-lowercase>
  </property-case>
 </coldfusion>
</properties>
</channel-definition>
```

Element	Description
`access`	Define the resolution rules and access level of the CFC being invoked
`use-accessors`	Whether the Value Object CFC has getters and setters. Set the value of `use-accessors` to `true` if there are getters and setters in the Value Object CFC. However, if you set `use-accessors` to `true` and there are no getters and setters in the value object CFC, ColdFusion sets the value of any property of the value object CFC in the `this` scope. If your CFC does not have any getters and setters, you can increase performance by setting this to `false` so that ColdFusion does not spend time looking for these methods. The default value is `true`.

© 2014 Adobe Systems Incorporated. All rights reserved.

`use-structs`	Whether to translate ActionScript to CFCs. Set the value of `use-structs` to `true` if you don't require any translation of ActionScript to CFCs. The assembler can still return structures to Flex, even if the value is `false` . The default value is `false`.
`force-cfc-lowercaseforce-query-lowercaseforce-struct-lowercase`	Whether to make property names, query column names, and structure keys lowercase when converting to ActionScript. Query column names must precisely match the case of the corresponding ActionScript variables. The default value is `false`.
`use-mappings`	A Boolean value specifying whether the source attribute can be relative to (start with) a ColdFusion mapping. The default value is true.
`method-access-level`	Specifies the `access` attribute values a CFC must have for ColdFusion to respond to the request. The following values are valid: • `remote` Flex can access only functions that specify remote access. (the default) • `public` Flex can access functions that specify both remote or public access.

Specifying the ColdFusion Data Service adapter

Flex provides adapters to connect to various back-end applications. To use the ColdFusion Data Service adapter, you specify it in the data management configuration file by copying the following adapter-definition to the adapters section of the data-management-config.xml file that is in the WEB-INF/flex folder of the server on which you want to run the Flex application. If you are running LiveCycle Data Services ES in ColdFusion, the data-management-config.xml file contains the adapter definitions by default.
The adapter definition includes the following line:

```
<adapter-definition id="coldfusion-dao"
class="coldfusion.flex.CFDataServicesAdapter"/>
```

Specifying a destination

A destination is the server-side service or object that you call. You configure Data Management destinations in the data-management-config.xml file.
The destination contains the following elements:

Element	Description
`destination id`	The ID must be unique for each destination.

© 2014 Adobe Systems Incorporated. All rights reserved.

adapter-ref	The name of the adapter to use. You use the ColdFusion `adapter-ref` element for any ColdFusion specific destinations.
channels-ref	Use the ColdFusion configured channels that have the instantiate-types flag set to `false`.
component	The name or path on the ColdFusion server.
scope	The scope, which can be `application`, `session`, or `request`. The `application` value specifies that there is only one instance; `request` specifies that there is a new CFC for each call. ColdFusion does not support `session`. (Do not confuse this setting with the ColdFusion variable scope; they are not related.)
hostname	The host name or IP address of the ColdFusion host. If you are running LiveCycle Data Services as part of ColdFusion you do not specify a host name or IP address; however, if you are running LiveCycle Data Services ES remotely, specify a host name or IP address.
identity	The ID of the ColdFusion Data Management server as configured in the ColdFusion Administrator.This is required only if you are accessing a ColdFusion server remotely using RMI and have more than one instance of ColdFusion on a machine.
remote-usernameremote-password	Credentials to pass to the assembler CFC for all clients. It is preferable to use the ActionScript `setRemoteCredentials()` API on the client.
identity property	The property or list of properties that are the primary key in the database.
query-row-type	Optional. If the assembler `fill` method returns a query, define an ActionScript type for each row in the query that the ArrayCollection returned.
fill-method	Whether to update the results of a fill operation after a create or update operation.
use-fill-contains	Optional. Whether the assembler has a `fill-contains` method. This method is used to determine whether to refresh the fill. If the specified method returns `true`, the fill is re-executed after a create or update operation. Set `use-fill-contains` to `true` only when `auto-refresh` is set to `true`. The default value is `false`.

© 2014 Adobe Systems Incorporated. All rights reserved.

`auto-refresh`	Optional. Whether to refresh the fill after a create or update operation. The default value is `true`.
`ordered`	Optional. Whether order is important for this filled collection. Allows performance optimization when order is not important. The default value is `true`.

The following code shows a sample destination:

```
<destination id="cfcontact">
<!-- Use the ColdFusion adapter for any CF specific destinations-->
 <adapter ref="coldfusion-dao" />
<channels>
<channel ref="cf-polling-amf" />
</channels>

 <properties>
  <!--The component name or path on the CF server-->
   <component>samples.contact.ContactAssembler</component>
  <!--Either "application" or "request"-->
   <scope>request</scope>
  <!-- The hostname or IP address of the CF host. If Data Services is installed as
   part of CF, you omit this. If Data Services runs outside of CF, you must
   define this. <hostname>localhost</hostname>-->

  <!--This is the ID of the ColdFusion Data Management service as configured in
  the ColdFusion Administrator. Only needed if you have more than one instance of
  CF on a machine and Data Services is not installed as part of CF.
  <identity>default</identity> -->
  <!--Credentials to pass to the assembler CFC for all clients. Generally better
  to use setRemoteCredentials() API on client <remote-username></remote-username>
  <remote-password></remote-password>-->
 <metadata>
  <identity property="contactId" />
  <!--Optional, If the Assembler fill routine returns a query,you must define an
   Actionscript type for the rows.-->
  <query-row-type>samples.contact.Contact</query-row-type>
 </metadata>

 <network>
  <!--  Add network elements here-->
 </network>

 <server>
 <!-- The method declarations are ignored for CFC Assemblers, with the exception of
 the fill-method settings. No parameters are defined here, unlike Java. Any
arguments
 provided via the AS call are passed along to the CFC, just use optional arguments
 when defining the CFC.-->
 <fill-method>
 <!--Does the assembler have a "fill-contains" method? This method is used to
 determine whether to refresh the fill. If the specified method returns true the
fill
 is re-executed after a create or update. Auto-refresh determines if the fill is
 always refreshed if not specified. May only be used when auto-refresh is true.
```

© 2014 Adobe Systems Incorporated. All rights reserved.

```
 Optional. Default is false.-->
 <use-fill-contains>false</use-fill-contains>
 <!-- Determines whether to refresh the fill on updates or creates. Optional.
Default
  value is true.-->
 <auto-refresh>true</auto-refresh>
 <!--Determines whether order is important for this filled collection. Allows for
 performance optimization when order is not important. Optional. Default value is
 true.-->

 <ordered>true</ordered>

 </fill-method>
 </server>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

```
 </properties>
</destination>
```

Enabling ColdFusion-specific debugging output

You enable ColdFusion-specific debugging output in the Flex console by adding the following `<pattern>` tag in the `<filters>` tag in the logging section in the services-config.xml file:

```
<pattern>DataService.coldfusion</pattern>
```

For more information, see "Configuring the Data Service" in *Developing Flex Applications*, which is included in the Flex documentation.

Note

The ColdFusion Administrator lets you enable or disable LiveCycle Data Management support. If you are running more than one instance of ColdFusion, use a unique ID to specify each instance of ColdFusion for which you enable LiveCycle Data Management support. You do so by specifying the identity in the `identity` element in the data-management-config.xml file.

© 2014 Adobe Systems Incorporated. All rights reserved.

Writing the ColdFusion CFCs

When you create your ColdFusion CFCs, you can do one of the following:

- Create an assembler CFC and a Value Object CFC.
- Create an assembler CFC, a Data Access Object (DAO) CFC, and a Value Object CFC.
  You place the database manipulation functionality directly in the methods in the assembler CFC and create a Value Object CFC, which is a CFC that contains property definitions and related `get` and `set` methods.
  To separate the lower level database functionality from the high-level Flex assembler operations, you create a Data Access Object (DAO) CFC that contains the lower level database functionality. Using this approach, which is the Bean/DAO methodology, requires that you place the `fill`, `get`, `sync`, and `count` methods in the assembler CFC. The methods in the assembler CFC call methods in the DAO CFC that perform the lower level database functions such as retrieving records. The DAO CFC creates Value Objects, which are CFCs that contain the values. A Value Object is essentially a row in the result set.
  The LiveCycle Data Management Service recognizes the methods: `fill`, `get`, `sync`, and `count`. The `fill` method retrieves records from a database and populates an array with the records. The `get` method retrieves a specific record. The `sync` method lets you keep track of synchronization conflicts by accepting a change list, which is an array of change objects. The `count` method returns a number that indicates how many records are in a result set. To perform any of these database tasks, the Flex application calls the appropriate `fill`, `get`, `sync`, or `count` method in the assembler CFC. You can also use a `fillContains` method, which checks whether to update the results of a fill. For more information, see *Managing fills* below.

Creating the fill method

The `fill` method retrieves records from a database and populates an array with the records. If you use the Bean/DAO methodology, you create the lower level `read` method separately in the DAO CFC.
The `fill` method returns the results of a read operation. In the `fill` method, you create an array to hold the results of the read, and then return the results of the read operation. The essential elements of a `fill` method appear as follows:

```
<cffunction name="fill" output="no" returntype="samples.contact.Contact[]"
access="remote">
 <cfreturn variables.dao.read()>
</cffunction>
```

You can return a Value Object CFC, a query, or an array of CFML structures. Using a query instead of a Value Object CFC may improve performance. However, ColdFusion cannot handle nested results sets when you use a query. For example, if one of the CFC properties you are returning from the `fill` method was populated with another complex type such as another CFC type, ColdFusion cannot automatically convert a column in the query to an object with a custom type. In this case, you return an array of CFCs, and the `fill` method or the `read` method in the DAO CFC constructs the correct object.
You can use structures wherever you currently create a ColdFusion component in the Assembler. However, you still receive CFC Value Objects from Flex. For example, the Change Objects that you receive in the `sync` method contain CFCs, assuming that you have a remote alias defined in the ActionScript type.
You can create Value Object CFCs in the `get` method. However, using the structure functionality, you can create and return a structure instead of a CFC, because the structures are translated in the same way as CFCs. You can also return an array of structures from the `fill` method instead of an array of CFCs, for example, if you have to do processing on your data and working with CFCs isn't fast enough. Generally, structures are faster than CFCs. You also use structures when a member of the result object is a complex object. In this case, you create another structure as the value of that key and provide the *_type_* key for it.
You specify the returntype of the `fill` method as a Value Object CFC, a query, or an array:

© 2014 Adobe Systems Incorporated. All rights reserved.

1. Value Object:

```
<cffunction name="fill" output="no"
 returntype="samples.contact.Contact[]" access="remote">
```

2. Query:

```
<cffunction name="fill" output="no"
 returntype="query" access="remote">
```

3. Array of structures:

```
<cffunction name="fill" output="no"
 returntype="array" access="remote">
```

In addition to specifying the returntype of the `fill` function depending on whether you are using Value Objects, a query, or an array of structures, you also do the following in the lower level `read` function:

- Specify the returntype of the `read`function as the Value Object CFC, a query, or an array, for example:
  - `<cffunction name="read" output="false" access="public" returntype="`**samples.contact.Contact[]**`">`
  - `<cffunction name="read" output="false" access="public" returntype="`**query**`">`
  - `<cffunction name="read" output="false" access="public" returntype="`**array**`">`
- If you are using Value Objects:
  - Create the array to contain the Value Objects, as follows:

```
<cfset var ret = ArrayNew(1)>
```

-
  - Loop through the query to create each Value Object based on each row of the query, for example:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cfloop query="qRead">
 <cfscript>
  obj = createObject("component",
   "samples.contact.Contact").init();
  obj.setcontactId(qRead.contactId);
  obj.setfirstName(qRead.firstName);
  obj.setlastName(qRead.lastName);
  obj.setaddress(qRead.address);
  obj.setcity(qRead.city);
  obj.setstate(qRead.state);
  obj.setzip(qRead.zip);
  obj.setphone(qRead.phone);
  ArrayAppend(ret, obj);
 </cfscript>
</cfloop>
```

- If you are using a query:
  - Ensure that you configured the destination with the row type for the destination so that ColdFusion correctly labels each rows in the query with the corresponding ActionScript type. Use the `query-row-type` element, which is in the metadata section of the destination.
  - Specify the following in the `fill`method:

```
<cffunction name="fill" output="no" returntype="query"
 access="remote">
 <cfargument name="param" type="string" required="no">
  <cfquery name="myQuery" .>
  </cfquery>
  <!--- Return the result --->
 <cfreturn myQuery>
</cffunction>
```

-
  - If you are using a DAO CFC, edit the `read` method to return a query instead of an array of CFCs.
  - Ensure that the query column names match the case of the properties in the ActionScript object. Use the `property-case` settings in the destination to do so. Set the `force-query-lowercase` element to `false` so that ColdFusion converts all column names to lowercase.
- If you are using an array of structures:
  - Create the array to contain the Value Objects, as follows:

```
<cfset var ret = ArrayNew(1)>
```

-
  - Loop through the query to create the structure that contains the results of the query, for example:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cfloop query="qRead">
 <cfscript>
  stContact = structNew();
  stContact["__type__"] = "samples.contact.Contact";
  stContact["contactId"] = qRead.contactId;
  stContact["firstName"] = qRead.firstName;
  stContact["lastName"] = qRead.lastName;
  stContact["address"] = qRead.address;
  stContact["city"] = qRead.city;
  stContact["state"] = qRead.state;
  stContact["zip"] = qRead.zip;
  stContact["phone"] = qRead.phone;
  ArrayAppend(ret, duplicate(stContact));
 </cfscript>
</cfloop>
```

- - Use the *_type_*structure element to specify that the Value Object CFC is the type, for example:

```
stContact["__type_"] = "samples.contact.Contact";
```

  - Use the associative array syntax, for example, `contact"firstName"` to ensure that you match the case of the ActionScript property. If you use the other syntax, for example, `contact.firstName="Joan"`, ColdFusion makes the key name uppercase.

Managing fills

To determine whether to refresh a fill result after an item is created or updated, you include a `fillContains` method in the assembler and set both `use-fill-contains` and `auto-refresh` to `true` in the fill-method section of the data-management-config.xml file. The following example shows a fill-method section:

```
<fill-method>
 <use-fill-contains>true</use-fill-contains>
 <auto-refresh>true</auto-refresh>
 <ordered>false</ordered>
</fill-method>
```

In this example, `ordered` is set to `false` because the fill result is not sorted by any criteria. However, if the fill result is sorted, you set `ordered` to `true`. When an item changes in a fill result that is ordered, refresh the entire fill result.
The `fillContains` method tells the Flex application whether it is necessary to run the fill again after an item in the fill result has changed. The `fillCcontains` method returns a value that indicates how the fill be treated for that change. When the `fillContains` method returns `true`, the fill is executed after a create or update operation.
The following example shows the `fillContains` method signature:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cffunction name="fillContains" output="no" returnType="boolean" access="remote">
  <cfargument name="fillArgs" type="array" required="yes">
  <cfargument name="item" type="[CFC type object]" required="yes">
  <cfargument name="isCreate" type="boolean" required="yes">
```

The `fillContains` method has the following arguments:

- `fillArgs` is a list of arguments to pass to the `fill` method.
- `item` is the record to check to determine if it is in the result set.
- `isCreate` indicates whether the record is new.
  A sample `fillContains` method, which determines whether the `fill` arguments (part of the first or last name) are in the Contact item passed to the function, is as follows:

```
<cffunction name="fillContains" output="no" returnType="boolean"access="remote">
 <cfargument name="fillArgs" type="array" required="yes">
 <cfargument name="item" type="samples.contact.Contact" required="yes">
 <cfargument name="isCreate" type="boolean" required="yes">

<cfif ArrayLen(fillArgs) EQ 0>
<!--- This is the everything fill. --->
<cfreturn true>
<cfelseif ArrayLen(fillArgs) EQ 1>
<!--- This is a search fill. --->
 <cfset search = fillArgs[1]>
 <cfset first = item.getFirstName()>
 <cfset last = item.getLastName()>
 <!--- If the first or last name contains the search string, --->
 <cfif (FindNoCase(search, first) NEQ 0) OR (FindNoCase(search, last)
  NEQ 0)>
<!--- this record is in the fill. --->
<cfreturn true>
<cfelse>
<!--- this record is NOT in the fill. --->
<cfreturn false>
</cfif>
</cfif>

<!--- By default, do the fill.--->
<cfreturn true>
</cffunction>
```

If you are running LiveCycle Data Services ES locally, you can determine whether a `fill` operation is a refresh or a client triggered fill. You do so by calling the DataServiceTransaction.getCurrentDataServiceTransaction().isRefill() method in your ColdFusion application as follows:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cfscript>
dst = CreateObject("java", "flex.data.DataServiceTransaction");
t = dst.getCurrentDataServiceTransaction();
isRefill = t.isRefill();
</cfscript>
```

This does not work over RMI when ColdFusion and Flex are not in the same web application.

Creating the get method

The `get` method retrieves a specific record. The get method calls the lower level `read` method. If you use the Bean/DAO methodology, as described in Writing the ColdFusion CFCs, you create the lower level `read` method separately in the DAO CFC.
The following example shows the essential elements of a `get` method:

```
<cffunction name="get" output="no" returnType="samples.contact.Contact"
access="remote">
 <cfargument name="uid" type="struct" required="yes">
 <cfset key = uid.contactId>
 <cfset ret=variables.dao.read(id=key)>
 <cfreturn ret[1]>
</cffunction>
```

The returntype of a `get` method can be any of the following:

- The Value Object CFC
- Any
- An array

Creating the sync method

The `sync` method lets you keep track of synchronization conflicts by accepting a change list, which is an array of change objects. In the `sync` method, you pass in an array of changes, loop over the array and apply the changes, and then return the change objects, as follows:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cffunction name="sync" output="no" returnType="array" access="remote">
 <cfargument name="changes" type="array" required="yes">

 <!-- Create the array for the returned changes. -->
 <cfset var newchanges=ArrayNew(1)>

 <!-- Loop over the changes and apply them. --->
 <cfloop from="1" to="#ArrayLen(changes)#" index="i" >
  <cfset co = changes[i]>
  <cfif co.isCreate()>
   <cfset x = doCreate(co)>
  <cfelseif co.isUpdate()>
   <cfset x = doUpdate(co)>
  <cfelseif co.isDelete()>
   <cfset x = doDelete(co)>
  </cfif>
  <cfset ArrayAppend(newchanges, x)>
 </cfloop>

 <!-- Return the change objects, which indicate success or failure. --->
 <cfreturn newchanges>
</cffunction>
```

Creating the count method

The `count` method returns a number that indicates how many records are in a result set. If you use the Bean/DAO methodology, as described in Writing the ColdFusion CFCs, you create the lower level `count` method separately in the DAO CFC.
The `count` method contains the following essential elements, without any error handling:

```
<cffunction name="count" output="no" returntype="Numeric" access="remote">
 <cfargument name="param" type="string" required="no">
 <cfreturn variables.dao.count()>
</cffunction>
```

This `count` method calls a different `count` method in the DAO CFC, which contains the following essential elements, without any error handling:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cffunction name="count" output="false" access="public" returntype="Numeric">
 <cfargument name="id" required="false">
 <cfargument name="param" required="false">
 <cfset var qRead="">

 <cfquery name="qRead" datasource="FDSCFCONTACT">
  select COUNT(*) as totalRecords
  from Contact
 </cfquery>

 <cfreturn qRead.totalRecords>
</cffunction>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

Notifying the Flex application when data changes

You use the LiveCycle Data Services ES event gateway type provided with ColdFusion, to have ColdFusion applications notify Flex when data that a destination manages has changed. You configure the LiveCycle Data Services ES event gateway and write an application that uses the event gateway. For more information, see Using the Data Management Event Gateway.

© 2014 Adobe Systems Incorporated. All rights reserved.

Authentication

To authenticate users when using the LiveCycle Data Services ES assembler, you use the Flex `setRemoteCredentials()` method on the DataService object. The credentials, which are in the FlexSession object, are passed to the ColdFusion application, where you can use the `cflogin` tag to perform authentication. Alternatively, you can set credentials in the Flex destination, although it is not the recommended way to do so.
You can set the credentials by doing either of the following:

- Specifying credentials in ActionScript
- Specifying credentials in the Flex destination

Specifying credentials in ActionScript

To specify credentials in ActionScript, you use the `setRemoteCredentials()` method, as the following example shows:

```
ds = new DataService("mydest");
ds.setRemoteCredentials("wilsont", "password");
```

Specifying credentials in the Flex destination

To specify credentials in the Flex destination, you edit the data-management-config.xml file that is in the WEB-INF/flex folder of the server on which you run the Flex application. In the `properties` element, you include the `remote-username` and `remote-password` elements, as follows:

```
<destination id="cfcontact">
 <adapter ref="coldfusion-dao" />
  <channels>
   <channel ref="cf-dataservice-rtmp" />
  </channels>
 <properties>
  <source>samples.contact.ContactAssembler</source>
  <scope>application</scope>
  <remote-username>wilsont</remote-username>
  <remote-password>password</remote-password>
...
 /properties>
</destination>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

Enabling SSL

You encrypt communication between ColdFusion and Flex by enabling Secure Sockets Layer (SSL). Enabling SSL only makes sense if you are running LiveCycle Data Services ES remotely. To use SSL, create a *keystore* file. The keystore is a self-signed certificate. (You do not require a certificate signed by a Certificate Authority, although if you do use one, you do not have to configure Flex as indicated in the following steps.) The information in the keystore is encrypted and can be accessed only with the password that you specify. To create the keystore, you use the Java keytool utility, which is included in your Java Runtime Environment (JRE).
To enable SSL, you do the following:

1. Create the keystore
2. Configure Flex
3. Enable SSL in the ColdFusion Administrator

Create the keystore

Generate the SSL server (ColdFusion) keystore file by using the keytool utility, with a command like the following:

```
keytool -genkey -v -alias FlexAssembler -dname "cn=FlexAssembler" -keystore
cf.keystore -keypass mypassword -storepass mypassword
```

The following table describes the parameters of the keytool utility that you use:

Parameter	Description
`-alias`	The name of the keystore entry. You can use any name as long as you are consistent when referring to it.
`-dname`	The Distinguished Name, which contains the Common Name (cn) of the server.
`-keystore`	The location of the keystore file.
`-keypass`	The password for your private key.
`-storepass`	The password for the keystore. The encrypted storepass is stored in ColdFusion configuration files.
`-rfc`	Generates the certificate in the printable encoding format.
`-file`	The name of the keystore file.
`-v`	Generates detailed certificate information.

Next, you place the certificate that you created in the file that the JVM uses to decide what certificates to trust. The file in which you place the certificate (usually named cacerts), is located in the JRE, under the lib/security folder.

Configure Flex

© 2014 Adobe Systems Incorporated. All rights reserved.

1. Export the keystore to a certificate by using the keytool utility, with a command like the following:

```
keytool -export -v -alias FlexAssembler -keystore cf.keystore -rfc -file
cf.cer
```

2. Import the certificate into the JRE cacerts file for your server by using the keytool utility, with a command like the following:

```
keytool -import -v -alias FlexAssembler -file cf.cer -keystore
C:\fds2\UninstallerData\jre\lib\security\cacerts
```

   The previous example specifies the location of the keystore for LiveCycle Data Services ES with integrated JRun, installed using the default settings. If you are using a different server, specify the location of the cacerts file for the JRE that you are using. For example, if you are using JBoss, you specify the keystore location as `$JAVA_HOME/jre/lib/security/cacerts.`

Enable SSL in the ColdFusion Administrator

1. In the ColdFusion Administrator, select Data & Services > Flex Integration, and specify the keystore file in the Full Path to Keystore text box.
2. Specify the keystore password in the Keystore password text box.
3. Select the Enable RMI over SSL for Data Management option, and then click Submit Changes.If you specify an invalid keystore file or password, ColdFusion does not enable SSL, and disables Flex Data Management Support.

© 2014 Adobe Systems Incorporated. All rights reserved.

Data translation-Developing guide

The following table lists the ColdFusion data types and the corresponding Adobe Flash or ActionScript data type:

ColdFusion data type	Flash data type
String	String
Array	[] = Array
Struct	{} = untyped Object
Query	ArrayCollection
CFC	Class = typed Object (if a matching ActionScript class exists, otherwise the CFC becomes a generic untyped Object (map) in ActionScript)
CFC Date	ActionScript Date
CFC String	ActionScript String
CFC Numeric	ActionScript Numeric
ColdFusion XML Object	ActionScript XML Object

#back to top

© 2014 Adobe Systems Incorporated. All rights reserved.

Using Server-Side ActionScript

Adobe ColdFusion server configuration includes the Flash Remoting service, a module that lets Adobe Flash developers create server-side ActionScript. These ActionScript files can directly access ColdFusion query and HTTP features through two new ActionScript functions: `CF.query` and `CF.http`.

© 2014 Adobe Systems Incorporated. All rights reserved.

About server-side ActionScript

ColdFusion includes a module called the Flash Remoting service that acts as a broker for interactions between Flash and ColdFusion. Flash Remoting supports a range of object types, and lets you reference an ActionScript file that lives on a ColdFusion server. You can partition data-intensive operations on the server, while limiting the amount of network transactions necessary to get data from the server to the client.
Flash developers can create server-side ActionScript files to access ColdFusion resources; they do not have to learn CFML (ColdFusion Markup Language). This ability lets you logically separate the Flash presentation elements of your applications from the business logic. You have the option of creating ActionScript files that reside on the server to partition this processing away from your client applications.
You have a simple interface for building queries using server-side ActionScript, and an equally simple interface for running these queries from your client-side ActionScript.

Client-side ActionScript requirements

On the client side, you only need a small piece of code that establishes a connection to the Flash Remoting service and references the server-side ActionScript you want to use.
For example (notice the embedded comments):

```
// This #include is needed to connect to the Flash Remoting service
#include "NetServices.as"

// This line determines where Flash should look for the Flash Remoting service.
// Ordinarily, you enter the URL to your ColdFusion server.
// Port 8500 is the Flash Remoting service default.
NetServices.setDefaultGatewayUrl("http://mycfserver:8500");

// With the Flash Remoting service URL defined, you can create a connection.
gatewayConnnection = NetServices.createGatewayConnection();

// Reference the server-side ActionScript.
// In this case, the stockquotes script file lives in the web root of the
// ColdFusion server identified previously. If it lived in a subdirectory
// of the web root called "mydir," you would reference it
// as "mydir.stockquotes".
stockService = gatewayConnnection.getService("stockquotes", this);

// This line invokes the getQuotes() method defined in the stockquotes
// server-side ActionScript.
stockService.getQuotes("macr");

// Once the record set is returned, you handle the results.
// This part is up to you.
function getQuotes_Result ( result )
{
 // Do something with results
}
```

Note

Client-side ActionScript does not support the two new server-side ActionScript functions, `CF.query` and `CF.http`.

Server-side requirements

© 2014 Adobe Systems Incorporated. All rights reserved.

Creating ActionScript that executes on the server helps leverage your knowledge of ActionScript. It also provides direct access to ColdFusion query and HTTP features. The `CF.query` and `CF.http` ActionScript functions let you perform ColdFusion HTTP and query operations.

> **Note**
>
> On the server side, ActionScript files use the extension .asr.

For example, the following server-side ActionScript code builds on the client-side code shown previously:

```
// Filename: stockquotes.asr
// Here is the getQuotes method invoked in the client-side ActionScript.
// It accepts a single stock quote symbol argument.
function getQuotes(symbol)
{
 // Query some provider for the specified stock quote and return the
 // results. In this case, the getQuotesFromProvider method is
 // defined elsewhere in this ActionScript code.
 data = getQuotesFromProvider(symbol);
 // Return the data to the client.
 // Note: this example does not include any of the error checking
 // logic you would normally use prior to returning the data.
 return data;
}
```

The `getQuotes` function conducts the stock quote request and returns the results of the request to the client as a RecordSet object.

Software requirements

To use server-side ActionScript files, you must have the following software installed:

- Adobe Flash
- ColdFusion
- Flash Remoting Components
  For more information about these products, go to www.adobe.com.

Location of server-side ActionScript files

You can place ActionScript files (`*.asr`) on the server anywhere below the root directory of the web server. To specify subdirectories of the web root or a virtual directory, use package dot notation (use dots instead of slashes in a fully qualified directory name). For example, in the following assignment code, the stockquotes.asr file is located in the mydir/stock/ directory:

```
stockService = gatewayConnnection.getService("mydir.stock.stockquotes", this);
```

You can also point to virtual mappings, such as `cfsuite.asr.stock.stockquotes` where `cfsuite` is a virtual mapping and `asr.stock` is subdirectories of that mapping.

Benefits

© 2014 Adobe Systems Incorporated. All rights reserved.

Server-side ActionScript lets your ActionScript engineers use their knowledge of ActionScript to write code for the back end of their SWF files, which can mean more meaningful levels of interactivity for your users. Your SWF files can share a library of server-side ActionScript functions, which means you can define functions that are tailored to your own business.

You could, for example, create a server-side ActionScript file that defines a whole library of SQL query methods. With these query methods defined on the server side, your Flash designers only have to run the specific query function they want to return data to their SWF movies. They do not have to write any SQL, and they do not have to create a query every time they retrieve data from a ColdFusion data source. It is a way of creating reusable queries that your entire Flash design team can use.

Coding the ColdFusion query and HTTP operations in ActionScript is very straightforward. The `CF.query` and `CF.http` functions provide a well-defined interface for building SQL queries and HTTP operations.

For example, the following is a typical server-side ActionScript function definition that returns query data:

```
// This function shows a basic CF.query operation using only
// arguments for data source name and for SQL.
function basicQuery()
{
 mydata = CF.query({datasource:"customers",
  sql:"SELECT * FROM myTable"});
 return mydata;
}
```

What to do next

If you are already familiar with ActionScript, here a few things to get you started:

- How to establish a connection with the Flash Remoting service using client-side ActionScript. See Connecting to the Flash Remoting service
- How to reference server-side ActionScript functions and methods. See Using server-side ActionScript functions.
- How to code the server-side `CF.query` and `CF.http` functions. See Using the CF.query function and Using the CF.http function. Also see the reference pages for these functions in the *CFML Reference*.
  For additional information on using Flash Remoting, see Using the Flash Remoting Service and *Using Flash Remoting.*

© 2014 Adobe Systems Incorporated. All rights reserved.

Connecting to the Flash Remoting service

Before you can use functions defined in your server-side ActionScript files, connect the Adobe SWF movie to the server-side Flash Remoting service.

Create a Flash Remoting service connection

1. Include the necessary ActionScript classes in the first frame of the SWF movie that uses server-side ActionScript functions.
   a. Use the following command to include the `NetServices`class:

   ```
   #include "NetServices.as"
   ```

   b. (Optional) Use the following command to include the `NetDebug`class:

   ```
   #include "NetDebug.as"
   ```

   For more information about the `NetDebug` and `RecordSet` classes, see *Using Flash Remoting*.

2. Since the Flash Remoting service serves as a broker for calls to server-side ActionScript functions, identify the Flash Remoting service URL as an argument in the `NetServices.setDefaultGatewayUrl`function. For example:

   ```
   NetServices.setDefaultGatewayURL("http://localhost:8500/flashservices")
   ```

   Specify a server host name. The default port number for the Flash Remoting service is 8500.

3. Create the gateway connection using the `NetServices.createGatewayConnection`function; for example:

   ```
   gatewayConnection = NetServices.createGatewayConnection();
   ```

© 2014 Adobe Systems Incorporated. All rights reserved.

Using server-side ActionScript functions

After you connect to the Flash Remoting service, you call functions that are defined in your server-side ActionScript files, and return results.

Call a function

1. Create an instance of the server-side ActionScript file using the `getService`function. This function instantiates the server-side ActionScript file as an object to use on the client side. For example:

   ```
   albumService = gatewayConnection.getService("recordsettest", this)
   ```

   Where `recordsettest` represents the name of the server-side ActionScript file, without the filename extension `.asr`.

2. Call a function defined in your server-side ActionScript object. Use dot notation to specify the object name followed by the function name; for example:

   ```
   albumService.getAlbum("The Color And The Shape", "1999");
   ```

   Where `albumService` is the instance of the server-side ActionScript file and `getAlbum` is a function that passes two arguments, `"The Color and The Shape"` and `"1999"`.

   > **Note**
   >
   > Arguments must occur in the order defined in the function declaration.

3. Handle the function results in ActionScript. See *Using the function results in ActionScript* below.

Using the function results in ActionScript

To use the results returned by server-side ActionScript, create a corresponding *results function*. The results function uses a special naming convention that ties it to the function that calls the server-side ActionScript. For example, if you defined a client-side ActionScript function called `basicCustomerQuery`, you also must create a results function called `basicCustomerQuery_Result`.
The results returned by server-side ActionScript functions differ somewhat depending on whether you are using `CF.http` or `CF.query`:

- The `CF.query` function returns a record set, which you manipulate using methods available in the RecordSet ActionScript class object. See *Using results returned by the CF.query function* section.
- The `CF.http` function returns simple text strings through properties that you reference in your server-side ActionScript. See *Using results returned by the CF.http function* section.

Using results returned by the CF.query function

You use functions in the RecordSet ActionScript object to access the data returned in a `CF.query` record set; for example, how many records are in the record set and the names of the columns. You can also use the RecordSet functions to pull the query data out of the record set. To do so, you reference a specific row number in the record set and use the `getItemAt` RecordSet function, as in the following example:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
// This function populates a Flash text box with data in the first row
// of the record set under the "email" column name.
function selectData_Result ( result )
{
 stringOutput.text = result.getItemAt(0)["email"];
 _root.employeesView.setDataProvider(result);
}
```

In the example, the column name is referenced in the `getItemAt` function between square brackets [ ]. (In ActionScript, indexes start at 0, so `getItemAt(0)` returns the first row.)For more information, see Using the CF.query function.

Using results returned by the CF.http function

The `CF.http` server-side ActionScript function returns data as simple text. You write server-side functions that reference the properties available in the object returned by the `CF.http` function. These properties store the file content of the retrieved file, HTTP status codes, the MIME type of the returned file, and so on. On the client side, you create return functions to handle data returned by the `CF.http` function. You write these functions to handle simple text data. For more information, see Using the CF.http function.

© 2014 Adobe Systems Incorporated. All rights reserved.

Global and request scope objects

Global and request scope objects are implicitly available in all server-side ActionScript. The following table describes these scope objects:

Scope name	Type	Description
config	Global	Initialization information for the server-side ActionScript adapter. Class: `javax.servlet.ServletConfig`
application	Global	The context for the current web application. The context defines methods that provide, for example, the MIME type of a file that can be used to write to a log file. There is one context per web application. Class: `javax.servlet.ServletContext`
request	Request	An object containing client request information. The object provides data, including parameter name and values, attributes, and an input stream. Class: `HttpServletRequest` (subtype of `javax.servlet.ServletRequest`)
response	Request	An object to assist in sending a response to the client. It provides HTTP-specific functionality in sending a response. Do not use the OutputStream or PrintWriter to send data back to the client. Class: `HttpServletResponse` (subtype of `javax.servlet.ServletResponse`)

For more information about these scope objects, see the documentation on the javax.servlet class at http://java.sun.com.

© 2014 Adobe Systems Incorporated. All rights reserved.

About the CF.query function and data sources

You use the `CF.query` function to populate SWF movie elements with data retrieved from a ColdFusion data source. To use the `CF.query` function you do the following:

Pull data into your SWF movie from a ColdFusion data source

1. Create a server-side ActionScript file that performs queries against a ColdFusion data source.
2. Write ActionScript code in your SWF movie that references your ActionScript file (.asr) on the ColdFusion server.
   You create server-side ActionScript to execute the query and return the data in a record set to the client your SWF movie. You can use methods in the RecordSet ActionScript object on the client to manipulate data in the record set and present data in your SWF movie.

> **Note**
>
> Client-side ActionScript files use the .as extension. Server-side ActionScript files use the .asr (ActionScript remote) extension.

Publishing dynamic data

You use the server-side ActionScript feature in ColdFusion to publish dynamic data. To do this, you write server-side ActionScript files that perform queries against ColdFusion data sources. Before using ActionScript, you must understand how to do the following:

- Create database queries in the server-side ActionScript file using the `CF.query` ActionScript function. See Using the CF.query function.
- Reference the server-side ActionScript file in your SWF movie. See Connecting to the Flash Remoting service.
  Using the `CF.query` function, you can do the following tasks:

- Create user login interfaces that validate users against a ColdFusion data source.
- Populate form elements and data grids with data from a ColdFusion data source.
- Create banners that pull data (such as URLs or image file paths) out of a database.
  The `CF.query` function can retrieve data from any supported ColdFusion data source (see *About ColdFusion data sources* below).

About ColdFusion data sources

For ColdFusion developers, the term *data source* can mean many different types of structured data accessible locally or across a network. You can query websites, Lightweight Directory Access Protocol (LDAP) servers, POP mail servers, and documents in a variety of formats. For server-side ActionScript, a data source ordinarily means the entry point to a ColdFusion database.
Your ColdFusion administrator can help you identify and configure data sources. To create ActionScript files that successfully perform queries on ColdFusion data sources, you must know how ColdFusion identifies the data source, as well as any other parameters that affect your ability to connect to that database, such as whether a user name and password are required to connect.
You use server-side ActionScript in ColdFusion to return record set data to a Flash client from a ColdFusion data source. You specify the ColdFusion data source name and the SQL statement you execute on the data source as arguments in the `CF.query` function in server-side ActionScript.
Typically, your server-side ActionScript handles the interaction with the ColdFusion data source, and returns a record set to the Flash client through the Flash Remoting service.
For more detailed information about ColdFusion data sources, see *Configuring and Administering ColdFusion*.

© 2014 Adobe Systems Incorporated. All rights reserved.

Using the CF.query function

You use the `CF.query` function in your server-side ActionScript to retrieve data from a ColdFusion data source. This function lets you perform queries against any ColdFusion data source.

> **Note**
>
> The `CF.query` function maps closely to the `cfquery` CFML tag, although it currently supports a subset of the `cfquery` attributes.

Use the `CF.query` function to do the following:

- Identify the data source you want to query.
- Pass SQL statements to the data source.
- Pass other optional parameters to the database.
  For reference information about the `CF.query` function, see CF.query in the *CFML Reference*.

About CF.query function syntax

You can write the `CF.query` ActionScript function using either named arguments or positional arguments. The named argument style is more readable, but it requires more code. Although the positional argument style supports a subset of `CF.query` arguments, it allows a more compact coding style that is more appropriate for simple expressions of the `CF.query` function.

Using CF.query named argument syntax

The `CF.query` function accepts the following named arguments:

```
// CF.query named argument syntax
CF.query
 ({
  datasource:"data source name",
  sql:"SQL stmts",
  username:"username",
  password:"password",
  maxrows:number,
  timeout:milliseconds
 })
```

> **Note**
>
> The named argument style requires curly brackets
> {} to surround the function arguments.

Using CF.query positional argument syntax

Positional arguments support a subset of `CF.query` arguments, and you can create more efficient code. The following is the syntax for the positional argument style:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
// CF.query positional argument syntax
CF.query(datasource, sql);
CF.query(datasource, sql, maxrows);
CF.query(datasource, sql, username, password);
CF.query(datasource, sql, username, password, maxrows);
```

Note

When using positional arguments, do not use curly braces {}.

About the CF.query record set

The `CF.query` function returns a RecordSet object, which is an instance of the RecordSet class of objects. The RecordSet class provides a wide range of functions for handling record set data.
You use methods in the RecordSet ActionScript class in your client-side ActionScript to change data returned in the `CF.query` record set.
Currently, the following methods are available in the RecordSet class:

Method	Description
addItem	Appends a record to the end of the specified RecordSet
addItemAt	Inserts a record at the specified index
addView	Requests notification of changes in a RecordSet object's state
filter	Creates a RecordSet object that contains selected records from the original RecordSet object
getColumnNames	Returns the names of all the columns of the RecordSet
getItemAt	Retrieves a record from a RecordSet object
getItemID	Gets the unique ID corresponding to a record
getLength	Returns the total number of records in a RecordSet object
getNumberAvailable	Returns the number of records that have been downloaded from the server
isFullyPopulated	Determines whether a RecordSet object can be edited or manipulated

© 2014 Adobe Systems Incorporated. All rights reserved.

isLocal	Determines whether a RecordSet object is local or server-associated
removeAll	Removes all records from the RecordSet object
removeItemAt	Removes a specified record
replaceItemAt	Replaces the entire contents of a record
setDeliveryMode	Changes the delivery mode of a server-associated record set
setField	Replaces one field of a record with a new value
sort	Sorts all records by a specified compare function
sortItemsBy	Sorts all the records by a selected field

These functions are available for every RecordSet object returned by the `CF.query` function to the Flash client. You run these functions as follows:

```
objectName.functionName();
```

For example, in the result function that you create to handle record set data returned by the `CF.query` function, you can reference the database column names returned in the record set using the `getColumnNames` RecordSet function:

```
function selectData_Result ( result )
{
 //result holds the query data; employeesView is a Flash list box
 stringOutput.text = result.getColumnNames();
 _root.employeesView.setDataProvider(result);
}
```

© 2014 Adobe Systems Incorporated. All rights reserved.

Building a simple application

The following procedure describes how to build a simple server-side ActionScript application. The example application, a corporate personnel directory, uses the NetServices object to connect to the `personneldirectory` server-side ActionScript. The `personneldirectory` server-side ActionScript retrieves data from a ColdFusion data source and returns the results to the SWF file as a RecordSet object.

> **Note**
>
> The server-side ActionScript application that you create provides the back-end services in an application.

This example requires the following:

- A server-side ActionScript file named personneldirectory.asr that includes functions that interact with a ColdFusion data source.
- A client-side SWF movie in which the NetServices object is created.

Create the application

1. Write server-side ActionScript that performs the database query and returns data to the client through the Flash Remoting service.
2. Create the SWF movie interface. See *Creating the SWF movie interface* below.
3. Define a search function that sends user data to the Flash Remoting service. See *Submitting user data to the Flash Remoting service* below.
4. Define a result function that captures the results returned from the Flash Remoting service. See *Capturing Flash Remoting service results* below.
5. Ensure that the SWF movie has established a connection to the Flash Remoting service. See *Checking for a Flash Remoting service connection* below.

Writing the server-side ActionScript function

The example here creates a search function that performs a simple search operation against a ColdFusion data source. This function accepts two arguments, firstName and lastName, and returns any records found that match these arguments.

Create a server-side ActionScript function

1. Create a server-side ActionScript file that contains the following code:

```
//search takes firstName lastName arguments
function search(firstName, lastName)
{
 searchdata = CF.query({datasource: "bigDSN",
  sql:"SELECT * from personnel WHERE fname = firstName AND lname =
lastName"{);

 if (searchdata)
  return searchdata;
 else
  return null;
}
```

Creating the SWF movie interface

© 2014 Adobe Systems Incorporated. All rights reserved.

The SWF movie interface example here consists of one frame with a variety of text boxes and a submit button.

1. In the Flash authoring environment, create a Flash source file, and save it as pDirectory.fla.
2. Create two input text boxes. Name one text box variable *lastName* and the other *firstName*.
3. Create a dynamic text box, and name its variable *status*.
4. Insert a list box component, and name it *dataView*.
5. Insert a push-button component.
6. Save your work.

Submitting user data to the Flash Remoting service

To send data to server-side ActionScript, create a function that passes the data from the SWF movie to server-side ActionScript. The `search` function, applied at the frame level, collects the user-entered data from the firstName and lastName text boxes and passes the data as function arguments to the directoryService object, which is created when the SWF movie connects to the Flash Remoting service. For more information, see *Checking for a Flash Remoting service connection* below.
The following is a Flash ActionScript example:

```
#include "NetServices.as"
function search()
{
 // The search() method is defined in the server-side AS file
 directoryService.search(firstName.text, lastName.text);
 dataView.setDataProvider(null);
 status.text = "waiting...";
}
```

Reviewing the code

The following table describes the code and its function:

Code	Description
`directoryService.search(firstName.text, lastName.text);`	Passes the contents of the firstName and lastName text boxes to server-side ActionScript.
`dataView.setDataProvider(null);`	Clears the `dataView` list box component.
`status.text = "waiting...";`	Displays a message in the status text box while the record set is being retrieved from server-side ActionScript.

Capturing Flash Remoting service results

When you create a function that calls a server-side ActionScript function, also create a function to handle the data returned by server-side ActionScript. Define the function with the same name as the function making the initial call, but you append `_Result` to the name.
For example, if you create a function called `basicQuery` to return query data, you define a results function to handle returned data; declare the results function as `basicQuery_Result`.
In the following example, the results function `search_Result` supplies the record set to the `dataView.setDataProvider` function:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
function search_Result(resultset)
{
 dataView.setDataProvider(resultset);
 status.text = (0+resultset.getLength())+" names found.";
}
```

Reviewing the code

The following table describes the code and its function:

Code	Description
`function search_Result(resultset)`	The `_Result` suffix tells the Flash Remoting service to return the results of the search function to this function.
`dataView.setDataProvider(resultset);`	Assigns the results returned by the Flash Remoting service to the `dataView` list box.
`status.text = (0+resultset.getLength())+" names found.";`	Displays the number of records returned by the Flash Remoting service.

Checking for a Flash Remoting service connection

To ensure that the SWF movie is connected to the Flash Remoting service, you use an `if` statement; for example:

```
if (inited == null)
{
 inited = true;
 NetServices.setDefaultGatewayUrl("http://localhost:8500/flashservices/
  gateway");
 gateway_conn = NetServices.createGatewayConnection();
 directoryService = gateway_conn.getService(personneldirectory, this);
 status.text = "Type into the text boxes, then click 'Search'";
}
```

In this example, the `inited` variable is evaluated for a value. If `inited` is `null` (not connected), the movie connects to the Flash Remoting service using the NetServices object. For more information about connecting to the Flash Remoting service, see Connecting to the Flash Remoting service.

© 2014 Adobe Systems Incorporated. All rights reserved.

About the CF.http function

You use the `CF.http` ActionScript function to retrieve information from a remote HTTP server using HTTP `Get` and `Post` methods, as follows:

- Using the `Get` method, you send information to the remote server directly in the URL. This method is common for a one-way transaction in which the `CF.http` function retrieves an object, such as the contents of a web page.
- The `Post` method can pass variables to a form or CGI program, and can also create HTTP cookies.
  The most basic way to use the `CF.http` function is to use it with the `Get` method argument to retrieve a page from a specified URL. The `Get` method is the default for the `CF.http` function.
  The following server-side example retrieves file content from the specified URL:

```
function basicGet(url)
{
 // Invoke with just the url argument. This is an HTTP GET.
 result = CF.http(url);
 return result.get("Filecontent");
}
```

The client-side example could look like the following:

```
#include "NetServices.as"
NetServices.setDefaultGatewayUrl("http://mycfserver:8500");
gatewayConnnection = NetServices.createGatewayConnection();
myHttp = gatewayConnnection.getService("httpFuncs", this);

// This is the server-side function invocation
url = "http://anyserver.com";
myHttp.basicGet(url);

// Create the results function
function basicGet_Result()
{
 url = "http://anyserver.com
 ssasFile.basicGet(url)
}
```

© 2014 Adobe Systems Incorporated. All rights reserved.

Using the CF.http function

The CF.http function returns an object that contains properties, also known as attributes. You reference these attributes to access the contents of the file returned, header information, HTTP status codes, and so on. The following table shows the available properties:

Property	Description
Text	A Boolean value indicating whether the specified URL location contains text data.
Charset	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
Header	Raw response header. The following is an example header:HTTP/1.1 200 OKDate: Mon, 04 Mar 2002 17:27:44 GMTServer: Apache/1.3.22 (Unix) mod_perl/1.26Set-Cookie: MM_cookie=207.22.48.162.4731015262864476; path=/; expires=Wed, 03-Mar-04 17:27:44 GMT; domain=.adobe.comConnection: closeContent-Type: text/html
Filecontent	File contents, for text and MIME files.
Mimetype	MIME type. Examples of MIME types include text/html, image/png, image/gif, video/mpeg, text/css, and audio/basic.
responseHeader	Response header. If there is one instance of a header key, this value can be accessed as a simple type. If there is more than one instance, values are put in an array in the responseHeader structure.
Statuscode	HTTP error code and associated error string. Common HTTP status codes returned in the response header include the following:400: Bad Request401: Unauthorized403: Forbidden404: Not Found405: Method Not Allowed

Referencing HTTP Post parameters in the CF.http function

To pass HTTP `Post` parameters in the `CF.http` function, construct an array of objects and assign this array to a variable named `params`. The following arguments can only be passed as an array of objects in the `params` argume nt of the `CF.http` function:

© 2014 Adobe Systems Incorporated. All rights reserved.

Parameter	Description
name	The variable name for data that is passed
type	Transaction type: • URL • FormField • Cookie • CGI • File
value	Value of URL, FormField, Cookie, File, or CGI variables that are passed

In the following example, the `CF.http` function passes HTTP `Post` parameters in an array of objects:

```
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:8500/";

 // Invoke with the method, url, params, username, and password
 result = CF.http("post", url, params, "karl", "salsa");
 return result.get("Filecontent");
}
```

Using the CF.http Post method

You use the `Post` method to send cookie, form field, CGI, URL, and file variables to a specified ColdFusion page or CGI program for processing. For POST operations, use the `params` argument for each variable that you post. The `Post` method passes data to a specified ColdFusion page or an executable that interprets the variables being sent, and returns data.
For example, when you build an HTML form using the `Post` method, you specify the name of the page to which form data is passed. You use the `Post` method in the `CF.http` function in a similar way. However, with the `CF.http` function, the page that receives the Post does not display anything. See the following example:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
function postWithParams()
{
 // Set up the array of Post parameters. These are just like cfhttpparam tags.
 // This example passes formfield data to a specified URL.
 params = new Array();
 params[1] = {name:"Formfield1", type:"FormField", value:"George"};
 params[2] = [name:"Formfield2", type:"FormField", value:"Brown"};

 url = "http://localhost:8500/";

 // Invoke CF.http with the method, url, and params
 result = CF.http("post", url, params);
 return result.get("Filecontent");
}
```

Using the CF.http Get method

You use the `Get` method to retrieve files, including text and binary files, from a specified server. You reference properties of the object returned by the `CF.http` function to access things like file content, header information, MIME type, and so on.
The following example uses the `CF.http` function to show a common approach to retrieving data from the web:

```
// Returns content of URL defined in url variable
// This example uses positional argument style
function get()
{
 url = "http://www.adobe.com/software/coldfusion/";

 //Invoke with just the url argument. Get is the default.
 result = CF.http(url);
 return result.get("Filecontent");
}
```

For more information about `CF.http` function properties, see CF.http in the *CFML Reference*.

- Referencing HTTP Post parameters in the CF.http function
- Using the CF.http Post method
- Using the CF.http Get method

© 2014 Adobe Systems Incorporated. All rights reserved.

Requesting and Presenting Information

Adobe ColdFusion lets you request and present information through multiple formats.

- Introduction to Retrieving and Formatting Data
- Building Dynamic Forms with cfform Tags
- Validating Data-Developing guide
- Creating Forms in Flash
- Creating Skinnable XML Forms
- Using Ajax User Interface Components and Features
- Using Ajax Data and Development Features

#back to top

© 2014 Adobe Systems Incorporated. All rights reserved.

Introduction to Retrieving and Formatting Data

Adobe ColdFusion lets you retrieve and format data. You can use forms to get user data and control the data that a dynamic web page displays. You can also populate a table with query results and use ColdFusion functions to format and manipulate data. To use these features, you should be familiar with HTML forms.

© 2014 Adobe Systems Incorporated. All rights reserved.

Using forms in ColdFusion

ColdFusion lets you use a variety of types of forms. You can use plain HTML or CFML, and you can generate HTML, Flash, or skinned XML forms.

ColdFusion forms tags

You can use HTML or CFML tags to define your form. ColdFusion includes the following CFML tags that correspond to HTML tags, but provide additional functionality:

- cfapplet
- cfform
- cfinput
- cfselect
- cftextarea
  These tags support all the attributes of their HTML counterparts, plus ColdFusion attributes and features. ColdFusion also provides the following forms tags that have no direct equivalent in HTML:

- cfcalendar Lets users select dates from a Flash month-by-month calendar.
- cfgrid Displays and lets users enter data in a row and column grid format; can get data directly from a query.
- cfslider Lets users input data by moving a sliding marker.
- cftree Displays data in a hierarchical tree format with graphical indicators; can get data directly from a query.

ColdFusion Form tag features

ColdFusion forms tags provide the following features:

- **Built-in validation support** You can validate data in the client browser or on the server. You can specify that a field is required, contains a specific type of data, has a maximum length, or is in a range of values. You can also use data masking to control user input. For more information on validation, see Validating data.

  > **Note**
  >
  > ColdFusion also provides a method of doing on-server validation of HTML form controls.

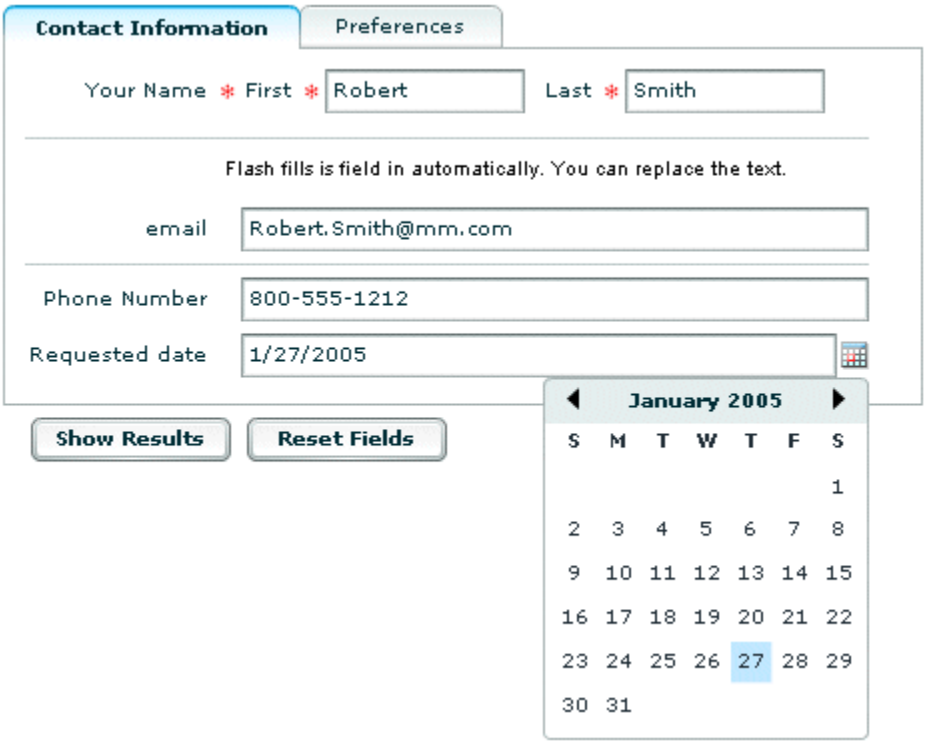
- **Flash format forms and elements** You can display a form as Flash, which works identically on a variety of platforms and provides additional display features not available in HTML. These features include accordion-style and multiple-tab form panes and automatic element positioning. You can also display `cftree` , `cfgrid`, and `cfcalendar` form elements as Flash items in an otherwise-HTML form. For more information on Flash forms and form elements, see Creating Forms in Flash.
- **XML Skinable forms** ColdFusion can generate XML forms and apply XSLT skins to format the forms. XML format forms let you separate the form presentation from the form logic and data field information. They give you detailed control over the appearance of the forms by applying custom skins, and let you create custom controls. For more information on XML skinnable forms, see Creating Skinnable XML Forms.
- **Direct support for ColdFusion variables** You can easily use ColdFusion variables directly to populate your form controls. For example, you can specify a query result to populate the `cfgrid` and `cftree` tags.These features make CFML forms tags powerful and flexible, and let you easily develop fully featured, pleasing forms. The CFML tags used here, do not describe or use most of their special features. SeeBuilding Dynamic Forms with cfform Tags for information on how to use many of the tags that are specific to ColdFusion, such as `cftree` and `cfgrid`.

Creating a basic form

The following simple form shows how you can create a form that lets a user enter data. This form uses basic CFML form tags. It does not use any of the advanced features of ColdFusion, such as validation, Flash or XML format, or

© 2014 Adobe Systems Incorporated. All rights reserved.

special input controls. You could convert it to a purely HTML form by removing the initial "cf" prefix from the tag names, and the form would work.



The following table shows the format of form control tags:

Control	Code
Text control	`<cfinput type="Text" name="ControlName" size="Value" maxlength="Value">`

List (select) box	`<cfselect name="ControlName">`
	`<option value="Value1">DisplayName1`
	`<option value="Value2">DisplayName2`
	`<option value="Value3">DisplayName3`
	`</cfselect>`

© 2014 Adobe Systems Incorporated. All rights reserved.

Radio buttons	`<cfinput type="Radio" name="ControlName" value="Value1">DisplayName1 <cfinput type="Radio" name="ControlName" value="Value2">DisplayName2 <cfinput type="Radio" name="ControlName" value="Value3">DisplayName3`
Check box	`<cfinput type="Checkbox" name="ControlName" value="Yes|No">Yes`
Reset button	`<cfinput type="Reset" name="ControlName" value="DisplayName">`
Submit button	`<cfinput type="Submit" name="ControlName" value="DisplayName">`

The following listing shows the form source in detail. To test the form and use it as input for later examples, save this code as formpage.cfm.

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<html>
<head>
<title>Input form</title>
</head>
<body>
<!--- Specify the action page in the form tag. The form variables will
  pass to this page when the form is submitted. --->

<cfform action="actionpage.cfm" method="post">

<!--- Text box. --->
<p>
First Name: <cfinput type="Text" name="FirstName" size="20"maxlength="35"><br>
Last Name: <cfinput type="Text" name="LastName" size="20" maxlength="35"><br>
Salary: <cfinput type="Text" name="Salary" size="10" maxlength="10">
</p>

<!--- List box. --->
<p>
City
<cfselect name="City">
 <option value="Arlington">Arlington
 <option value="Boston">Boston
 <option value="Cambridge">Cambridge
 <option value="Minneapolis">Minneapolis
 <option value="Seattle">Seattle
</cfselect>
</p>

<!--- Radio buttons. --->
<p>
Department:<br>
<cfinput type="radio" name="Department" value="Training">Training<br>
<cfinput type="radio" name="Department" value="Sales">Sales<br>
<input type="radio" name="Department"
 value="Marketing">Marketing<br>
</p>

<!--- Check box. --->
<p>
Contractor? <cfinput type="checkbox" name="Contractor"
 value="Yes" checked>Yes
</p>

<!--- Reset button. --->
<cfinput type="Reset" name="ResetForm" value="Clear Form">
<!--- submit button --->
<cfinput type="Submit" name="SubmitForm" value="Submit">

</cfform>
</body>
</html>
```

Forms guidelines

© 2014 Adobe Systems Incorporated. All rights reserved.

When using forms, keep in mind the following guidelines:

- To make the coding process easy to follow, name form controls the same as target database fields. For example, if a text control corresponds to a data source FirstName field, use FirstName as the control name.
- For ease of use, limit radio buttons to between three and five mutually exclusive options. If you need more options, consider a drop-down list.
- Use list boxes to allow the user to choose from many options or to choose multiple items from a list.
- Check boxes, radio buttons, and list boxes do not pass data to action pages unless they are selected on a form. If you try to reference these variables on the action page, you receive an error if they are not present. For information on how to determine whether a variable exists on the action page, see *Testing for a variables existence*.
- You can dynamically populate drop-down lists using query data. For more information, see Dynamically populating list boxes.

© 2014 Adobe Systems Incorporated. All rights reserved.

Working with action pages

When the user submits a form, ColdFusion runs the action page specified by the `cfform` or `form` tag `action` attribute. A ColdFusion action page is like any other application page, except that you can use the form variables that are passed to it from an associated form.

Processing form variables on action pages

The action page gets a form variable for every form control that contains a value when the form is submitted.

> **Note**
>
> If multiple controls have the same name, one form variable is passed to the action page with a comma-delimited list of values.

A form variable's name is the name that you assigned to the form control on the form page. Refer to the form variable by name within tags, functions, and other expressions on an action page.
On the action page, the form variables are in the Form scope, prefix them with "Form." to explicitly tell ColdFusion that you are referring to a form variable. For example, the following code references the LastName form variable for output on an action page:

```
<cfoutput>
 #Form.LastName#
</cfoutput>
```

The Form scope also contains a list variable called `Form.fieldnames`. It contains a list of all form variables submitted to the action page. If no form variables are passed to the action page, ColdFusion does not create the `Form.fieldnames` list.

Using form data to generate SQL statements

As described in previous chapters, you can retrieve a record for every employee in a database table by composing a query like the following:

```
<cfquery name="GetEmployees" datasource="cfdocexamples">
 SELECTFirstName, LastName, Contract
 FROM Employee
</cfquery>
```

When you want to return information about employees that matches user search criteria, you use the SQL WHERE clause with a SQL SELECT statement. When the WHERE clause is processed, it filters the query data based on the results of the comparison.
For example, to return employee data for only employees with the last name of Smith, you build a query that looks like the following:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cfquery name="GetEmployees" datasource="cfdocexamples">
 SELECT FirstName, LastName, Contract
 FROM Employee
 WHERE LastName = 'Smith'
</cfquery>
```

However, instead of placing the LastName directly in the SQL WHERE clause, you can use the text that the user entered in the form for comparison:

```
<cfquery name="GetEmployees" datasource="cfdocexamples">
 SELECT FirstName, LastName, Salary
 FROM Employee
 WHERE LastName=<cfqueryparam value="#Form.LastName#"
 CFSQLType="CF_SQL_VARCHAR">
</cfquery>
```

For security, this example encapsulates the form variable within the `cfqueryparam` tag to ensure that the user passed a valid string value for the LastName. For more information on using the `cfqueryparam` tag with queries and on dynamic SQL, see Accessing and Retrieving Data.

Creating action pages

Use the following procedure to create an action page for the formpage.cfm page that you created in the previous example.

Create an action page for the form

1. Create a ColdFusion page with the following content:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<html>
<head>
<title>Retrieving Employee Data Based on Criteria from Form</title>
</head>

<body>
<cfquery name="GetEmployees" datasource="cfdocexamples">
 SELECT FirstName, LastName, Salary
 FROM Employee
 WHERE LastName=<cfqueryparam value="#Form.LastName#"
 CFSQLType="CF_SQL_VARCHAR">
</cfquery>
<h4>Employee Data Based on Criteria from Form</h4>
<cfoutput query="GetEmployees">
#FirstName#
#LastName#
#Salary#<br>
</cfoutput>
<br>
<cfoutput>Contractor: #Form.Contractor#</cfoutput>
</body>
</html>
```

2. Save the page as actionpage.cfm in the myapps directory.
3. View the formpage.cfm page in your browser.
4. Enter data, for example, Smith, in the Last Name box and submit the form.The browser displays a line with the first and last name and salary for each entry in the database that match the name you typed, followed by a line with the text "Contractor: Yes".
5. Click Back in your browser to redisplay the form.
6. Remove the check mark from the check box and submit the form again. This time an error occurs because the check box does not pass a variable to the action page. For information on modifying the actionpage.cfm page to fix the error, see *Testing for a variable's existence* below.

Reviewing the code

The following table describes the highlighted code and its function:

Code	Description
`<cfquery name="GetEmployees"datasource="cfdocexamples">`	Queries the data source cfdocexamples and names the query GetEmployees.

© 2014 Adobe Systems Incorporated. All rights reserved.

`SELECT FirstName, LastName, Salary`
 	
`FROM Employee`
`WHERE LastName=<cfqueryparam`
`value="#Form.LastName#"`
`CFSQLType="CF_SQL_VARCHAR">`
	Retrieves the FirstName, LastName, and Salary fields from the Employee table, but only if the value of the LastName field matches what the user entered in the LastName text box in the form on formpage.cfm.

`<cfoutput query="GetEmployees">`	Displays results of the GetEmployees query.
`#FirstName#` `#LastName#` `#Salary# `	Displays the value of the FirstName, LastName, and Salary fields for a record, starting with the first record, then goes to the next line. Keeps displaying the records that match the criteria you specified in the SELECT statement, followed by a line break, until you run out of records.
`</cfoutput>`	Closes the cfoutput block.
` ` `<cfoutput>Contractor:` `#Form.Contractor#` `</cfoutput>`	Displays a blank line followed by the text "Contractor": and the value of the form Contractor check box. A more complete example would test to ensure the existence of the variable and would use the variable in the query.

Testing for a variable's existence

Before relying on a variable's existence in an application page, you can test to see if it exists using the ColdFusion `IsDefined` function. A *function* is a named procedure that takes input and operates on it. For example, the `IsDefined` function determines whether a variable exists. CFML provides a large number of functions, which are documented in the *CFML Reference*.

The following code prevents the error in the previous example by checking to see whether the Contractor Form variable exists before using it:

```
<cfif IsDefined("Form.Contractor")>
 <cfoutput>Contractor: #Form.Contractor#</cfoutput>
</cfif>
```

The argument passed to the `IsDefined` function must always be enclosed in double-quotation marks. For more information on the `IsDefined` function, see the *CFML Reference*.

If you attempt to evaluate a variable that you did not define, ColdFusion cannot process the page and displays an

© 2014 Adobe Systems Incorporated. All rights reserved.

error message. To help diagnose such problems, turn on debugging in the ColdFusion Administrator. The Administrator debugging information shows which variables are being passed to your application pages.

Requiring users to enter values in form fields

One of the limitations of HTML forms is the inability to define input fields as required. Because this is an important requirement for database applications, ColdFusion lets you require users to enter data in fields. To specify a field as required, you can do either of the following:

- Use the `required` attribute of the `cfinput`, `cfselect`, `cftextarea`, and `cftree` tags.
- Use a hidden field that has a `name` attribute composed of the field name and the suffix `_required`. You can use this technique with CFML and HTML form tags.
  For example, to require that the user enter a value in the FirstName field of a `cfinput` tag, use the following syntax:

```
<cfinput type="Text" name="FirstName" size="20" maxlength="35" required="Yes">
```

To require that the user enters a value in the FirstName field of an HTML `input` tag, use the following syntax:

```
<input type="Text" name="FirstName" size="20" maxlength="35">
<input type="hidden" name="FirstName_required">
```

In either of these examples, if the user leaves the FirstName field empty, ColdFusion rejects the form submittal and returns a message informing the user that the field is required. You can customize the contents of this error message.
If you use a `required` attribute, you customize the message by using the `message` attribute, as follows:

```
<cfinput type="Text" name="FirstName" size="20" maxlength="35" required="Yes"
 message="You must enter your first name.">
```

If you use a hidden field tag, you customize the message using the `value` attribute of the hidden field, as follows:

```
<input type="hidden" name="FirstName_required"
 value="You must enter your first name.">
```

Form variable notes and considerations

When using form variables in an action page, keep in mind the following guidelines:

- A form variable is available on the action page and pages that it includes.
- Prefix form variables with "Form."__when referencing them on the action page.
- Surround variable values with number signs (#) for output.
- Variables for check boxes, radio buttons, and list boxes with `size` attributes greater than 1 only get passed to the action page if you select an option. Text boxes, passwords, and text area fields pass an empty string if you do not enter text.
- An error occurs if the action page tries to use a variable that was not passed.

© 2014 Adobe Systems Incorporated. All rights reserved.

- If multiple controls have the same name, one form variable is passed to the action page with a comma-delimited list of values.
- You can validate form variable values on the client or the server.

© 2014 Adobe Systems Incorporated. All rights reserved.

Working with queries and data

The ability to generate and display query data is one of the most important and flexible features of ColdFusion. Some of these tools are effective for presenting any data, not just query results.

Using HTML tables to display query results

You can use HTML tables to specify how the results of a query appear on a page. To do so, you place the `cfoutput` tag *inside* the table tags. You can also use the HTML `th` tag to place column labels in a header row. To create a row in the table for each row in the query results, place the `tr` block inside the `cfoutput` tag.
In addition, you can use CFML functions to format individual pieces of data, such as dates and numeric values.

Place the query results in a table

1. Open the ColdFusion actionpage.cfm page in your editor.
2. Modify the page so that it appears as follows:

```
<html>
<head>
<title>Retrieving Employee Data Based on Criteria from Form</title>
</head>

<body>
<cfquery name="GetEmployees" datasource="cfdocexamples">
 SELECT FirstName, LastName, Salary
 FROM Employee
 WHERE LastName=<cfqueryparam value="#Form.LastName#"
 CFSQLType="CF_SQL_VARCHAR">
</cfquery>
<h4>Employee Data Based on Criteria from Form</h4>
<table>
<tr>
<th>First Name</th>
<th>Last Name</th>
<th>Salary</th>
</tr>
<cfoutput query="GetEmployees">
<tr>
<td>#FirstName#</td>
<td>#LastName#</td>
<td>#Salary#</td>
</tr>
</cfoutput>
</table>
<br>
<cfif IsDefined("Form.Contractor")>
 <cfoutput>Contractor: #Form.Contractor#</cfoutput>
</cfif>
</body>
</html>
```

3. Save the page as actionpage.cfm in the myapps directory.
4. View the formpage.cfm page in your browser.
5. Enter Smith in the Last Name text box and submit the form.The records that match the criteria specified in the form appear in a table.

© 2014 Adobe Systems Incorporated. All rights reserved.

Reviewing the code

The following table describes the highlighted code and its function:

Code	Description
`<table>`	Places data into a table.

`<tr>`
`<th>First Name</th>`
`<th>Last Name</th>`
`<th>Salary</th>`
`</tr>`
	In the first row of the table, includes three columns, with the headings: First Name, Last Name, and Salary.

`<cfoutput query="GetEmployees">`	Tells ColdFusion to display the results of the GetEmployees query.

`<tr>`
`<td>#FirstName#</td>`
`<td>#LastName#</td>`
`<td>#Salary#</td>`
`</tr>`
	For each record in the query, creates a row in the table, with three columns that display the values of the FirstName, LastName, and Salary fields of the record.

`</cfoutput>`	Ends the output region.
`</table>`	Ends the table.

Formatting individual data items

You can format individual data items. For example, you can format the salary data as monetary values. To format the salary data using the dollar format, you use the CFML function `DollarFormat`.

Change the format of the Salary

1. Open the file actionpage.cfm in your editor.
2. Change the following line:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<td>#Salary#</td>
```

to

```
<td>#DollarFormat(Salary)#</td>
```

3. Save the page.

Building flexible search interfaces

One option with forms is to build a search based on the form data. For example, you could use form data as part of the WHERE clause to construct a database query.
To give users the option to enter multiple search criteria in a form, you can wrap conditional logic around a SQL AND clause as part of the WHERE clause. The following action page allows users to search for employees by department, last name, or both.

Build a more flexible search interface

1. Open the ColdFusion actionpage.cfm page in your editor.
2. Modify the page so that it appears as follows:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<html>
<head>
<title>Retrieving Employee Data Based on Criteria from Form</title>
</head>
<body>
<cfquery name="GetEmployees" datasource="cfdocexamples">
 SELECT Departmt.Dept_Name,
  Employee.FirstName,
  Employee.LastName,
  Employee.StartDate,
  Employee.Salary
 FROM Departmt, Employee
 WHERE Departmt.Dept_ID = Employee.Dept_ID
 <cfif IsDefined("Form.Department")>
 AND Departmt.Dept_Name=<cfqueryparam value="#Form.Department#"
    CFSQLType="CF_SQL_VARCHAR">
 </cfif>
 <cfif Form.LastName IS NOT "">
  AND Employee.LastName=<cfqueryparam value="#Form.LastName#"
    CFSQLType="CF_SQL_VARCHAR">
 </cfif>
</cfquery>

<h4>Employee Data Based on Criteria from Form</h4>
<table>
<tr>
<th>First Name</th>
<th>Last Name</th>
<th>Salary</th>
</tr>
<cfoutput query="GetEmployees">
<tr>
<td>#FirstName#</td>
<td>#LastName#</td>
<td>#Salary#</td>
</tr>
</cfoutput>
</table>
</body>
</html>
```

3. Save the file.
4. View the formpage.cfm page in your browser.
5. Select a department, optionally enter a last name, and submit the form.

Reviewing the code

The following table describes the highlighted code and its function:

Code	Description

© 2014 Adobe Systems Incorporated. All rights reserved.

`SELECT Departmt.Dept_Name,`
` Employee.FirstName,`
` Employee.LastName,`
` Employee.StartDate,`
` Employee.Salary`
`FROM Departmt, Employee`
`WHERE Departmt.Dept_ID =`
`Employee.Dept_ID`
	Retrieves the fields listed from the Departmt and Employee tables, joining the tables based on the Dept_ID field in each table.

`<cfif`
`IsDefined("Form.Department")>`
` AND`
`Departmt.Dept_Name=<cfqueryparam`
`value="#Form.Department#"`
`  CFSQLType="CF_SQL_VARCHAR">`
` </cfif>`
	If the user specified a department on the form, only retrieves records where the department name is the same as the one that the user specified. Use number signs (#) in the SQL AND statement to identify Form.Department as a ColdFusion variable, but not in the IsDefined function.

`<cfif Form.LastName IS NOT "">`
`  AND`
`Employee.LastName=<cfqueryparam`
`value="#Form.LastName#"`
`  CFSQLType="CF_SQL_VARCHAR">`
` </cfif>`
	If the user specified a last name in the form, only retrieves the records in which the last name is the same as the one that the user entered in the form.

© 2014 Adobe Systems Incorporated. All rights reserved.

Returning results to the user

When you return your results to the user, ensure that your pages respond to the user's needs and are appropriate for the type and amount of information. In particular, consider the following situations:

- When there are no query results
- When you return partial results

Handling no query results

Your code must accommodate the cases in which a query does not return any records. To determine whether a search has retrieved records, use the `RecordCount` query variable. You can use the variable in a conditional logic expression that determines how to display search results appropriately to users.

> **Note**
>
> For more information on query variables, including `RecordCount`, see Accessing and Retrieving Data.

For example, to inform the user when no records are found by the GetEmployees query, insert the following code before displaying the data:

```
<cfif GetEmployees.RecordCount IS "0">
 No records match your search criteria. <BR>
<cfelse>
```

Do the following:

- Prefix `RecordCount` with the query name.
- Add a procedure after the `cfif` tag that displays a message to the user.
- Add a procedure after the `cfelse` tag to format the returned data.
- Follow the second procedure with a `</cfif>` tag end to indicate the end of the conditional code.

Return search results to users

1. Edit the actionpage.cfm page.
2. Change the page so that it appears as follows:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<html>
<head>
<title>Retrieving Employee Data Based on Criteria from Form</title>
</head>

<body>
<cfquery name="GetEmployees" datasource="cfdocexamples">
 SELECT Departmt.Dept_Name,
  Employee.FirstName,
  Employee.LastName,
  Employee.StartDate,
  Employee.Salary
 FROM Departmt, Employee
 WHERE Departmt.Dept_ID = Employee.Dept_ID
 <cfif isdefined("Form.Department")>
  AND Departmt.Dept_Name = <cfqueryparam value="#Form.Department#"
   CFSQLType="CF_SQL_VARCHAR">
 </cfif>
 <cfif Form.LastName is not "">
  AND Employee.LastName = <cfqueryparam value="#Form.LastName#"
   CFSQLType="CF_SQL_VARCHAR">
 </cfif>
</cfquery>

<cfif GetEmployees.recordcount is "0">
No records match your search criteria. <br>
Please go back to the form and try again.
<cfelse>
<h4>Employee Data Based on Criteria from Form</h4>
<table>
<tr>
<th>First Name</th>
<th>Last Name</th>
<th>Salary</th>
</tr>
<cfoutput query="GetEmployees">
<tr>
<td>#FirstName#</td>
<td>#LastName#</td>
<td>#Salary#</td>
</tr>
</cfoutput>
</cfif>
</table>
</body>
</html>
```

3. Save the file.
4. Return to the form, enter search criteria, and submit the form.
5. If no records match the criteria you specified, the message appears.

Returning results incrementally

You can use the `cfflush` tag to incrementally display long-running requests to the browser before a ColdFusion

© 2014 Adobe Systems Incorporated. All rights reserved.

page is fully processed. This tag lets you give the user quick feedback when it takes a long time to complete processing a request. For example, when a request takes time to return results, you can use the `cfflush` tag to display the message, "Processing your request – please wait." You can also use it to incrementally display a long list as it gets retrieved.

The first time you use the `cfflush` tag on a page, it sends to the browser all of the HTML headers and any other available HTML. Subsequent `cfflush` tags on the page send only the output that ColdFusion generated after the previous flush.

You can specify an `interval` attribute to tell ColdFusion to flush the output each time that at least the specified number of bytes become available. (The count does not include HTML headers and any data that is already available when you make this call.) You can use the `cfflush` tag in a `cfloop` tag to incrementally flush data as it becomes available. This format is useful when a query responds slowly with large amounts of data.

When you flush data, make sure that a sufficient amount of information is available, because some browsers do not respond if you flush only a small amount. Similarly, if you use an `interval` attribute, set it for a reasonable size, such as a few hundred bytes or more, but not many thousands of bytes.

Limitations of the cfflush tag: 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`, `cfform`, `cfheader`, `cfhtmlhead`, `cflocation`, and `SetLocale`. (These tags and functions normally modify the HTML header, but cannot do so after a `cfflush` tag, because the `cfflush` tag 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`, `cfqueryparam`, and custom tags, can 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.
- You can catch `cfflush` errors, except Cookie errors, with a `cfcatch` tag. Catch cookie errors with a `cfcatch type="Any"` tag.

Example: using the cfloop tag and Rand function

The following example uses the `cfloop` tag and the `Rand` random number generating function to artificially delay the generation of data for display. It simulates a situation in which it takes time to retrieve the first data and additional information becomes available slowly.

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<html>
<head>
 <title>Your Magic numbers</title>
</head>

<body>
<h1>Your Magic numbers</h1>
<P>It will take us a little while to calculate your ten magic numbers.
It takes a lot of work to find numbers that truly fit your personality.
So relax for a minute or so while we do the hard work for you.</P>
<h2>We are sure you will agree it was worth the short wait!</h2>
<cfflush>

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

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

Reviewing the code

The following table describes the code and its function:

Code	Description
`<h2>We are sure you will agree it was worth the short wait!</h2>` `<cfflush>`	Sends the HTML header and all HTML output to the cfflush tag to the user. This displays the explanatory paragraph and H2 tag contents.
`<cfflush interval=10>`	Flushes additional data to the user whenever at least 10 bytes are available.

© 2014 Adobe Systems Incorporated. All rights reserved.

`<cfloop index="randomindex" from="1" to="200000" step="1">`
`<cfset random=Rand()>`
`</cfloop>`
	Inserts an artificial delay by using the Rand function to calculate many random numbers.

`<cfloop index="Myindex" from="1" to="10" step="1">` `<cfloop index="randomindex" from="1" to="100000" step="1">` `<cfset random=rand()>` `</cfloop>` `<cfoutput>` `Magic number #Myindex# is:  #RandRange (100000,999999)#  ` `</cfoutput>` `</cfloop>`	Generates and displays 10 random numbers. This code uses two loops. The outer loop repeats ten times, once for each number to display. The inner loop uses the `Rand` function to create another delay by generating more (unused) random numbers. It then calls the RandRange function to generate a six-digit random number for display.

© 2014 Adobe Systems Incorporated. All rights reserved.

Dynamically populating list boxes

The code in Creating a basic form hard-coded the form's list box options. Instead of manually entering the information on a form, you can dynamically populate a list box with database fields. When you write code this way, the form page automatically reflects the changes that you make to the database.
You use two tags to dynamically populate a list box:

- Use the `cfquery` tag to retrieve the column data from a database table.
- Use the `cfselect` tag with the `query` attribute to dynamically populate the `options` of this form control.

Dynamically populate a list box

1. Open the formpage.cfm page.
2. Modify the file so that it appears as follows:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<html>
<head>
<title>Input form</title>
</head>
<body>
<cfquery name="GetDepartments" datasource="cfdocexamples">
 SELECT DISTINCT Location
 FROM Departmt
</cfquery>

<!--- Define the action page in the form tag.
 The form variables pass to this page
 when the form is submitted --->

<cfform action="actionpage.cfm" method="post">

<!--- Text box. --->
<p>
First Name: <cfinput type="Text" name="FirstName" size="20"
maxlength="35"><br>
Last Name: <cfinput type="Text" name="LastName" size="20" maxlength="35"><br>
Salary: <cfinput type="Text" name="Salary" size="10" maxlength="10">
</p>

<!--- List box. --->
City
<cfset optsize=getDepartments.recordcount + 1>
<cfselect name="City" query="GetDepartments" value="Location"
size="#optsize#">
 <option value="">Select All
</cfselect>

<!--- Radio buttons. --->
<p>
Department:<br>
<cfinput type="radio" name="Department" value="Training">Training<br>
<cfinput type="radio" name="Department" value="Sales">Sales<br>
<cfinput type="radio" name="Department" value="Marketing">Marketing<br>
<cfinput type="radio" name="Department" value="HR">HR<br>
</p>

<!--- Check box. --->
<p>
Contractor? <cfinput type="checkbox" name="Contractor" value="Yes" checked>Yes

</p>

<!--- Reset button. --->
<cfinput type="reset" name="ResetForm" value="Clear Form">

<!--- Submit button. --->
<cfinput type="submit" name="SubmitForm" value="Submit">
</cfform>
</body>
</html>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

3. Save the page as formpage.cfm.
4. View the formpage.cfm page in a browser. The changes that you just made appear in the form.Remember that you need an action page to submit values.

Reviewing the code

The following table describes the highlighted code and its function:

Code	Description

`<cfquery name="GetDepartments" datasource="cfdocexamples">`
`SELECT DISTINCT Location`
`FROM Departmt`
`</cfquery>`
	Gets the locations of all departments in the Departmt table. The DISTINCT clause eliminates duplicate location names from the returned query results.

`<cfset optsize=getDepartments.recordcount + 1>`	Sets the optsize variable to the number of entries to add dynamically to the selection list, plus one for the manually coded Select All option.

`<cfselect name="City" query="GetDepartments" value="Location" size="#optsize#">`
 	
`<option value="">Select All`
`</cfselect>`
	Populates the City selection list from the Location column of the GetDepartments query. The control has one option for each row returned by the query.Adds an option that allows users to select all locations. If the user selects this option, the form value is an empty string. The action page must check for the empty string and handle it appropriately.

© 2014 Adobe Systems Incorporated. All rights reserved.

Creating dynamic check boxes and multiple-selection list boxes

When an HTML or CFML form contains a list of check boxes with the same name or a multiple-selection list box (that is, a box in which users can select multiple items from the list), the user's entries are made available as a comma-delimited list with the selected values. These lists can be useful for a wide range of input types.

> **Note**
>
> If the user does not select a check box or make a selection from a list box, no variable is created. The `cfinput` and `cfupdate` tags do not work correctly if there are no values. To prevent errors, make the form fields required, use dynamic SQL, or use the `cfparam` tag to set a default value for the form field.

Check boxes

When you place a series of check boxes with the same name in a form, the variable that is created contains a comma-delimited list of values. The values can be either numeric values or alphanumeric strings. These two types of values are treated slightly differently.

Handling numeric values

Suppose you want a user to select one or more departments using check boxes. You then query the database to retrieve detailed information on the selected departments. The code for a simple set of check boxes that lets the user select departments looks like the following:

```
<cfinput type="checkbox"
 name="SelectedDepts"
 value="1">
 Training<br>

<cfinput type="checkbox"
 name="SelectedDepts"
 value="2">
 Marketing<br>

<cfinput type="checkbox"
 name="SelectedDepts"
 value="3">
 HR<br>

<cfinput type="checkbox"
 name="SelectedDepts"
 value="4">
 Sales<br>
</html>
```

The user sees the name of the department, but the `value` attribute of each check box is a number that corresponds to the underlying database primary key for the department's record.
If the user checks the Marketing and Sales items, the value of the SelectedDepts form field is 2,4 and you use the SelectedDepts value in the following SQL statement:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
SELECT *
 FROM Departmt
 WHERE Dept_ID IN ( #Form.SelectedDepts# )
```

The ColdFusion server sends the following statement to the database:

```
SELECT *
 FROM Departmt
 WHERE Dept_ID IN ( 2,4 )
```

Handling string values

To search for a database field that contains string values (instead of numeric), modify the `checkbox` and `cfquery` syntax to make sure that the string values are sent to the data source in single-quotation marks (').
The first example searched for department information based on a numeric primary key field called Dept_ID. Suppose, instead, that the primary key is a database field called Dept_Name that contains string values. In that case, your code for check boxes should look like the following:

```
<cfinput type="checkbox"
 name="SelectedDepts"
 value="Training">
 Training<br>

<cfinput type="checkbox"
 name="SelectedDepts"
 value="Marketing">
 Marketing<br>

<cfinput type="checkbox"
 name="SelectedDepts"
 value="HR">
 HR<br>

<cfinput type="checkbox"
 name="SelectedDepts"
 value="Sales">
 Sales<br>
```

If the user checked Marketing and Sales, the value of the SelectedDepts form field would be the list Marketing,Sales and you use the following SQL statement:

```
SELECT *
 FROM Departmt
 WHERE Dept_Name IN
 (#ListQualify(Form.SelectedDepts,"'")#)
```

© 2014 Adobe Systems Incorporated. All rights reserved.

In SQL, all strings must be surrounded in single-quotation marks. The `ListQualify` function returns a list with the specified qualifying character (here, a single-quotation mark) around each item in the list.
If you select the second and fourth check boxes in the form, the following statement gets sent to the database:

```
SELECT *
 FROM Departmt
 WHERE Dept_Name IN ('Marketing','Sales')
```

Multiple selection lists

A multiple-selection list box is defined by a `select` or `cfselect` tag with a `multiple` or `multipe="yes"` attribut e and a `size` attribute value greater than 1. ColdFusion treats the result when a user selects multiple choices from a multiple-selection list box like the results of selecting multiple check boxes. The data made available to your page from any multiple-selection list box is a comma-delimited list of the entries selected by the user; for example, a list box could contain the four entries: Training, Marketing, HR, and Sales. If the user selects Marketing and Sales, the form field variable value is Marketing, Sales.
You can use multiple-selection lists to search a database in the same way that you use check boxes.

Handling numeric values

Suppose you want the user to select departments from a multiple-selection list box. The query retrieves detailed information on the selected departments, as follows:

```
Select one or departments to get more information on:
<cfselect name="SelectDepts" multiple>
 <option value="1">Training
 <option value="2">Marketing
 <option value="3">HR
 <option value="4">Sales
</cfselect>
```

If the user selects the Marketing and Sales items, the value of the SelectDepts form field is 2,4. If this parameter is used in the following SQL statement:

```
SELECT *
 FROM Departmt
 WHERE Dept_ID IN (#form.SelectDepts#)
```

The following statement is sent to the database:

```
SELECT *
 FROM Departmt
 WHERE Dept_ID IN (2,4)
```

Handling string values

© 2014 Adobe Systems Incorporated. All rights reserved.

Suppose you want the user to select departments from a multiple-selection list box. The database search field is a string field. The query retrieves detailed information on the selected departments, as follows:

```
<cfselect name="SelectDepts" multiple>
 <option value="Training">Training
 <option value="Marketing">Marketing
 <option value="HR">HR
 <option value="Sales">Sales
</cfselect>
```

If the user selects the Marketing and Sales items, the SelectDepts form field value is Marketing,Sales.
Just as you did when using check boxes to search database fields containing string values, use the ColdFusion `ListQualify` function with multiple-selection list boxes:

```
SELECT *
 FROM Departmt
 WHERE Dept_Name IN (#ListQualify(Form.SelectDepts,"'")#)
```

The following statement is sent to the database:

```
SELECT *
 FROM Departmt
 WHERE Dept_Name IN ('Marketing','Sales')
```

#back to top

© 2014 Adobe Systems Incorporated. All rights reserved.

Building Dynamic Forms with cfform Tags

You can use the `cfform` tag to create rich, dynamic forms with sophisticated graphical controls, including several Java applet or Flash controls. You can use these controls without writing a line of Java or Flash code.

© 2014 Adobe Systems Incorporated. All rights reserved.

Creating custom forms with the cfform tag

The cfform tag and its CFML subtags let you create dynamic forms in three formats:

- **HTML** Generates standard HTML tags wherever possible, and uses applets or Flash for more complex controls, such as grids, trees, and calendars. HTML lets you present a familiar appearance, but does not let you easily separate data and presentation, or provide some of the more complex structures, such as Flash tabbed navigators or accordions, or customized XML controls.
- **Flash** Presents a modern, visually pleasing appearance. Flash format supports several controls, such as tabbed navigators and accordions, that are not available in HTML. Flash forms are also browser-independent. In Flash format, Flash Player works in all commonly used browsers on Windows and Macintosh systems, and in Netscape and Mozilla on Linux.
- **XML** Lets you specify an Extensible Stylesheet Language Transformation (XSLT) skin that converts the XML into styled HTML output. Adobe ColdFusion provides several skins that you can use, and you can write your own custom skins and support custom controls.The `cfform` tag and its subtags also provide you with several methods for validating input data. For example, you can perform the validation on the browser or on the server. You can check the data type, or you can mask data input.Individual `cfform` tags have additional dynamic features. Several of the tags do not have HTML counterparts, and others directly support dynamically populating the control from data sources. Also, the `cfform` tag `preservedata` attribute retains user input in a form after the user submits the form, so the data reappears if the form gets redisplayed.The information here describes features of the `cfform` tag and focuses on using several of the `cfform` child tags that do not have HTML counterparts. For other features of ColdFusion forms that you create using the `cfformtag`, see the following:
  - Validating data
  - Creating Forms in Flash
  - Creating Skinnable XML Forms

The cfform controls

The following table describes the ColdFusion controls that you use in forms created using the `cfform` tag. You can use these tags only inside a `cfform` tag. Unless otherwise stated, these controls are supported in HTML, Flash, and XML skinnable forms.

Control	Description	For more information
cfapplet	Embeds a custom Java applet in the form. Not supported in Flash format forms.	Embedding Java applets.
cfcalendar	Displays an interactive Flash calendar that can be included in an HTML or Flash format form. ignored in XML skinable forms. The calendar lets a user select a date for submission as a form variable.	The cfcalendar tag in the *CFML Reference*
cfform	Creates a container control for organizing and formatting multiple form controls. Used in the `cfform` tag body of Flash and XML skinable forms. Ignored in HTML forms.	Creating Forms in Flash, Creating Skinnable XML Forms

© 2014 Adobe Systems Incorporated. All rights reserved.

cfformitem	Inserts a horizontal line, a vertical line, or formatted or unformatted text in a Flash form. Used in the `cf form` or `cfformgroup` tag body for Flash and XML forms. Ignored in HTML forms.	Creating Forms in Flash, Creating Skinnable XML Forms
cfgrid	Creates a Java applet or Flash data grid that you can populate from a query or by defining the contents of individual cells. You can also use grids to insert, update, and delete records from a data source.	Creating data grids with the cfgrid tag
cfinput	Equivalent to the HTML `input` tag, with the addition of input validation.	*Creating a basic form* in Using forms in ColdFusion
cfselect	Displays a selection box. Equivalent to the HTML `select` tag, with the addition of input validation.	Building drop-down list boxes
cfslider	Creates a Java applet-based control that lets users enter data by moving a slider. Not supported in Flash format forms.	Building slider bar controls
cftextarea	Displays a text input area. Equivalent to the HTML `textarea` tag, with the addition of input validation.	The cftextarea tag in the *CFML Reference*
cftree	Creates a Java applet or Flash hierarchical tree-format control that can include graphical images for the different elements. Can also generate a ColdFusion structure that represents the tree data and attributes.	Building tree controls with the cftree tag

Preserving input data with the preservedata attribute

The `cfform` `preservedata` attribute tells ColdFusion to continue displaying the user data in a form after the user submits the form. Data is preserved in the `cfinput`, `cfslider`, `cftextinput`, and `cftree` controls and in `cfselect` controls populated by queries. If you specify a default value for a control, and a user overrides that default in the form, the user input is preserved.
You can retain data on the form when the same page contains the form and the form's action code; that is, the form submits to itself. You can also retain the data if the action page has a copy of the form, and the control names are the same in the forms on both pages. (The action page form need not be identical to the initial form. It can have more or fewer elements than the initial page form; only the form elements with identical names on both pages keep their data.)

© 2014 Adobe Systems Incorporated. All rights reserved.

Note

The `preservedata` setting on the action page controls the preservation of the data.

For example, if you save this form as preserve.cfm, it continues to display any text that you enter after you submit it, as follows:

```
<cfform action="preserve.cfm" preservedata="Yes">
 <p>Please enter your name:
 <cfinput type="Text" name="UserName" required="Yes"><p>
 <input type="Submit" name=""> <input type="RESET">
</cfform>
```

Usage notes for the preservedata attribute

When you use the `preservedata` attribute, follow these guidelines:

- In the `cftree` tag, the `preservedata` attribute causes the tree to expand to the previously selected element. For this to work correctly, set the `completePath` attribute to True.
- The `preservedata` attribute has no effect on a `cfgrid` tag. If you populate the control from a query, update the data source with the new data (typically by using a `cfgridupdate` tag) before redisplaying the grid. The grid then displays the updated database information.

Browser considerations

The applet-based versions of the `cfgrid`, `cfslider`, and `cftree` forms use JavaScript and Java to display their content. To allow them to display consistently across a variety of browsers, these applets use the Java plug-in. As a result, they are independent of the level of Java support provided by the browser.
ColdFusion downloads and installs the browser plug-in if necessary. Some browsers display a single permission dialog box asking you to confirm the plug-in installation. Other browsers, like older versions of Netscape, require you to navigate some simple option windows.
Because the controls use JavaScript to return data to ColdFusion, if you disable JavaScript in your browser, it cannot properly run forms that contain these controls. In that case, the controls still display, but data return and validation does not work and you can receive a JavaScript error.
Because Java is handled by the plug-in and not directly by the browser, disabling Java execution in the browser does not affect the operation of the controls. If for some other reason, however, the browser is unable to render the controls as requested, a "not supported" message appears in place of the control.
You can use the `cfform` tag's `notsupported` attribute to specify an alternative error message.
You can avoid browser Java and JavaScript issues with the `cfgrid` and `cftree` controls by using the Flash format versions of these controls. These controls work on Windows, Mac OS X, and Linux, and do not rely on Java support. There is no Flash version of the `cfslider` control, and there is no applet version of the `cfcalendar` control.

© 2014 Adobe Systems Incorporated. All rights reserved.

Building drop-down list boxes

The drop-down list box that you can create in a `cfform` tag with a `cfselect` tag is like the HTML `select` tag. However, the `cfselect` tag gives you more control over user inputs, provides error handling, and, most importantly, lets you automatically populate the selection list from a query.
You can populate the drop-down list box from a query, or using lists of option elements created by the `option` tag. The syntax for the `option` tag with the `cfselect` tag is the same as for the HTML `option` tag.
When you populate a `cfselect` tag with data from a query, you only need to specify the name of the query that is supplying data for the `cfselect` tag and the query column name for each list element to display.

Populate a drop-down list box with query data using the cfselect tag

1. Create a ColdFusion page with the following content:

```
<cfquery name="getNames"
 datasource="cfdocexamples">
 SELECT * FROM Employee
</cfquery>

<cfform name="Form1" action="submit.cfm">

 <cfselect name="employees"
  query="getNames"
  value="Emp_ID"
  display="FirstName"
  required="Yes"
  multiple="Yes"
  size="8">
 </cfselect>

 <br><input type="Submit" value="Submit">
</cfform>
```

2. Save the file as selectbox.cfm and view it in your browser.
   Because the tag includes the `multiple` attribute, the user can select multiple entries in the list box. Also, because the `value` tag specifies Emp_ID, the primary key for the Employee table, Employee IDs (not first names) get passed in the Form.Employee variable to the application page specified in the `cfform` `action` a ttribute.
   You can use a query to create a two-level hierarchical list grouped by one of the query columns. For an example of this use, see the example for the `cfselect` entry in the *CFML Reference*.

© 2014 Adobe Systems Incorporated. All rights reserved.

Building slider bar controls

You can use the `cfslider` control in a `cfform` tag to create a slider control and define a variety of characteristics, including label text, label font name, size, boldface, italics, and color, and slider range, positioning, and behavior. Slider bars are useful because they are highly visual and users can only enter valid values. The `cfslider` tag is not supported in Flash format forms.

Create a slider control

1. Create a ColdFusion page with the following content:

```
<cfform name="Form1" action="submit.cfm">
 <cfslider name="myslider"
  bgcolor="cyan"
  bold="Yes"
  range="0,1000"
  scale="100"
  value="600"
  fontsize="14"
  label="Slider %value%"
  height="60"
  width="400">
</cfform>
```

2. Save the file as slider.cfm and view it in your browser.
   To get the value of the slider in the action page, use the variable Form.*slider_name*; in this case, Form.myslider.

© 2014 Adobe Systems Incorporated. All rights reserved.

Building tree controls with the cftree tag

The `cftree` tag lets you display hierarchical information within a form in a space-saving collapsible tree populated from data source queries. To build a tree control with the `cftree` tag, you use individual `cftreeitem` tags to populate the control.
You can create trees in three formats:

- **Applet** Creates a Java applet that the client must download. Downloading an applet takes time; therefore, using the `cftree` tag can be slightly slower than using an HTML form element to retrieve the same information. In addition, browsers must be Java-enabled for the `cftree` tag to work properly.
- **Flash** Generates a Flash control that you can include in an HTML or Flash form. For more information on Flash Forms see Creating Forms in Flash.
- **Object** Creates a hierarchical ColdFusion structure that represents the tree data and many of the `cftree` an d `cftreeitem` attributes.The different formats support different sets of features and attributes. The information here discusses general techniques that apply to all three formats, and indicates any techniques that do not apply to a specific format. It uses applet format for all examples, which use applet-specific attributes. For details on the features and attributes supported in each format, see the `cftree` entry in the *C FML Reference*.

Create and populate a tree control from a query

1. Create a ColdFusion page with the following content:

```
<cfquery name="engquery" datasource="cfdocexamples">
 SELECT FirstName || ' ' || LastName AS FullName
 FROM Employee
</cfquery>
<cfform name="form1" action="submit.cfm">
<cftree name="tree1"
 required="Yes"
 hscroll="No">
 <cftreeitem value="FullName"
  query="engquery"
  queryasroot="Yes"
  img="folder,document">
</cftree>
</cfform>
```

2. Save the page as tree1.cfm and view it in your browser.

Reviewing the code

The following table describes the highlighted code and its function:

Code	Description
`<cftree name="tree1"`	Creates a tree and names it tree1.
`required="Yes"`	Specifies that a user must select an item in the tree.
`hscroll="No"`	Does not allow horizontal scrolling.

© 2014 Adobe Systems Incorporated. All rights reserved.

`<cftreeitem value="FullName" query="engquery"`	Creates an item in the tree and puts the results of the query named engquery in it. Because this tag uses a query, it puts one item on the tree per query entry.
`queryasroot="Yes"`	Specifies the query name as the root level of the tree control.
`img="folder,document"`	Uses the folder and document images that ship with ColdFusion in the tree structure. When populating a cftree tag with data from a cfquery tag, you can specify images or filenames for each level of the tree as a comma-separated list.

Grouping output from a query

In a query that you display using a `cftree` control, to organize your employees by department, separate column names with commas in the `cftreeitem` `value` attribute.

Organize the tree based on ordered results of a query

1. Create a ColdFusion page named tree2.cfm with the following content:

```
<!--- CFQUERY with an ORDER BY clause. --->
<cfquery name="deptquery" datasource="cfdocexamples">
 SELECT Dept_ID, FirstName || ' ' || LastName
 AS FullName
 FROM Employee
 ORDER BY Dept_ID
</cfquery>

<!--- Build the tree control. --->
<cfform name="form1" action="submit.cfm">

<cftree name="tree1"
 hscroll="No"
 border="Yes"
 height="350"
 required="Yes">

<cftreeitem value="Dept_ID, FullName"
 query="deptquery"
 queryasroot="Dept_ID"
 img="computer,folder,document"
 imgopen="computer,folder"
 expand="yes">

</cftree>
<br>
<br><input type="Submit" value="Submit">
</cfform>
```

2. Save the page and view it in your browser.

© 2014 Adobe Systems Incorporated. All rights reserved.

Reviewing the code

The following table describes the highlighted code and its function

Code	Description
`ORDER BY Dept_ID`	Orders the query results by department.
`<cftreeitem value="Dept_ID,FullName"`	Populates the tree with the department ID, and under each department, the full name for each employee in the department.
`queryasroot="Dept_ID"`	Labels the root "Dept_ID".
`img="computer,folder,document"imgopen="computer,folder"`	Uses the ColdFusion supplied computer image for the root level, folder image for the department IDs, and document for the names, independent of whether any level is expanded (open) or collapsed. The `imgopen` attribute has only two items, because the employee names can never be open.

The `cftreeitem` comma-separated `value`, `img`, and `imgopen` attributes correspond to the tree level structure. In applet format, if you omit the `img` attribute, ColdFusion uses the folder image for all levels in the tree; if you omit the `imgopen` attribute, ColdFusion uses the folder image for all expanded levels in the tree. Flash format ignores the `img` and `imgopen` attributes and always uses folders for levels with children and documents for nodes without children.

The cftree form variables

The `cftree` tag lets you force a user to select an item from the tree control by setting the `required` attribute to Yes. With or without the `required` attribute, ColdFusion passes two form variables to the application page specified in the `cfform` `action` attribute:

- Form.*treename*.path Returns the complete path of the user selection, in the form: *root\node1\node2\node_n\value*
- Form.*treename*.node Returns the node of the user selection.
  To return the root part of the path, set the `completepath` attribute of the `cftree` tag to Yes; otherwise, the path value starts with the first node. If you specify a root name for a tree item using the `queryasroot` tag, that value is returned as the root. If you do not specify a root name, ColdFusion returns the query name as the root. If there is no query name, ColdFusion returns the tree name as the root.
  In the previous example, if the user selects the name "John Allen" in the tree, ColdFusion returns the following form variables:

```
Form.tree1.path = 3\John Allen
Form.tree1.node = John Allen
```

The deptquery root does not appear in the path, because the `cftree` tag does not specify `completePath="Yes"`. You can specify the character used to delimit each element of the path form variable in the `cftree` `delimiter` attribute. The default is a backslash character ().

© 2014 Adobe Systems Incorporated. All rights reserved.

Input validation

Although the cftree tag does not include a validate attribute, you can use the required attribute to force a user to select an item from the tree control. In addition, you can use the onValidate attribute to specify your own JavaScript code to perform validation.

Structuring tree controls

Tree controls built with the cftree tag can be complex. Knowing how to specify the relationship between multiple cftreeitem entries helps you handle the most complex cftree constructs.

Creating a one-level tree control

The following example consists of a single root and some individual items:

```
<cfquery name="deptquery" datasource="cfdocexamples">
 SELECT Dept_ID, FirstName || ' ' || LastName
 AS FullName
 FROM Employee
 ORDER BY Dept_ID
</cfquery>

<cfform name="form1" action="submit.cfm">
 <cftree name="tree1">
  <cftreeitem value="FullName"
  query="deptquery"
  queryasroot="Department">
  img="folder,document">
 </cftree>
<br>
<cfinput type="submit" value="Submit">
</cfform>
```

Creating a multilevel tree control

When populating a cftree control, you create the multilevel structure of the tree by specifying a parent for each item in the tree. The parent attribute of the cftreeitem tag allows your cftree tag to show relationships between elements in the tree.
In this example, every cftreeitem tag, except the top level Divisions, specifies a parent. For example, the cftreeitem tag specifies Divisions as its parent.
The following code populates the tree directly, not from a query:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cfform name="form2" action="cfform_submit.cfm">
<cftree name="tree1" hscroll="No" vscroll="No"
 border="No">
 <cftreeitem value="Divisions">
 <cftreeitem value="Development"
  parent="Divisions" img="folder">
 <cftreeitem value="Product One"
  parent="Development" img="document">
 <cftreeitem value="Product Two"
  parent="Development">
 <cftreeitem value="GUI"
  parent="Product Two" img="document">
 <cftreeitem value="Kernel"
  parent="Product Two" img="document">
 <cftreeitem value="Product Three"
  parent="Development" img="document">
 <cftreeitem value="QA"
  parent="Divisions" img="folder">
 <cftreeitem value="Product One"
  parent="QA" img="document">
 <cftreeitem value="Product Two"
  parent="QA" img="document">
 <cftreeitem value="Product Three"
  parent="QA" img="document">
 <cftreeitem value="Support"
  parent="Divisions" img="fixed">
 <cftreeitem value="Product Two"
  parent="Support" img="document">
 <cftreeitem value="Sales"
  parent="Divisions" img="computer">
 <cftreeitem value="Marketing"
  parent="Divisions" img="remote">
 <cftreeitem value="Finance"
  parent="Divisions" img="element">
</cftree>
</cfform>
```

Image names in a cftree tag

> **Note**
>
> The information here applies to applet format trees. In Flash, you cannot control the tree icons. Flash uses open and closed folders and documents as the icons. In object format, the image information is preserved in fields in the object structure.

The default image displayed in a tree is a folder. However, you can use the `img` attribute of the `cftreeitem` tag to specify a different image.

When you use the `img` attribute, ColdFusion displays the specified image beside the tree items when they are not open. When you use the `imgopen` attribute, ColdFusion displays the specified image beside the tree items when they are open (expanded). You can specify a built-in ColdFusion image name, the file path to an image file, or the URL of an image of your choice, such as http://localhost/Myapp/Images/Level3.gif. You cannot use a custom image in Flash format. As a general rule, make the height of your custom images less than 20 pixels.

When populating a `cftree` control with data from a `cfquery` tag, you can use the `img` attribute of `cftreeitem` ta

© 2014 Adobe Systems Incorporated. All rights reserved.

g to specify images or filenames for each level of the tree as a comma-separated list.
The following are the ColdFusion built-in image names:

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

Note

In applet format, you can also control the tree appearance by using the `cftree` tag `lookAndFeel` attribute to specify a Windows, Motif, or Metal look.

Embedding URLs in a cftree tag

The `href` attribute in the `cftreeitem` tag lets you designate tree items as links. To use this feature in a `cftree` control, you define the destination of the link in the `href` attribute of the `cftreeitem` tag. The URL for the link can be a relative URL or an absolute URL, as in the following examples.

Embed links in a cftree control

1. Create a ColdFusion page named tree3.cfm with the following contents:

```
<cfform action="submit.cfm">
<cftree name="oak"
 highlighthref="Yes"
 height="100"
 width="200"
 hspace="100"
 vspace="6"
 hscroll="No"
 vscroll="No"
 border="No">
 <cftreeitem value="Important Links">
 <cftreeitem value="Adobe Home"
  parent="Important Links"
  img="document"
  href="http://www.adobe.com">
 <cftreeitem value="ColdFusion Developer Center"
  parent="Important Links"
  img="document"
  href="http://www.adobe.com/devnet/coldfusion/">
</cftree>
</cfform>
```

2. Save the page and view it in your browser.

Reviewing the code

The following table describes the highlighted code and its function:

© 2014 Adobe Systems Incorporated. All rights reserved.

Code	Description
`href="http://www.adobe.com">`	Makes the node of the tree a link.
`href="http://www.adobe.com/devnet/mx/coldfusion/">`	Makes the node of the tree a link.Although this example does not show it, the `href` attribute can refer to the name of a column in a query if that query populates the tree item.

Specifying the tree item in the URL

When a user clicks a tree item to link to a URL, the `cftreeItemKey` variable, which identifies the selected value, is appended to the URL in the following form:

```
http://myserver.com?CFTREEITEMKEY=selected_item_value_attribute
```

If the value attribute includes spaces, ColdFusion replaces the spaces with plus characters (+).
Automatically passing the name of the selected tree item as part of the URL makes it easy to implement a basic "drill down" application that displays additional information based on the selection. For example, if the specified URL is another ColdFusion page, it can access the selected value as the variable `URL.CFTREEITEMKEY`.
To disable this behavior, set the `appendkey` attribute in the `cftree` tag to no.

© 2014 Adobe Systems Incorporated. All rights reserved.

Creating data grids with the cfgrid tag

The `cfgrid` tag creates a `cfform` grid control that resembles a spreadsheet table and can contain data populated from a `cfquery` tag or from other sources of data. As with other `cfform` tags, the `cfgrid` tag offers a wide range of data formatting options, as well as the option of validating user selections with a JavaScript validation script.
You can also perform the following tasks with a `cfgrid` tag:

- Sort data in the grid alphanumerically.
- Update, insert, and delete data.
- Display images in the grid.

> **Note**
>
> Flash format grids support a subset of the features available in applet format grids. For details on features supported in each format, see the `cfgrid` tag in the CFML Reference.

Users can sort the grid entries in ascending order by double-clicking any column header. Double-clicking again sorts the grid in descending order. In applet format, you can also add sort buttons to the grid control.
When users select grid data and submit the form, ColdFusion passes the selection information as form variables to the application page specified in the `cfform` `action` attribute.
Just as the `cftree` tag uses the `cftreeitem` tag, the `cfgrid` tag uses the `cfgridcolumn` and `cfgridrow` tags . You can define a wide range of row and column formatting options, as well as a column name, data type, selection options, and so on. You use the `cfgridcolumn` tag to define individual columns in the grid or associate a query column with a grid column.
Use the `cfgridrow` tag to define a grid that does not use a query as the source for row data. If a query attribute is specified in the `cfgrid` tag, the `cfgridrow` tags are ignored.
The `cfgrid` tag provides many attributes that control grid behavior and appearance. Only the most important of these attributes are described here. For detailed information on these attributes, see the `cfgrid` tag in the *CFML Reference*.

Working with a data grid and entering data

The following image shows an example applet format grid created using a `cfgrid` tag.
The following table describes some navigating tips:

Action	Procedure
Sorting grid rows	Double-click the column header to sort a column in ascending order. Double-click again to sort the rows in descending order.
Rearranging columns	Click any column heading and drag the column to a new position.
Determining editable grid areas	When you click an editable cell, it is surrounded by a yellow box.
Determining noneditable grid areas	When you click a cell (or row or column) that you cannot edit, its background color changes. The default color is salmon pink.

© 2014 Adobe Systems Incorporated. All rights reserved.

Editing a grid cell	Double-click the cell. Press Return when you finish entering the data.
Deleting a row	Click any cell in the row and click the Delete button. (Not available in Flash format grids.)
Inserting a row	Click the Insert button. An empty row appears at the bottom of the grid. To enter a value in each cell, double-click the cell, enter the value, and click Return. (Not available in Flash format grids.)

Populate a grid from a query

1. Create a ColdFusion page named grid1.cfm with the following contents:

```
<cfquery name="empdata" datasource="cfdocexamples">
 SELECT * FROM Employee
</cfquery>

<cfform name="Form1" action="submit.cfm" >
 <cfgrid name="employee_grid" query="empdata"
   selectmode="single">
  <cfgridcolumn name="Emp_ID">
  <cfgridcolumn name="LastName">
  <cfgridcolumn name="Dept_ID">
 </cfgrid>
 <br>
 <cfinput name="submitit" type="Submit" value="Submit">
</cfform>
```

Note

Use the `cfgridcolumn` `display="No"` attribute to hide columns that you want to include in the grid but not expose to an end user. You typically use this attribute to include columns such as the table's primary key column in the results returned by the `cfgrid`_ tag._

1. Save the file and view it in your browser.

Reviewing the code

The following table describes the highlighted code and its function:

Code	Description
`<cfgrid name="employee_grid" query="empdata"`	Creates a grid named employee_grid and populate it with the results of the query empdata.If you specify a cf grid tag with a query attribute defined and no corresponding cfgridcolumn attributes, the grid contains all the columns in the query.

© 2014 Adobe Systems Incorporated. All rights reserved.

`selectmode="single">`	Allows the user to select only one cell; does not allow editing. Other modes are row, column, and edit.
`<cfgridcolumn name="Emp_ID">`	Puts the contents of the Emp_ID column in the query results in the first column of the grid.
`<cfgridcolumn name="LastName">`	Puts the contents of the LastName column in the query results in the second column of the grid.
`<cfgridcolumn name="Dept_ID">`	Puts the contents of the Dept_ID column in the query results in the third column of the grid.

Creating an editable grid

You can build grids to allow users to edit data within them. Users can edit individual cell data, as well as insert, update, or delete rows. To enable grid editing, you specify `selectmode="edit"` in the `cfgrid` tag.
You can let users add or delete grid rows by setting the `insert` or `delete` attributes in the `cfgrid` tag to Yes. Setting the `insert` and `delete` attribute to Yes causes the `cfgrid` tag to display Insert and Delete buttons as part of the grid.
You can use a grid in two ways to change your ColdFusion data sources:

- Create a page to which you pass the `cfgrid` form variables. In that page, perform `cfquery` operations to update data source records based on the form values returned by the `cfgrid` tag.
- Pass grid edits to a page that includes the `cfgridupdate` tag, which automatically extracts the form variable values and passes that data directly to the data source.
  Using the `cfquery` tag gives you complete control over interactions with your data source. The `cfgridupdate` tag provides a much simpler interface for operations that do not require the same level of control.

Controlling cell contents

You can control the data that a user can enter into a `cfgrid` cell in the following ways:

- By default, a cell is not editable. Use the `cfgrid` attribute `selectmode="edit"` to edit cell contents.
- Use the `cfgridcolumn` `type` attribute to control sorting order, to make the fields check boxes, or to display an image.
- Use the `cfgridcolumn` `values` attribute to specify a drop-down list of values from which the user can choose. You can use the `valuesDisplay` attribute to provide a list of items to display that differs from the actual values that you enter in the database. You can use the `valuesDelimiter` attribute to specify the separator between values in the `values` `valuesDisplay` lists.
- Although the `cfgrid` tag does not have a `validate` attribute, it does have an `onValidate` attribute that lets you specify a JavaScript function to perform validation.
  For more information on controlling the cell contents, see the attribute descriptions for the `cfgridcolumn` tag in the *CFML Reference*.

How user edits are returned

When a user inserts or deletes a row in a grid or changes any cells in a row and submits the grid, ColdFusion creates the following arrays as Form variables:

Array name	Description

© 2014 Adobe Systems Incorporated. All rights reserved.

gridname.colname	Stores the new values of inserted, deleted, or updated cells. (Entries for deleted cells contain empty strings.)
gridname.Original.colname	Stores the original values of inserted, deleted, or updated cells.
gridname.RowStatus.Action	Stores the type of change made to the grid rows: D for delete, I for insert, or U for update.

Note

The periods in these names are **not** structure separators; they are part of the text of the array name.

ColdFusion creates a *gridname.colname* array and a *gridname.Original.colname* array for each column in the grid. For each inserted, deleted, or changed row in the grid, ColdFusion creates a row in each of these arrays.
For example, the following arrays are created if you update a `cfgrid` tag called mygrid that consists of two displayable columns (col1, col2) and one hidden column (col3):

```
Form.mygrid.col1
Form.mygrid.col2
Form.mygrid.col3
Form.mygrid.original.col1
Form.mygrid.original.col2
Form.mygrid.original.col3
Form.mygrid.RowStatus.Action
```

The value of the array index increments for each row that is added, deleted, or changed, and does not indicate a grid row number. All rows for a particular change have the same index in all arrays. Unchanged rows do not have entries in the arrays.
If the user changes a single cell in col2, the following array elements contain the edit operation, the edited cell value, and the original cell value:

```
Form.mygrid.RowStatus.Action[1]
Form.mygrid.col2[1]
Form.mygrid.original.col2[1]
```

If the user changes the values of the cells in col1 and col3 in one row and the cell in col2 in another row, the information about the original and changed values is in the following array entries:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
Form.mygrid.RowStatus.Action[1]
Form.mygrid.col1[1]
Form.mygrid.original.col1[1]
Form.mygrid.col3[1]
Form.mygrid.original.col3[1]
Form.mygrid.RowStatus.Action[2]
Form.mygrid.col2[2]
Form.mygrid.original.col2[2]
```

The remaining cells in the arrays (for example, Form.mygrid.col21 and Form.mygrid.original.col21) have the original, unchanged values.

Example: editing data in a grid

The following example creates an editable grid. For code brevity, the example handles only three of the fields in the Employee table. A more realistic example would include, at a minimum, all seven table fields. It can also hide the contents of the Emp_ID column or display the Department name (from the Departmt table), instead of the Department ID.

Create the editable grid

1. Create a ColdFusion page with the following content:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cfquery name="empdata" datasource="cfdocexamples">
 SELECT * FROM Employee
</cfquery>

<cfform name="GridForm"
 action="handle_grid.cfm">

 <cfgrid name="employee_grid"
  height=425
  width=300
  vspace=10
  selectmode="edit"
  query="empdata"
  insert="Yes"
  delete="Yes">

  <cfgridcolumn name="Emp_ID"
   header="Emp ID"
   width=50
   headeralign="center"
   headerbold="Yes"
   select="No">

  <cfgridcolumn name="LastName"
   header="Last Name"
   width=100
   headeralign="center"
   headerbold="Yes">

  <cfgridcolumn name="Dept_ID"
   header="Dept"
   width=35
   headeralign="center"
   headerbold="Yes">

 </cfgrid>
 <br>
 <cfinput name="submitit" type="Submit" value="Submit">
</cfform>
```

2. Save the file as grid2.cfm.
3. View the results in your browser.

h7. Reviewing the code
The following table describes the code and its function:

Code	Description

© 2014 Adobe Systems Incorporated. All rights reserved.

`<cfgrid name="employee_grid"`
` height=425`
` width=300`
` vspace=10`
` selectmode="edit"`
` query="empdata"`
` insert="Yes"`
` delete="Yes">`
	Populates a cfgrid control with data from the empdata query. Selecting a grid cell enables you to edit it. You can insert and delete rows. The grid is 425 X 300 pixels and has 10 pixels of space above and below it.

`<cfgridcolumn name="Emp_ID"`
` header="Emp ID"`
` width=50`
` headeralign="center"`
` headerbold="Yes"`
` select="No">`
	Creates a 50-pixel wide column for the data in the Emp_ID column of the data source. Centers a header named Emp ID and makes it bold. Does not allow users to select fields in this column for editing. Since this field is the table's primary key, users should not be able to change it for existing records, and the DBMS should generate this field as an autoincrement value.

`<cfgridcolumn name="LastName"`
` header="Last Name"`
` width=100`
` headeralign="center"`
` headerbold="Yes">`
	Creates a 100-pixel wide column for the data in the LastName column of the data source. Centers a header named Last Name and makes it bold.

`<cfgridcolumn name="Dept_ID"`
` header="Dept"`
` width=35`
` headeralign="center"`
` headerbold="Yes">`
	Creates a 35-pixel wide column for the data in the Dept_ID column of the data source. Centers a header named Dept and makes it bold.

Updating the database with the cfgridupdate tag

The `cfgridupdate` tag provides a simple mechanism for updating the database, including inserting and deleting records. It can add, update, and delete records simultaneously. It is convenient because it automatically handles collecting the `cfgrid` changes from the various form variables, and generates appropriate SQL statements to update your data source.
In most cases, use the `cfgridupdate` tag to update your database. However, this tag does not provide the complete SQL control that the `cfquery` tag provides. In particular, the `cfgridupdate` tag has the following characteristics:

- You can update only a single table.
- Rows are deleted first, then rows are inserted, then any changes are made to existing rows. You cannot modify the order of changes.
- Updating stops when an error occurs. It is possible that some database changes are made, but the tag does not provide any information on them.

© 2014 Adobe Systems Incorporated. All rights reserved.

Update the data source with the cfgridupdate tag

1. Create a ColdFusion page with the following contents:

```
<html>
<head>
 <title>Update grid values</title>
</head>
<body>

<h3>Updating grid using cfgridupdate tag.</h3>

<cfgridupdate grid="employee_grid"
 datasource="cfdocexamples"
 tablename="Employee">

Click <a href="grid2.cfm">here</a> to display updated grid.

 </body>
</html>
```

2. Save the file as handle_grid.cfm.
3. View the grid2.cfm page in your browser, change the grid, and then submit them.

> **Note**
>
> To update a grid cell, modify the cell contents, and then press Return.

h7. Reviewing the code
The following table describes the highlighted code and its function:

Code	Description
`<cfgridupdate grid="employee_grid"`	Updates the database from the Employee_grid grid.
`datasource="cfdocexamples"`	Updates the cfdocexamples data source.
`tablename="Employee"`	Updates the Employee table.

Updating the database with the cfquery tag

You can use the `cfquery` tag to update your database from the `cfgrid` changes. This tag provides you with full control over how the updates are made and lets you handle any errors that arise.

Update the data source with the cfquery tag

1. Create a ColdFusion page with the following content:

```
<html>
<head>
 <title>Catch submitted grid values</title>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

```
</head>
<body>

<h3>Grid values for Form.employee_grid row updates</h3>

<cfif isdefined("Form.employee_grid.rowstatus.action")>

 <cfloop index = "counter" from = "1" to =
  #arraylen(Form.employee_grid.rowstatus.action)#>

  <cfoutput>
   The row action for #counter# is:
   #Form.employee_grid.rowstatus.action[counter]#
   <br>
  </cfoutput>

  <cfif Form.employee_grid.rowstatus.action[counter] is "D">
   <cfquery name="DeleteExistingEmployee"
    datasource="cfdocexamples">
    DELETE FROM Employee
    WHERE Emp_ID=<cfqueryparam
    value="#Form.employee_grid.original.Emp_ID[counter]#"
    CFSQLType="CF_SQL_INTEGER" >
   </cfquery>

  <cfelseif Form.employee_grid.rowstatus.action[counter] is "U">
   <cfquery name="UpdateExistingEmployee"
    datasource="cfdocexamples">
    UPDATE Employee
    SET
     LastName=<cfqueryparam
       value="#Form.employee_grid.LastName[counter]#"
       CFSQLType="CF_SQL_VARCHAR" >,
     Dept_ID=<cfqueryparam
       value="#Form.employee_grid.Dept_ID[counter]#"
       CFSQLType="CF_SQL_INTEGER" >
    WHERE Emp_ID=<cfqueryparam
     value="#Form.employee_grid.original.Emp_ID[counter]#"
     CFSQLType="CF_SQL_INTEGER">
   </cfquery>

  <cfelseif Form.employee_grid.rowstatus.action[counter] is "I">
   <cfquery name="InsertNewEmployee"
    datasource="cfdocexamples">
    INSERT into Employee (LastName, Dept_ID)
    VALUES (<cfqueryparam
     value="#Form.employee_grid.LastName[counter]#"
       CFSQLType="CF_SQL_VARCHAR" >,
      <cfqueryparam value="#Form.employee_grid.Dept_ID[counter]#"
       CFSQLType="CF_SQL_INTEGER" >)
   </cfquery>

  </cfif>
 </cfloop>
</cfif>

Click <a href="grid2.cfm">here</a> to display updated grid.
```

© 2014 Adobe Systems Incorporated. All rights reserved.

```
</body>
</html>
```

2. Rename your existing handle_grid.cfm file as handle_grid2.cfm to save it, and then save this file as handle_grid.cfm.
3. View the grid2.cfm page in your browser, change the grid, and then submit them.

h7. Reviewing the code

The following table describes the code and its function:

Code	Description
`<cfif isdefined("Form.employee_grid.rowstatus.action")> <cfloop index = "counter" from = "1" to = #arraylen(Form.employee_grid.rowstatus.action)#>`	If there is an array of edit types, changes the table. Otherwise, does nothing. Loops through the remaining code once for each row to be changed. The `counter` variable is the common index into the arrays of change information for the row being changed.
`<cfoutput> The row action for #counter# is: #Form.employee_grid.rowstatus.action[counter]#   </cfoutput>`	Displays the action code for this row: U for update, I for insert, or D for delete.

`<cfif Form.employee_grid.rowstatus.action[counter] is "D">`
 `<cfquery name="DeleteExistingEmployee" datasource="cfdocexamples">`
 `DELETE FROM Employee`
 `WHERE Emp_ID=<cfqueryparam value="#Form.employee_grid.original.Emp_ID[counter]#" CFSQLType="CF_SQL_INTEGER" >`
 `</cfquery>`
	If the action is to delete a row, generates a SQL DELETE query specifying the Emp_ID (the primary key) of the row to be deleted.

© 2014 Adobe Systems Incorporated. All rights reserved.

<cfelseif Form.employee_grid.rowstatus.action[counter] is "U"> <cfquery name="UpdateExistingEmployee" datasource="cfdocexamples"> UPDATE Employee SET LastName=<cfqueryparam value="#Form.employee_grid.LastName[counter]#" CFSQLType="CF_SQL_VARCHAR" >, Dept_ID=<cfqueryparam value="#Form.employee_grid.Dept_ID[counter]#" CFSQLType="CF_SQL_INTEGER" > WHERE Emp_ID=<cfqueryparam value="#Form.employee_grid.original.Emp_ID[counter]#" CFSQLType="CF_SQL_INTEGER"> </cfquery>	Otherwise, if the action is to update a row, generates a SQL UPDATE query to update the LastName and Dept_ID fields for the row specified by the Emp_ID primary table key.
<cfelseif Form.employee_grid.rowstatus.action[counter] is "I">\ <cfquery name="InsertNewEmployee" datasource="cfdocexamples">\ INSERT into Employee (LastName, Dept_ID) VALUES (<cfqueryparam value="#Form.employee_grid.LastName[counter]#" CFSQLType="CF_SQL_VARCHAR" >, <cfqueryparam value="#Form.employee_grid.Dept_ID[counter]#" CFSQLType="CF_SQL_INTEGER" >) </cfquery>	Otherwise, if the action is to insert a row, generates a SQL INSERT query to insert the employee's last name and department ID from the grid row into the database. The INSERT statement assumes that the DBMS automatically increments the Emp_ID primary key. If you use the version of the cfdocexamples database that is provided for UNIX installations, the record is inserted without an Emp_ID number.
</cfif> </cfloop> </cfif>	Closes the cfif tag used to select among deleting, updating, and inserting.Closes the loop used for each row to be changed.Closes the `cfif` tag that surrounds all the active code.

© 2014 Adobe Systems Incorporated. All rights reserved.

Embedding Java applets

The `cfapplet` tag lets you embed Java applets either on a ColdFusion page or in a `cfform` tag. To use the `cfapplet` tag, first register your Java applet using the ColdFusion Administrator Java Applets page (under Extensions). In the ColdFusion Administrator, you define the interface to the applet, encapsulating it so that each invocation of the `cfapplet` tag is simple.
The `cfapplet` tag within a form offers several advantages over using the HTML `applet` tag:

- **Return values**: The `cfapplet` tag requires a form field `name` attribute, so you can avoid coding additional JavaScript to capture the applet's return values. You can reference return values like any other ColdFusion form variable: `Form.`*variablename*.
- **Ease of use**: The applet's interface is defined in the ColdFusion Administrator, so each instance of the `cfapplet` tag in your pages only needs to reference the applet name and specify a form variable name.
- **Parameter defaults**: ColdFusion uses the parameter value pairs that you defined in the ColdFusion Administrator. You can override these values by specifying parameter value pairs in the `cfapplet` tag. When an applet is registered, you enter just the applet source and the form variable name:

```
<cfapplet appletsource="Calculator"name="calc_value">
```

By contrast, with the HTML `applet` tag, you must declare all the applet's parameters every time you want to use it in a ColdFusion page.

Registering a Java applet

Before you can use a Java applet in your ColdFusion pages, register the applet in the ColdFusion Administrator.

1. Open the ColdFusion Administrator by clicking the Administrator icon in the ColdFusion Program group and entering the Administrator password.
2. Under Extensions, click Java Applets. The Java Applets page appears.
3. Click the Register New Applet button. The Add/Edit Applet page appears.
4. Enter options in the applet registration fields, as described in the ColdFusion Administrator online help. Use the Add button to add parameters.
5. Click Submit.

Using the cfapplet tag to embed an applet

After you register an applet, you can use the `cfapplet` tag to place the applet in a ColdFusion page. The `cfapplet` tag has two required attributes: `appletsource` and `name`. Because you registered the applet and you defined each applet parameter with a default value, you can run the applet with a simple form of the `cfapplet` tag:

```
<cfapplet appletSource="appletname" name="form_variable">
```

Overriding alignment and positioning values

To override any of the values defined in the ColdFusion Administrator for the applet, you can use the optional `cfapplet` parameters to specify custom values. For example, the following `cfapplet` tag specifies custom spacing and alignment values:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cfapplet appletSource="myapplet"
 name="applet1_var"
 height=400
 width=200
 vspace=125
 hspace=125
 align="left">
```

Overriding parameter values

You can override the values that you assigned to applet parameters in the ColdFusion Administrator by providing new values for any parameter. To override a parameter, you must have already defined the parameter and its default value in the ColdFusion Administrator Applets page. The following example overrides the default values of two parameters, Param1 and Param2:

```
<cfapplet appletSource="myapplet"
 name="applet1_var"
 Param1="registered parameter1"
 Param2="registered parameter2">
```

Handling form variables from an applet

The `cfapplet` tag `name` attribute corresponds to a variable in the action page, Form.*appletname*, which holds any value that the applet method returns when it is executed in the `cfform` tag.
Not all Java applets return values. For instance, graphical widgets might not return a specific value. For this type of applet, the method field in the ColdFusion Administrator remains empty, but you must still provide a `cfapplet name` attribute.
You can only use one method for each applet that you register. If an applet includes more than one method that you want to access, you can register the applet with a unique name for each additional method you want to use.

Reference a Java applet return value in your application page

1. Specify the name of the method in the Add/Registered Java Applet page of the ColdFusion Administrator.
2. Specify the method name in the `name` attribute of the `cfapplet` tag.
   When your page executes the applet, ColdFusion creates a form variable with the name that you specified. If you do not specify a method, ColdFusion does not create a form variable.

#back to top

© 2014 Adobe Systems Incorporated. All rights reserved.

Validating Data-Developing guide

You can validate data in Adobe ColdFusion, including form data, variable data and function parameters.

© 2014 Adobe Systems Incorporated. All rights reserved.

About ColdFusion validation

Data validation lets you control data that is entered into an application by ensuring that the data conforms to specific type or formatting rules. Validation techniques have the following features:

- They let you provide feedback to users so that they can immediately correct information they provide. For example, a form can provide immediate feedback when a user enters a name in a telephone number field, or the form could force the user to enter the number in the correct format.
- They help prevent application errors that can arise when processing invalid data. For example, a validation test can prevent a variable that is used in a calculation from having nonnumeric data.
- They can help enhance security by preventing malicious users from providing data that takes advantage of system security weaknesses, such as buffer overrun attacks.
  ColdFusion provides several techniques to ensure that data is valid. These include techniques for validating form data and for validating ColdFusion variables. They also include techniques for validating form data before the user submits it to ColdFusion, or on the ColdFusion server.
  When you design data validation you consider the following factors:

- **The validation technique** Whether to validate on the client's browser or on the server, and the specific server- or client-side validation technique, such as whether to validate when a field loses focus or when the user submits the form.
- **The validation type** The specific method that you use to validate the data, including the rules that you apply to test the data validity, such as testing for a valid telephone number.

Validation techniques

Different validation techniques apply to different ColdFusion tags or coding environments; for example, you can use masking only in HTML and Flash format `cfinput` tags. Validation techniques also vary in where and when they execute; for example, on the client browser when the user submits form data, or on the server when processing data.
The following table describes the ColdFusion validation techniques:

Validation technique	Applies to	Where and when performed	Description
mask(`mask` attribute)	HTML and Flash format `cfinput` tags	On the client as the user enters data	ColdFusion generates JavaScript or ActionScript to directly control the data a user enters by specifying a pattern. For example, 999-999-9999 requires a user to enter ten digits, and automatically fills in the dash (-) separators to create a formatted telephone number.For detailed information on using masks, see Handling invalid data.

© 2014 Adobe Systems Incorporated. All rights reserved.

onBlur(`validateat="onBlur"` attribute)	`cfinput` and `cftextarea` tags	On the client when the data field loses focus	In HTML and XML format, ColdFusion generates JavaScript that runs on the browser to check whether entered data is valid and provide immediate feedback, if the entry is invalid.In Flash format, uses Flash built-in validation routines.
onSubmit(`validateat="onSubmit"` attribute)	`cfinput` and `cftextarea` tags	On the client when the user clicks Submit	In HTML or XML format, the validation logic is identical to onBlur validation, but the test is not done until the user submits the form.In Flash format, this validation type is identical to onBlur Validation. Flash checks do not differentiate between the two events for validation.
onServer(`validateat="onServer"` attribute)	`cfinput` and cftextarea tags	On the server when ColdFusion gets the submitted form	ColdFusion checks submitted data for validity and runs a validation error page if the data is not valid. You can use the `cferror` tag to specify the validation error page.
hidden field	All Forms, including HTML-only forms	On the server when ColdFusion gets the submitted form	ColdFusion uses the same validation logic as with onServer validation, but you must create additional, hidden, fields and you can use this technique with HTML tags or CFML tags.For detailed information on using hidden fields, see Validating form data using hidden fields.

© 2014 Adobe Systems Incorporated. All rights reserved.

JavaScript(`onValidate="function"` attribute)	`cfgrid`, `cfinput`, `cfslider`, `cftextarea`, and `cftree` tags in HTML and XML format forms	On the client, when the user clicks Submit, before field-specific onSubmit validation	ColdFusion includes the specified JavaScript function in the HTML page it sends to the browser, and the browser calls it. For detailed information on using JavaScript for validation, see Validating form input and handling errors with JavaScript.
IsValid function	ColdFusion variables	On the server, when the function executes	ColdFusion tests the variable to determine whether it follows a specified validation rule and the function returns true or false.For more information on using the `IsValid` function for validation, see Validating data with the IsValid function and the cfparam tag.
cfparam tag	ColdFusion variables	On the server, when the tag executes	ColdFusion checks the specified variable. If the value does not meet the validation criteria, ColdFusion generates an expression exception.For more information on using the `cfparam` tag for validation, see, Validating data with the IsValid function and the cfparam tag.
`cfargument` tag	UDF and CFC function arguments	On the server, when a function is called or invoked	ColdFusion checks the argument value when it is passed to the function. If the value does not meet the validation criteria, ColdFusion generates an application exception.For more information on using the `cfargument` tag, see Writing and Calling User-Defined Functions.

© 2014 Adobe Systems Incorporated. All rights reserved.

> **Note**
>
> For more information on ColdFusion error handling, see Handling Errors.

Selecting a validation technique

The following considerations affect the validation technique that you select:

- If you are validating form data, the techniques you use can vary depending on whether you are using HTML, Flash, or XML forms; for example, different form types have different validation limitations.
- Different validation techniques are appropriate for different form controls and data types.
- Available techniques vary depending on when and where you want the data validated; on the client or the server, when the user enters data or submits a form, or when ColdFusion processes a variable or function argument.
- Each technique has specific features and considerations, such as the form of user feedback, feature limitations, and so on.
- Security issues or concerns that apply to your environment or application can affect the technique you select. The table in the preceding section described some of the considerations (see *Validation techniques* above). The following table describes additional considerations for selecting a validation technique. For additional considerations that are specific to form fields, see Validation type considerations.

Validation technique	Features	Considerations	Security issues
mask(`mask` attribute)	Directly controls user input.	Limited to `cfinput` tags. Provides limited control over user input patterns.	In HTML and XML format, can be circumvented because JavaScript runs directly in the browser.
onBlur(`validateat="onBlur"` attribute)	Provides immediate feedback if a user enters invalid data.	Limited to `cfinput` and `cftextarea` tags. In HTML or XML format, requires the browser to enable JavaScript.	In HTML and XML format, can be circumvented because JavaScript runs directly in the browser.
onSubmit(`validateat="onSubmit"` attribute)	All entered data is available to the user; only the invalid data needs reentering.	Limited to `cfinput` and `cftextarea` tags. In Flash format, is identical to onBlur. In HTML or XML format, validates after all fields have been entered, and requires the browser to enable JavaScript.	In HTML and XML format, can be circumvented because JavaScript runs directly in the browser.
onServer(`validateat="onServer"` attribute)	Does not require browser support.	Limited to `cfinput` and `cftextarea` tags.	Can be circumvented because validation rules are submitted with the form.

© 2014 Adobe Systems Incorporated. All rights reserved.

Hidden form field	Does not require browser support. Can be used with HTML or CFML form elements.	Limited to forms.	Can be circumvented because validation rules are submitted with the form.
JavaScript(`onValidate = "function"` attribute)	Allows all on-client processing supported by the browser. Can be used with HTML or CFML form elements.	Limited to specific ColdFusion form tags. Calls a single JavaScript function. JavaScript levels of support can vary among browsers, and users can disable JavaScript in their browsers.	Can be circumvented because JavaScript runs directly in the browser.
`IsValid` function	Can be used for any variable, not just form fields. Returns a Yes or No result that you use to determine further processing.	When used with a form field, runs after the data is submitted. Must be used each time a variable needs to be validated. Provides some data type checks not available in forms validation techniques.	None
`cfparam` tag	Can be used for any variable, not just form fields. The tag can set a default value in addition to validating data.	When used with a form field, the tag runs after the data is submitted. You respond to validation failures using error-handling code.	None
`cfargument` tag	Used for arguments to functions written using the `cffunction` tag.	Runs when the function is called on the server. You respond to validation failures using error-handling code.	None

Security considerations

Although form-specific validation techniques provide good methods for preventing users from submitting invalid or badly formatted data, they cannot prevent users from submitting maliciously formatted data from HTML forms. Malicious users can circumvent validation techniques that require validation on the browser using JavaScript or submission of validation rules in hidden fields. If you must use a technique for preventing malicious data submissions, consider using the following techniques:

- The `onSubmit` or `OnBlur` validation in Flash forms, which use Flash built-in validation.
- The `IsValid` function and the `cfparam`, and `cfargument` tags, which let you test variables and arguments in your CFML code.
- The `cfqueryparam` tag in `cfquery` tags, which can help protect databases from malicious query input (see Enhancing security with cfqueryparam.
- The script protection option, which helps prevent cross-site scripting attacks. You can set this option on the

© 2014 Adobe Systems Incorporated. All rights reserved.

ColdFusion Administrator Server Settings > Settings page or by using the Application.cfc This.scriptProtect variable or the cfapplication tag `scriptprotect` attribute. For more information on cross-site scripting attacks and this option, see the cfapplication tag page in the *CFML Reference*.

Data validation types

The following table lists the types of data you can validate when you use most ColdFusion validation techniques. It does not include mask validation. Some validation types are not available for all techniques; in these cases the table indicates the limitations. The `onBlur` and `onSubmit` validation algorithms for Flash forms can vary from the validation algorithms described in the following table, and most commonly have less functionality. The asterisk ⭐ in the Type field column indicates that the field is required. For more detailed descriptions of the onServer validation algorithms, see the table in Validating form data using hidden fields.

Type field	Description
date	When validating on the server, allows any date/time format that returns true in the `IsDate` function, including a time value. When validating on the client, same as USdate.
USdate *	A U.S. date of the format mm/dd/yy, with 1- or 2-digit days and months, and 1-through 4-digit years. The separators can be slash (\/), hyphen (-), or period (\.) characters
eurodate *	A date of the format dd/mm/yy, with 1- or 2-digit days and months, and 1- through 4-digit years. The separators can be slash (\/), hyphen (-), or period (\.) characters.
time *	When validating on the server, allows any date/time format that returns True in the `IsDate` function, including a date value. When validating on the client, allows a time of format hh:mm[:ss] [A/PM].
float *	A number; allows integers. When validating form fields on the server, integer values are converted to real numbers.
numeric	A number; allows integers. When validating form fields on the server, integer values are unchanged.
integer *	An integer.
range *	A numeric range specified by a `range` attribute or `max` and `min` attributes.
boolean	A value that can be converted to a Boolean value: Yes, No, True, or False (all case-independent), or a number.

© 2014 Adobe Systems Incorporated. All rights reserved.

telephone *	Standard U.S. telephone formats. Allows an initial 1 long-distance designator and up to 5-digit extensions, optionally starting with x.
zipcode *	U.S. 5- or 9-digit ZIP code format #####-####. The separator can be a hyphen (-) or a space.
creditcard *	Strips blanks and dashes; verifies number using mod10 algorithm. The number must have 13-16 digits.
ssn * or social_security_number *	US. Social Security number format, #########. The separator can be a dash (-) or a space.
email *	A valid e-mail address of the form name@server.domain. ColdFusion validates the format only; it does not check that entry is a valid active e-mail address.
URL *	A valid URL pattern; supports http, https, ftp file, mailto, and news URLs.
guid *	A unique identifier that follows the Microsoft/DCE format, xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx, where x is a hexadecimal number.
uuid *	A universally unique identifier (UUID) that follows the ColdFusion format, xxxxxxxx-xxxx-xxxx-xxxxxxxxxxxxxxxx, where x is a hexadecimal number.
regex * or regular_expression *	Matches the value against a regular expression specified in a `pattern` attribute. Valid in HTML and XML format only; ignored in Flash format.

Note

For more details on how ColdFusion handles data when it does onServer and hidden field validation, see Validating form data using hidden fields.

The following validation types can only be used in `cfinput` tags:

Type	Description
maxlength	Limits the input to a maximum number of characters specified by a `maxlength` attribute.

© 2014 Adobe Systems Incorporated. All rights reserved.

noblanks	Does not allow fields that consist only of blanks. ColdFusion uses this validation only if the `required` at tribute is True.
SubmitOnce	Used only with `cfform` `submit` and `image` types; prevents the user from submitting the same form multiple times before until the next page loads, Use this attribute, for example, to prevent a user from submitting an order form a second time before getting the confirmation for the initial order, and thereby making a duplicate order, Valid in HTML and XML format only; ignored in Flash format.

You can use the following validation types in `cfparam` and `cfargument` tags and the `IsValid` function only:

Type	Description
any	Any type of value
array	An array of values
binary	A binary value
query	A query object
string	A string value or single character
struct	A structure
variableName *	A string formatted according to ColdFusion variable naming conventions.

© 2014 Adobe Systems Incorporated. All rights reserved.

Handling invalid data

How you handle invalid data depends on the validation type. The information here describes validation error-handling rules and considerations. For detailed information on error handling in ColdFusion, including invalid data handling, see Handling Errors.

1. For onBlur, onSubmit, or onServer validation, you can use the `cfinput` or `cftextarea` tag's `message` attribute to specify a text-only error message to display. Otherwise, ColdFusion uses a default message that includes the name of the form field that was invalid. (For OnServer validation, you can customize this message, as described in *Handling form field validation errors* in Determining error-handling strategies.) The following example displays an error message when the user enters an invalid e-mail address:

```
E-mail: <cfinput type="text" size="25" name="email"
  validate="email" message="You must enter a valid e-mail address.">
```

2. For hidden form validation, you can specify a text-only error message in the hidden field's `value` attribute. Otherwise, ColdFusion uses a default message that includes the name of the form field that was invalid. (You can customize this message, as described in *Handling form field validation errors* in Determining error-handling strategies.) The following `cfinput`tag, for example, uses a hidden field validation to display an error message if the user enters an invalid address. (It uses onServer validation to display a different error message if the user fails to enter a number.)

```
Telephone: <cfinput type="text" size="20" name="telephone"
 validateat="onServer" required="Yes"
 message="You must enter a telephone number">
 <cfinput type="hidden" name="telephone_cfformtelephone"
 value="The number you entered is not in the correct format.<br>Use a
 number such as (617) 555-1212, 617-555-1212, or 617-555-1212 x12345">
```

3. For HTML and XML format forms (using ColdFusion skins), most ColdFusion form tags have an `onError` attribute that lets you specify a Javascript function to run if an onSubmit error occurs.
4. For the `IsValid`function, you write separate code paths to handle valid and invalid data. The following example shows a simplified case that displays an error message if the user entered an invalid e-mail address, or a different message if the address is valid:

```
<cfif IsValid("email", custEmail)>
 Thank you for entering a valid address.
 <!--- More processing would go here. --->
<cfelse>
 You must enter a valid e-mail address.<br>
 Click the Back button and try again.
</cfif>
```

5. For `cfparam` and `cfargument` tags, you use standard ColdFusion error-handling techniques. You can include the tag in a `try` block and use a `catch` block to handle the error, or you can use a custom error-handling page. The following example form action page code uses a custom error page, expresserr.cfm, to handle the error that the `cfparam`tag generates if a user submits a form with an invalid e-mail address:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cferror type="EXCEPTION" exception="expression" template="expresserr.cfm">
<cfif IsDefined("form.fieldnames")>
 <cfparam name="form.custEmail" type="email">
 <!--- Normal form processing code goes here. --->
</cfif>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

Masking form input values

The `cfinput` tag `mask` attribute controls the format of data that can be entered into a `text` or `datefield` input field. You can also use a mask attribute in the `cfcalendar` tag. You can combine masking and validation on a field.

- In HTML and Flash form format, a mask can control the format of data entered into a `text` field.
- In the `cfcalendar` tag, and, for Flash format forms, the `datefield` type `cfinput` field, a mask can control the format of the date that ColdFusion uses for the date a user chooses in the displayed calendar.

> **Note**
>
> The standard ColdFusion XML skins do not support masking.

Masking text input

In text fields, ColdFusion automatically inserts literal mask characters, such as - characters in telephone numbers. Users type only the variable part of the field. You can use the following characters to mask data:

Mask character	Effect
A	Allows an uppercase or lowercase character: A-Z and a-z.
X	Allows an uppercase or lowercase character or number: A-Z, a-z, and 0-9.
9	Allows a number: 0-9.
?	Allows any character.
All other characters	Automatically inserts the literal character.

The following pattern enforces entry of a part number of the format EB-1234-c1-098765, where the user starts the entry by typing the first numeric character, such as 1. ColdFusion fills in the preceding EB prefix and all hyphen (-) characters. The user must enter four numbers, followed by two alphanumeric characters, followed by six numbers.

```
<cfinput type="text" name="newPart" mask="EB-9999-XX-999999" />
```

> **Note**
>
> You cannot force a user to type an A, X, 9, or question mark (?) character. To ensure that a pattern is all-uppercase or all-lowercase, use the ColdFusion UCase or LCase functions in the action page.

Masking cfcalendar and datefield input

In the `cfcalendar` tag and the Flash format `datefield` input control, you use the following masks to determine the format of the output. You can use uppercase or lowercase characters in the mask:

© 2014 Adobe Systems Incorporated. All rights reserved.

Mask	Pattern
D	Single- or double-digit day of month, such as 1 or 28
DD	Double-digit day of month, such as 01 or 28
M	Single- or double-digit month, such as 1 or 12
MM	Double-digit month, such as 01 or 12
MMM	Abbreviated month name, such as Jan or Dec
MMMM	Full month name, such as January or December
YY	Two-character year, such as 05
YYYY	Four-character year, such as 2005
E	Single-digit day of week, in the range 0 (Sunday)-6 (Saturday)
EEE	Abbreviated day of week name, such as Mon or Sun
EEEE	Full month day of week name, such as Monday or Sunday

The following pattern specifies that the Flash form sends the date selected using a `datefield` input control to ColdFusion as text in the format 04/29/2004:

```
<cfinput name="startDate" type="datefield" label="date:" mask="mm/dd/yyyy"/>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

Validating data with the IsValid function and the cfparam tag

The `IsValid` function and `cfparam` tag validate any ColdFusion variable value, not just forms variables. Because they reside entirely on the ColdFusion server, they can provide a secure mechanism for ensuring that the required validation steps get performed. Users cannot evade any of the checks by modifying the form data that gets submitted. These techniques also provide greater flexibility in how you respond to user errors, because you can use full CFML syntax in your error-handling code.
These two validation techniques operate as follows:

- The `IsValid` function tests the value of a ColdFusion variable. If the value is valid, it returns True; if the value is invalid, it returns False.
- The `cfparam` tag with a `type` attribute tests the value of a ColdFusion value for validity. If the value is valid, it does nothing; if the value is invalid, it throws a ColdFusion expression exception.
  You can use either technique interchangeably. The technique you choose should depend on your coding style and programming practices. It can also depend on the specific information that you want to display if an error occurs.

Example: IsValid function validation

The following example checks whether a user has submitted a numeric ID and a valid e-mail address and phone number. If any of the submitted values does not meet the validation test, the page displays an error message.

```
<!--- Action code. First make sure the form was submitted. --->
<cfif isDefined("form.saveSubmit")>
 <cfif isValid("integer", form.UserID) and isValid("email", form.emailAddr)
   and isValid("telephone", form.phoneNo)>
  <cfoutput>
   <!--- Application code to update the database goes here --->
   <h3>The e-mail address and phone number for user #Form.UserID#
    have been added</h3>
  </cfoutput>
 <cfelse>
  <H3>Please enter a valid user ID, phone number, and e-mail address.</H2>
 </cfif>
 <cfelse>
</cfif>

<!--- The form. --->
<cfform action="#CGI.SCRIPT_NAME#">
 User ID:<cfinput type="Text" name="UserID"><br>
 Phone: <cfinput type="Text" name="phoneNo"><br>
 E-mail: <cfinput type="Text" name="emailAddr"><br>
 <cfinput type="submit" name="saveSubmit" value="Save Data"><br>
</cfform>
```

Examples: cfparam tag validation

The following two examples use `cfparam` tags to do the same tests as in the [Example: IsValid function validation]. They check whether a user has submitted a numeric ID and a valid e-mail address and phone number. If any of the submitted values does not meet the validation test, ColdFusion throws an expression exception.
In the first example, the error is handled by the exprerr.cfm page specified in the `cferror` tag. In this case, if the user made multiple errors, ColdFusion lists only one.
In the second example, each invalid field is handled in a separate try/catch block. In this case, the user gets information about each error.

© 2014 Adobe Systems Incorporated. All rights reserved.

Using an error-handling page

The self-posting form and action page looks as follows:

```
<!--- Action part of the page. --->
<!--- If an expression exception occurs, run the expresser.cfm page. --->
<cferror type="EXCEPTION" exception="expression" template="expresserr.cfm">
<!--- Make sure the form was submitted. --->
<cfif isDefined("form.saveSubmit")>
<!--- Use cfparam tags to check the form field data types. --->
  <cfparam name="form.emailAddr" type="email">
  <cfparam name="form.UserID" type="integer">
  <cfparam name="form.phoneNo" type="telephone">
 <!--- Application code to update the database goes here. --->
 <cfoutput>
  <h3>The e-mail address and phone number for user #Form.UserID#
   have been added</h3>
 </cfoutput>
</cfif>

<!--- The form. --->
<cfform action="#CGI.SCRIPT_NAME#">
 User ID:<cfinput type="Text" name="UserID"><br>
 Phone: <cfinput type="Text" name="phoneNo"><br>
 E-mail: <cfinput type="Text" name="emailAddr"><br>
 <cfinput type="submit" name="saveSubmit" value="Save Data"><br>
</cfform>
```

The expresserr.cfm page looks as follows:

```
<cfoutput>
 You entered invalid data.<br>
 Please click the browser Back button and try again<br>
 #cferror.RootCause.detailMessage#
</cfoutput>
```

Using cftry and cfcatch tags

The self-posting form and action page looks as follows:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<!--- Use a Boolean variable to indicate whether all fields are good. --->
<cfset goodData="Yes">
<!--- Make sure the form was submitted. --->
<cfif isDefined("form.saveSubmit")>
<!--- The cftry block for testing the User ID value. --->
 <cftry>
<!--- The cfparam tag checks the field data types. --->
  <cfparam name="form.UserID" type="integer">
<!--- If the data is invalid, ColdFusion throws an expression exception. --->
<!--- Catch and handle the exception. --->
  <cfcatch type="expression">
   <!--- Set the data validity indicator to False. --->
   <cfset goodData="No">
   <cfoutput>
    Invalid user ID.<br>
    #cfcatch.detail#<br><br>
   </cfoutput>
  </cfcatch>
 </cftry>
<!--- The cftry block for testing the e-mail address value. --->
 <cftry>
  <cfparam name="form.emailAddr" type="email">
  <cfcatch type="expression">
   <cfset goodData="No">
   <cfoutput>
    Invalid e-mail address.<br>
    #cfcatch.detail#<br><br>
   </cfoutput>
  </cfcatch>
 </cftry>
<!--- The cftry block for testing the telephone number value. --->
 <cftry>
  <cfparam name="form.phoneNo" type="telephone">
  <cfcatch type="expression">
   <cfset goodData="No">
   <cfoutput>
    Invalid telephone number.<br>
    #cfcatch.detail#<br><br>
   </cfoutput>
  </cfcatch>
 </cftry>
 <!--- Do this only if the validity indicator was not set to False in a
   validation catch block. --->
 <cfif goodData>
  <!--- Application code to update the database goes here. --->
  <cfoutput>
   <h3>The e-mail address and phone number for user #Form.UserID#
    have been added</h3>
  </cfoutput>
 </cfif> <!--- goodData is True--->
</cfif> <!--- Form was submitted. --->

<cfform action="#CGI.SCRIPT_NAME#" preservedata="Yes">
 User ID:<cfinput type="Text" name="UserID"><br>
 Phone: <cfinput type="Text" name="phoneNo"><br>
 E-mail: <cfinput type="Text" name="emailAddr"><br>
 <cfinput type="submit" name="saveSubmit" value="Save Data"><br>
</cfform>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

#back to top

© 2014 Adobe Systems Incorporated. All rights reserved.

Validating form data using hidden fields

ColdFusion lets you specify form field validation on the server by using hidden form fields whose names consist of the name of the field to validate and the validation type. Hidden field validation uses the same underlying techniques and algorithms as onServer validation of ColdFusion form fields.
Hidden field validation has the following features:

- You can use it with standard HTML tags. For example, you can validate data in an HTML `input` tag. This feature was useful in releases previous to ColdFusion MX 7, because the `cfinput` tag did not support all HTML `type` attributes.
- It is backward-compatible with validation previous to ColdFusion MX 7, when hidden field validation was the only way to do validation on the server.
- Because you use a separate tag for each validation type, if you specify multiple validation rules for a field, you can specify a different error message for each rule.
- You can use hidden field validation with any form field type that submits a data value, not `input`, `cfinput`, `textarea`, or `cftextarea`.

Specifying hidden form field validation

To specify hidden field validation, you do the following:

- Create one HTML `input` element or CFML `cfinput` tag of `type="hidden"` for each validation rule.
- Specify the name of the field to validate as the first part of the hidden field name.
- Specify the type of validation, starting with an underscore character (_), as the second part of the hidden field name.
- You can specify multiple rules for each form data field. For example, to specify range and required validation for a field named myValue, create hidden myValue_cfformrange and myValue_cfformrequired fields.
- For most types of validation, specify the error message as the field `value` attribute.
- For range, maximum length, or regular expression validation, specify the rule, such as the maximum length, in the `value` attribute. For these validation types, you cannot specify a custom error message.
  The following example uses hidden fields to require data in a date field and ensure that the field contains a date. It consists only of HTML tags.

```
<input type="text" name="StartDate" size="16" maxlength="16"><br>
<input type="hidden" name="StartDate_required"
  value="You must enter a start date.">
<input type="hidden" name="StartDate_date"
 value="Please enter a valid date as the start date.">
```

Use the following suffixes at the end of hidden form field names to specify the validation type. The type identifier always starts with an underscore. Several validation rules have two names you can use. The names that do not start with "_cf" were used in earlier releases and are retained for backward compatibility. For consistency and clarity, Adobe recommends using the names that start with "_cf" in new forms.

Field name suffix	Verifies
_integer, _cfforminteger	An integer of the range -2,147,483,648 -2,147,483,647. Treats the initial characters "$ ¤ ¥ £ +" as valid input, but removes them from the number.

© 2014 Adobe Systems Incorporated. All rights reserved.

_cfformnumeric	Any numeric value. Treats the initial characters "$ ¤ ¥ £ +" as valid input, but does NOT remove them from the number.
_float, _cfformfloat	Any value (including an integer) that can be represented as a floating point number with up to 12 significant digits. Treats the initial characters "$ ¤ ¥ £ +" as valid input, but removes them from the number. Converts input data to a real number; for example a dump of an integer value on the action page includes a decimal point followed by a 0.
_range, _cfformrange	A numeric value within boundaries specified by the `value` attribute. Specify the range in the value attribute using the format "min=minvalue max=maxvalue." You cannot specify a custom error message for this validation.
_date, _cfformdate	A date in any format that ColdFusion can understand; converts the input to ODBC date format. Allows entry of a time part, but removes it from the ODBC value.
_cfformusdate	A date in the form m/d/y, m-d-y , or m.d.y, The m and d format can be 1 or 2 digits; y can be 2 or 4 digits. Does not convert the string to an ODBC value and does not allow a time part.
_eurodate, _cfformeurodate	A date in the form d/m/y, d-m-y, or d.m.y. The m and d format can be 1 or 2 digits; y can be 2 or 4 digits. Converts the input to ODBC date format. Allows entry of a time part, but removes it from the ODBC value.
_time, _cfformtime	A time. Can be in 12-hour or 24-hour clock format, and can include seconds in the form hh:mm:ss or a case-independent am or pm indicator.Converts the input to ODBC time format. Allows entry of a date part, but removes it from the ODBC value.
_cfformcreditcard	After stripping blanks and dashes, a number that conforms to the mod10 algorithm. Number must have 13-16 digits.
_cfformSSN, _cfformsocial_security_number	A nine-digit Social Security number. Can be of the form xxx-xx-xxxx or xxx xx xxxx.

© 2014 Adobe Systems Incorporated. All rights reserved.

_cfformtelephone	Standard U.S. telephone formats. Does not support international telephone numbers.Allows area codes with or without parentheses, and hyphens (-), spaces, periods, or no separators between standard number groups. Can be preceded by a 1 long-distance designator, and can end with an up-to-5 digit extension, optionally starting with x. The area code and exchange must begin with a digit in the range 1 - 9.
_cfformzipcode	A 5-digit or 9-digit U.S. ZIP code. In 9-digit codes, precede the final four digits by a hyphen (-) or space.
_cfformemail	A valid e-mail address. Valid address characters are a-zA-Z0-9_- and the period and separator. There must be a single at sign (@) and the text after the @ character must include a period, as in my_address@MyCo.com or b-b.jones27@hisco.co.uk.
_cfformURL	A valid URL. Must start with http: , https:\, ftp:\, file:\ , mailto: , or news:. Can include, as appropriate, user name and password designators and query strings. The main part of the address can only have the characters A-Za-z0-9 and -.
_cfformboolean	A value that can be treated as a Boolean: Yes, No, True, False, 0, 1.
_cfformUUID	A universally unique identifier (UUID) that follows the ColdFusion format, xxxxxxxx-xxxx-xxxx-xxxxxxxxxxxxxxxx, where x is a hexadecimal number.
_cfformGUID	A unique identifier that follows the Microsoft/DCE format, xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx, where x is a hexadecimal number.
_cfformnoblanks	The field must not include blanks. ColdFusion uses this validation only if you also specify a `_required` hidden field.
_cfformmaxlength	The number of characters must not exceed the number specified by the tag `value` attribute.

© 2014 Adobe Systems Incorporated. All rights reserved.

_cfformregex, _cfformregular_expression	The data must match a JavaScript regular expression specified by the tag `value` attribute.
_required, _cfformrequired	Data must be entered or selected in the form field.

Hidden form field considerations

Consider the following rules and recommendations when determining whether and how to use hidden form field validation:

- Use hidden field validation if you want to validate data from standard HTML `input` tags. The `cfinput` and `cftextarea` tags include a `validateAt` attribute that provides a simpler method for specifying server-side validation.
- Consider using hidden field validation with the `cfinput` and `cftextarea` tags if you specify multiple validation rules for a single field and want to provide a separate error message for each validation.
- Do not use the suffixes listed in the table as field names.
- Adding a validation rule to a field does not make it a required field. Add a separate `_required` hidden field to ensure user entry.

Hidden form field example

The following procedure creates a simple form for entering a start date and a salary. It uses hidden fields to ensure that you enter data and that the data is in the right format.
This example uses a self-submitting form; the same ColdFusion page is both the form page and its action page. The page uses an `IsDefined` function to check that form data has been submitted. This way, the pages does not show any results until you submit the input. The form uses HTML tags only; you can substitute these tags with the CFML equivalents.

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<html>
<head>
 <title>Simple Data Form</title>
</head>
<body>
<h2>Simple Data Form</h2>

<!--- Form part. --->
<form action="datatest.cfm" method="Post">
 <input type="hidden"
  name="StartDate_cfformrequired"
  value="You must enter a start date.">
 <input type="hidden"
  name="StartDate_cfformdate"
  value="Enter a valid date as the start date.">
 <input type="hidden"
  name="Salary_cfformrequired"
  value="You must enter a salary.">
 <input type="hidden"
  name="Salary_cfformfloat"
  value="The salary must be a number.">
 Start Date:
 <input type="text"
  name="StartDate" size="16"
  maxlength="16"><br>
 Salary:
 <input type="text"
  name="Salary"
  size="10"
  maxlength="10"><br>
 <input type="reset"
  name="ResetForm"
  value="Clear Form">
 <input type="submit"
  name="SubmitForm"
  value="Insert Data">
</form>
<br>

<!--- Action part. --->
<cfif isdefined("Form.StartDate")>
 <cfoutput>
  Start Date is: #DateFormat(Form.StartDate)#<br>
  Salary is: #DollarFormat(Form.Salary)#
 </cfoutput>
</cfif>
</html>
```

When the user submits this form, ColdFusion scans the form fields to find any validation rules. It then uses the rules to analyze the user's input. If any of the input rules are violated, ColdFusion displays an error page with the error message that you specified in the hidden field's `value` attribute. The user must go back to the form, correct the problem, and resubmit the form. ColdFusion does not accept form submission until the user enters the entire form correctly.

Because numeric values often contain commas and currency symbols, ColdFusion automatically deletes these

© 2014 Adobe Systems Incorporated. All rights reserved.

characters from fields with _cfforminteger and _cfformfloat rules before it validates the form field and passes the data to the form's action page. ColdFusion does not delete these characters from fields with _cfformnumeric rules.

Reviewing the code

The following table describes the code and its function:

Code	Description
`<form action="datatest.cfm" method="Post">`	Gathers the information from this form sends it to the dataform.cfm page (this page) using the Post method.
`<input type="hidden" name="StartDate_cfformrequired" value="You must enter a start date.">` `<input type="hidden" name="StartDate_cfformdate" value="Enter a valid date as the start date.">`	Requires input into the StartDate input field. If there is no input, displays the error information "You must enter a start date." Requires the input to be in a valid date format. If the input is not valid, displays the error information "Enter a valid date as the start date."
`<input type="hidden" name="Salary_cfformrequired" value="You must enter a salary.">` `<input type="hidden" name="Salary_cfformfloat" value="The salary must be a number.">`	Requires input into the Salary input field. If there is no input, displays the error information "You must enter a salary." Requires the input to be in a valid number. If it is not valid, displays the error information "The salary must be a number."
`Start Date:` `<input type="text" name="StartDate" size="16" maxlength="16"> `	Creates a text box called StartDate in which users can enter their starting date. Makes it 16-characters wide.
`Salary:` `<input type="text" name="Salary" size="10" maxlength="10"> `	Creates a text box called Salary in which users can enter their salary. Makes it ten-characters wide.

© 2014 Adobe Systems Incorporated. All rights reserved.

`<cfif isdefined("Form.StartDate")>` `  <cfoutput>` ` Start Date is:` `#DateFormat(Form.StartDate)# ` ` Salary is:` `#DollarFormat(Form.Salary)#` ` </cfoutput>` `</cfif>`	Displays the values of the StartDate and Salary form fields only if they are defined. They are not defined until you submit the form, so they do not appear on the initial form. Uses the DateFormat function to display the start date in the default date format. Uses the DollarFormat function to display the salary with a dollar sign and commas.

© 2014 Adobe Systems Incorporated. All rights reserved.

Validating form data with regular expressions

You can use *regular expressions* to match and validate the text that users enter in `cfinput` and `cftextinput` tags. Ordinary characters are combined with special characters to define the match pattern. The validation succeeds only if the user input matches the pattern.

Regular expressions let you check input text for a wide variety of custom conditions for which the input must follow a specific pattern. You can concatenate simple regular expressions into complex search criteria to validate against complex patterns, such as any of several words with different endings.

You can use ColdFusion variables and functions in regular expressions. The ColdFusion server evaluates the variables and functions before the regular expression is evaluated. For example, you can validate against a value that you generate dynamically from other input data or database values.

Note: The rules listed here are for JavaScript regular expressions, and apply to the regular expressions used inc `finputandcftextinputtags` only. These rules differ from the rules used by the ColdFusion functions `REFind`,`RERepl ace`,`REFindNoCase`, and `REReplaceNoCase`. For information on regular expressions used in ColdFusion functions, see Using Regular Expressions in Functions.

Special characters

Because special characters are the operators in regular expressions, to represent a special character as an ordinary one, escape it by preceding it with a backslash. For example, use two backslash characters (\\) to represent a backslash character.

Single-character regular expressions

The following rules govern regular expressions that match a single character:

- Special characters are:`+ * ? . [ ^ $ ( ) { | \`
- Any character matches itself if it is not a special character or if a preceding backslash (\) escapes the character.
- A backslash (\) followed by any special character matches the literal character itself; that is, the backslash escapes the special character.
- A period (.) matches any character except newline.
- A set of characters enclosed in brackets ([]) is a one-character regular expression that matches any of the characters in that set. For example, “[akm]” matches an *a*, *k*, or *m*. If you include] (closing square bracket) in square brackets, it must be the first character. Otherwise, it does not work, even if you use \].
- A dash can indicate a range of characters. For example, [a-z] matches any lowercase letter.
- If the first character of a set of characters in brackets is the caret (^), the expression matches any character except those characters in the set. It does not match the empty string. For example:“[^akm]” matches any character except *a*, *k*, or *m*. The caret loses its special meaning if it is not the first character of the set.
- You can make regular expressions case insensitive by substituting individual characters with character sets; for example, “[Nn][Ii][Cc][Kk]”is a case-insensitive pattern for the name Nick (or NICK, or nick, or even nIcK).

© 2014 Adobe Systems Incorporated. All rights reserved.

- You can use the following escape sequences to match specific characters or character classes:

Escape seq	Matches	Escape seq	Meaning
[\b]	Backspace.	\s	Any of the following white-space characters: space, tab, form feed, and line feed.
\b	A word boundary, such as a space.	\S	Any character except the white-space characters matched by \s.
\B	A nonword boundary.	\t	Tab.
\cX	The control character Ctrl-x. For example, \cv matches Ctrl-v, the usual control character for pasting text.	\v	Vertical tab.
\d	A digit character [0-9].	\w	An alphanumeric character or underscore. The equivalent of [A-Za-z0-9_].
\D	Any character except a digit.	\W	Any character not matched by \w. The equivalent of [^A-Za-z0-9_].
\f	Form feed.	\n	Backreference to the nth expression in parentheses. See Backreferences.
\n	Line feed.	\ooctal	The character represented in the ASII character table by the specified octal number.
\r	Carriage return.	\\x*hex*	The character represented in the ASCII character table by the specified hexadecimal number.

Multicharacter regular expressions

Use the following rules to build a multicharacter regular expression:

© 2014 Adobe Systems Incorporated. All rights reserved.

- Parentheses group parts of regular expressions into a subexpression that can be treated as a single unit. For example,“(ha)+” matches one or more instances of *ha*.
- A one-character regular expression or grouped subexpression followed by an asterisk (*) matches zero or more occurrences of the regular expression. For example, “[a-z]*” matches zero or more lowercase characters.
- A one-character regular expression or grouped subexpression followed by a plus sign (+) matches one or more occurrences of the regular expression. For example, “[a-z]+” matches one or more lowercase characters.
- A one-character regular expression or grouped subexpression followed by a question mark (?) matches zero or one occurrence of the regular expression. For example, “xy?z” matches either *xyz* or *xz*.
- The carat (^) at the beginning of a regular expression matches the beginning of the field.
- The dollar sign ($) at the end of a regular expression matches the end of the field.
- The concatenation of regular expressions creates a regular expression that matches the corresponding concatenation of strings. For example, “[A-Z][a-z]*” matches any capitalized word.
- The OR character (|) allows a choice between two regular expressions. For example, “jell(y|ies)” matches either *jelly* or *jellies*.
- Curly brackets ({}) indicate a range of occurrences of a regular expression. You use them in the form “{m, n}” where *m* is a positive integer equal to or greater than zero indicating the start of the range and *n* is equal to or greater than *m*, indicating the end of the range. For example, “(ba){0,3}” matches up to three pairs of the expression *ba*. The form “{m,}” requires at least *m* occurrences of the preceding regular expression. The form “{m}” requires exactly *m* occurrences of the preceding regular expression. The form “{,n}” is not allowed.

Backreferences

Backreferencing lets you match text in previously matched sets of parentheses. A slash followed by a digit n (\n) refers to the *n*th parenthesized subexpression.

One example of how you can use backreferencing is searching for doubled words; for example, to find instances of “the the” or “is is” in text. The following example shows backreferencing in a regular expression:

```
(\b[A-Za-z]+)[ ]+\1
```

This code matches text that contains a word that is repeated twice; that is, it matches a word (specified by the \b word boundary special character and the “[A-Za-z]+)” followed by one or more spaces (specified by “[]+”), followed by the first matched subexpression, the first word, in parentheses. For example, it would match “is is”, but not “This is”.

Exact and partial matches

ColdFusion validation normally considers a value to be valid if any of it matches the regular expression pattern. If you want to ensure that the entire entry matches the pattern, “anchor” it to the beginning and end of the field, as follows:

- If a caret (^) is at the beginning of a pattern, the field must begin with a string that matches the pattern.
- If a dollar sign ($) is at the end of a pattern, the field must end with a string that matches the pattern.
- If the expression starts with a caret and ends with a dollar sign, the field must exactly match the pattern.

© 2014 Adobe Systems Incorporated. All rights reserved.

Expression examples

The following examples show some regular expressions and describe what they match:

Expression	Description
[\?&]value=	Any string containing a URL parameter value.
^[A-Z]:(\\[A-Z0-9_]+)+$	An uppercase Windows directory path that is not the root of a drive and has only letters, numbers, and underscores in its text.
^(\+|-)?[1-9][0-9]*$	An integer that does not begin with a zero and has an optional sign.
^(\+|-)?[1-9][0-9]*(\.[0-9]*)?$	A real number.
^(\+|-)?[1-9]\.[0-9]*E(\+|-)?[0-9]+$	A real number in engineering notation.
a{2,4}	A string containing two to four occurrences of *a*: aa, aaa, aaaa; for example, aardvark, but not automatic.
(ba){2,}	A string containing least two *ba* pairs; for example, Ali baba, but not Ali Baba.

An excellent reference on regular expressions is Mastering Regular Expressions by Jeffrey E.F. Friedl, published by O'Reilly & Associates, Inc.

© 2014 Adobe Systems Incorporated. All rights reserved.

Validating form fields

In basic form field validation, do the following:

- Use a `cfinput` or `cftextarea` tag.
- Specify a validation type, such as numeric, or multiple types.
- Optionally, specify an error message.
- Optionally, specify a validation technique. (By default, ColdFusion uses onSubmit validation.)
  The following example specifies onBlur validation of a telephone number:

```
Phone: <cfinput type="text" name="HPhone"
  validateat="onBlur"
  validate="required,telephone"
  message="Please enter a standard U.S. telephone number with an optional
   extension, such as x12345">
```

The following information describes considerations for validation in `cfinput` and `cftextarea` tags, and show a more complete example.

Validation type considerations

General considerations: Consider the following issues when you determine how to validate form data:

- When you validate form data using onBlur, onSubmit, onServer, or hidden form field validation, you can specify one or more validation types for each field that you validate. For example, you can specify that a field entry is required and that it must be numeric. To specify multiple validation types for onSubmit, onBlur, or onServer validation, specify the type values in a comma-delimited list.
- If you use onBlur, onSubmit, or onServer type validation, you can specify only one error message for each field that you validate. If you use hidden field validation, you can create a custom message for each validation rule (except for range checking).
- In the `cfinput` tag, most validation type attributes apply only to text or password fields.
  Validation algorithm differences: The underlying validation code used when validating form data can differ depending on the validation technique and the form type. As a result, the algorithms used vary in some instances, including the following:

- The validation algorithms used for date/time values in onSubmit and OnBlur validation are different from those validation algorithms used for all server-side validation techniques.
- The algorithms used for onSubmit and OnBlur validation in Flash can vary from those algorithms used for HTML or XML format, and generally follow simpler rules.
  For detailed information on the validation algorithms used for validation techniques used on the server, see Validating form data using hidden fields.

Validating data in XML skinnable forms

If you create an XML skinnable form and use any skin provided by Adobe, such as the basic.xsl or silver.xsl skin, you can use all form validation techniques that are available for HTML forms.
If you use a custom skin (XSL file), the available validation techniques depend on the skin. The *cf_webroot*\CFIDE\scripts\xsl directory contains a _cfformvalidation.xsl file that implements all ColdFusion HTML form validation techniques and supports onBlur, onSubmit, onServer, and hidden form field validation. XML skin writers can include this file in their skin XSLT to implement ColdFusion validation for their skin.

Example: basic form validation

The following form asks for information to use when registering a new user. It checks to make sure that the user

© 2014 Adobe Systems Incorporated. All rights reserved.

enters required information. (Only the telephone number is optional.) It also checks to make sure that the telephone number and e-mail address are properly formatted and that the number to use in a challenge question is in the proper range. This example performs onSubmit validation. It posts back to itself, and dumps the submitted results.

```
<cfif IsDefined("form.fieldnames")>
 <cfdump var="#form#"><br>
</cfif>

<cfform name="myform" preservedata="Yes" >
 First Name: <cfinput type="text" size="15" name="firstname"
  required="yes" message="You must enter a first name."><br>
 Last Name: <cfinput type="text" size="25" name="lastname"
  required="yes" message="You must enter a last name."><br>
 Telephone: <cfinput type="text" size="20" name="telephone"
  validate="telephone" message="You must enter your telephone
  number, for example 617-555-1212 x1234"><br>
 E-mail: <cfinput type="text" size="25" name="email"
  validate="email" required="Yes"
  message="You must enter a valid e-mail address."><br>
 Password: <cfinput type="password" size="12" name="password1"
  required="yes" maxlength="12"
  message="You must enter a password."><br>
 Reenter password: <cfinput type="password" size="12" name="password2"
  required="yes" maxlength="12"
  message="You must enter your password twice."><br>
 We will ask you for the following number, in the range 100-999 if you forget
  your password.<br>
 Number: <cfinput type="text" size="5" name="chalenge"
  validate="range" range="100,999" required="Yes"
  message="You must enter a reminder number in the range 100-999."><br>
 <cfinput type="submit" name="submitit">
</cfform>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

Validating form input and handling errors with JavaScript

ColdFusion lets you write your own validation routines in JavaScript, and lets you create JavaScript error handlers.

Validating input with JavaScript

In addition to native ColdFusion input validation using the `validate` attribute of the `cfinput` and `cftextarea` tag s, the following tags support the `onValidate` attribute, which lets you specify a JavaScript function to handle your `cfform` input validation:

- `cfgrid`
- `cfinput`
- `cfslider`
- `cftextarea`
- `cftree`
  ColdFusion passes the following arguments to the JavaScript function that you specify in the `onValidate` att ribute:

- The form JavaScript DOM object
- The `name` attribute of the form element
- The value of the control to validate
  For example, if you write the `cfinput` tag as the following:

```
<cfinput type="text"
 ...
<!--- Do not include () in JavaScript function name. --->
 onvalidate="handleValidation"
 ...
>
```

You define the JavaScript function as the following:

```
<script>
<!--
function handleValidation(form_object, input_object, object_value) {
...
}
//-->
</script>
```

Example: validating a password

The following example validates a password. The password must have at least one of each of the following: an uppercase letter, a lowercase letter, and a number. It must be from 8 through 12 characters long. If the password is invalid, the browser displays a message box. If the password is valid, it redisplays the page with a brief success message.

Use JavaScript to validate form data

1. Create a ColdFusion page with the following content:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<html>
<head>
 <title>JavaScript Validation</title>

<!--- JavaScript used for validation. --->
<script>
<!--
 // Regular expressions used for pattern matching.
 var anUpperCase = /[A-Z]/;
 var aLowerCase = /[a-z]/;
 var aNumber = /[0-9]/;
 /* The function specified by the onValidate attribute.
  Tests for existence of at least one uppercase, lowercase, and numeric
  character, and checks the length for a minimum.
  A maximum length test is not needed because of the cfinput maxlength
  attribute. */
 function testpasswd(form, ctrl, value) {
  if (value.length < 8 || value.search(anUpperCase) == -1 ||
   value.search (aLowerCase) == -1 || value.search (aNumber) == -1)
  {
   return (false);
  }
  else
  {
   return (true);
  }
 }
//-->
</script>
</head>

<body>
<h2>JavaScript validation test</h2>
<!--- Form is submitted only if the password is valid. --->
<cfif IsDefined("Form.passwd1")>
 <p>Your Password if valid.</p>
</cfif>
<p>Please enter your new password:</p>

<cfform name="UpdateForm" preservedata="Yes" >
 <!--- The onValidate attribute specifies the JavaScript validation
    function. The message attribute is the message that appears
    if the validation function returns False. --->
 <cfinput type="password" name="passwd1" required="YES"
  onValidate="testpasswd"
  message="Your password must have 8-12 characters and include uppercase
and lowercase letters and at least one number."
  size="13" maxlength="12">

<input type="Submit" value=" Update... ">
</cfform>

</body>
</html>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

2. Save the page as validjs.cfm.
3. View the validjs.cfm page in your browser.

Handling failed validation

The `onError` attribute lets you specify a JavaScript function to execute if an onValidate, onBlur, or onSubmit validation fails. For example, if you use the `onValidate` attribute to specify a JavaScript function to handle input validation, you can also use the `onError` attribute to specify a JavaScript function to handle a failed validation (that is, when `onValidate` returns a False value). If you use the `onValidate` attribute, you can also use the `onError` attribute to specify a JavaScript function that handles the validation errors. The following `cfform` tags support the `onerror` attribute:

- `cfgrid`
- `cfinput`
- `cfselect`
- `cfslider`
- `cftextinput`
- `cftree`

ColdFusion passes the following JavaScript objects to the function in the `onerror` attribute:

- The JavaScript form object
- The `name` attribute of the form element
- The value that failed validation
- The error message text specified by the CFML tag's `message` attribute

The following example shows a form that uses an `onError` attribute to tell ColdFusion to call a showErrorMessage JavaScript function that uses the `alert` method to display an error message. The function assembles the message from the invalid value and the contents of the `cfinput` tag's `message` attribute.

```
<!--- The JavaScript function to handle errors.
   Puts a message, including the field name and value, in an alert box. --->
<script>
 <!--
 function showErrorMessage(form, ctrl, value, message) {
  alert("The value " + value +" of the " + ctrl + " field " + message);
 }
 //-->
</script>

<!--- The form.
 The cfinput tags use the onError attribute to override the ColdFusion
 default error message mechanism. --->
<cfform>
 <!--- A minimum quantity is required and must be a number. --->
 Minimum Quantity: <cfinput type="Text" name="MinQuantity"
  onError="showErrorMessage" validate="numeric" required="Yes"
  message="is not a number." ><br>
 <!--- A maximum quantity is optional, but must be a number if supplied. --->
 Maximum Quantity: <cfinput type="Text" name="MaxQuantity"
  onError="showErrorMessage" validate="numeric"
   message="is not a number." ><br>
 <cfinput type="submit" name="submitit">
</cfform>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

Creating Forms in Flash

You can create effective forms in Adobe Flash format, in which Adobe ColdFusion displays forms using Flash, not HTML.

© 2014 Adobe Systems Incorporated. All rights reserved.

About Flash forms

ColdFusion can deliver forms to the client in Flash (SWF file) format. ColdFusion automatically generates the Flash binary from your CFML code and displays it on the client. Flash forms have the following advantages over HTML forms:

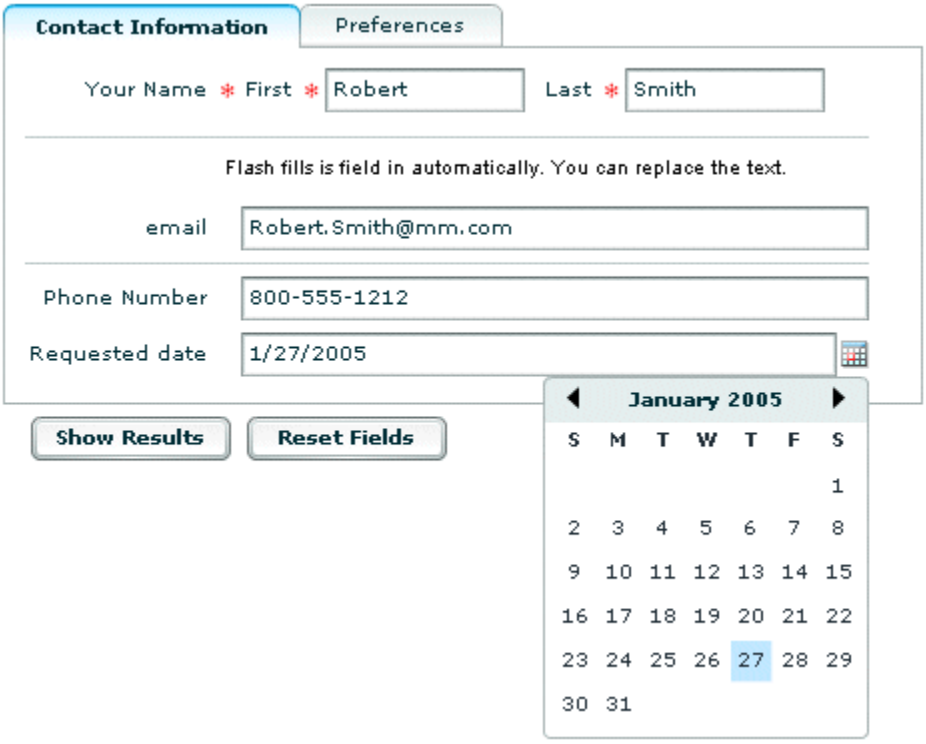
- They are browser-independent. Flash Player works in all commonly used browsers on Windows and Macintosh systems, and in Netscape and Mozilla on Linux.
- By default, they present a modern, visually pleasing appearance, and you can apply predefined color skins or customize the appearance with specifications like those specifications in a Cascading Style Sheet (CSS).
- They let you develop complex, multipart forms that do not require multiple pages, by using tabbed or accordion-style dialog boxes.
- They automatically do much of the layout work for you.

> **Note**
>
> Flash form configuration requirements differ from ColdFusion requirements. For example, Flash forms do not work sometimes with all J2EE servers supported by ColdFusion. For more information, see Installing ColdFusion.

In addition to creating Flash forms, ColdFusion lets you specify Flash format for `cfcalendar`, `cftree`, and `cfgrid` tags. Use these tags to embed Flash calendar choosers, trees, and grids in HTML forms, to eliminate the need to use Java applets. Information about using Flash grids and trees in HTML forms is not discussed here. However, the information about grids and trees also applies to these elements.

A Flash form example

Flash forms provide many features that help you quickly create easy-to-use, professional-looking, complex forms. The following image contains a two-tab form that shows many of these features:



This form includes the following features:

© 2014 Adobe Systems Incorporated. All rights reserved.

- Each tab contains a different section of the overall form, and users can enter data on both tabs before submitting the form. This technique can eliminate the need for multiple forms on multiple HTML pages.
- The first and last names are required fields, indicated by the red asterisks.
- The Flash form automatically fills the e-mail field with data from the name fields, but the user can override this information.
- When the user selects the date field, a calendar automatically opens for picking the date.

Flash form CFML differences from HTML forms

Because ColdFusion sends a Flash form to the client in SWF file format, everything inside a Flash form is rendered by Flash. Rendering the form in Flash has several effects:

- Plain text and HTML tags in the body of a Flash Form have no effect.
- Specify all form content inside CFML tags that support Flash forms.
- ColdFusion provides two tags that let you take advantage of Flash features and perform tasks that you would otherwise do in HTML: use the `cfformitem` tag to add text blocks and horizontal and vertical rules to your form, and you use the `cfformgroup` tag to structure your form.
- Standard ColdFusion forms tags, such as `cfinput` and `cftree`, include attributes that work only with Flash forms, and attribute values that let you specify form style and behavior. These tags include the `skin` attribute with many Flash-specific `style` attribute values for appearance, and the `bind` attribute for filling a field value with data from other fields.The reference pages for the individual tags in the *CFML Reference* describe the form tags and their features, indicating which attributes and values work with Flash forms.

© 2014 Adobe Systems Incorporated. All rights reserved.

Best practices for Flash forms

Minimizing form recompilation

Flash forms are sent to the client as SWF files, which ColdFusion must compile from your CFML code. The following techniques can help limit how frequently ColdFusion must recompile a Flash form.

- Only data must be dynamic. Whenever a variable name changes, or a form characteristic, such as an element width or a label changes, the Flash output must be recompiled. If a data value changes, the output does not need to be recompiled.
- Use `cfformgroup type="repeater"` if you must loop no more than ten times over no more than ten elements. This tag does not require recompiling when the number of elements changes. It does have a processing overhead that increases with the number of loops and elements, however, so for large data sets or many elements, it is often more efficient not to use the repeater.

Caching data in Flash forms

The `cfform` tag `timeout` attribute specifies how many seconds ColdFusion retains Flash form data on the server. When a Flash form is generated, the values for the form are stored in memory on the server. When the Flash form is loaded on the client, it requests these form values from the server. If this attribute is 0, the default, the data on the server is immediately deleted after the data has been requested from the Flash form.
A Flash form can be reloaded multiple times if a user displays a page with a Flash form, goes to another page, and uses the browser Back button to return to the page with the form. This behavior is common with search forms, login forms, and the like. When the user returns to the original page:

- If the `timeout` value is 0, or the time-out period has expired, the data is no longer available, and ColdFusion returns a data-expired exception to the browser; in this case, the browser typically tells the user to reload the page.
- If the time-out has not expired, the browser displays the original data.If your form data contains sensitive information, such as credit card numbers or social security numbers, leave the time-out set to 0. Otherwise, consider setting a time-out value that corresponds to a small number of minutes.

Using Flash forms in a clustered environment

Flash forms require sticky sessions when used in a cluster.

#back to top

© 2014 Adobe Systems Incorporated. All rights reserved.

Binding data in Flash forms

The `bind` attribute lets you set the value of the fields using the contents of other form fields. You can use the `bind` attribute with the `cftextarea` tag and any `cfinput` type that takes a value, including `hidden`. This data binding occurs dynamically as the user enters data within Flash on the client system. Flash does not send any information to ColdFusion until the user submits the form. To use the `bind` attribute to specify the field value, use the following formats:

Data source	bind attribute format
`cfinput type = "text"` or `cftextarea` text	`bind="{`*sourceName*`.text}"`
`cfinput` selected radio button	`bind="{`*sourceName*`.selectedData}"`
`cftree` selected item	`bind="{`*sourceName*`.selectedNode.getProperty('data').value}`
`cfgrid` selected item	`bind="{`*sourceName*`.selectedItem.`*COLUMNAME*`}"`
`cfselect` selected item	`bind="{`*sourceName*`.selectedItem.data}"`

Note

If you use the `bind` attribute, you cannot use the `value` attribute.

The following rules and techniques apply to the binding formats:

- The *sourceName* value in these formats is the name attribute of the tag that contains the element that you are binding to.
- You can bind to additional information about a selected item in a tree. Replace `value` with `display` to get the displayed value, or with `path` to get the path to the node in the tree.
- You can bind to the displayed value of a `cfselect` item by replacing `data` with `label`.
- If the user selects multiple items in a `cfselect` control, the selectedItem object contains the most recent selection, and a selectedItems array contains all selected items. You can access the individual values in the array, as in myTree.selectedItems1.data. The selectedItems array exists only if the user selects multiple items; otherwise, it is undefined.
- You can use ActionScript expressions in Flash bind statements.The following example shows how to use the values from the firstName and lastName fields to construct an e-mail address. The user can change or replace this value with a typed entry.

```
<cfformgroup type="horizontal" label="Your Name">
 <cfinput type="text" required="Yes" name="firstName" label="First"
  value="" width="100"/>
 <cfinput type="text" required="Yes" name="lastName" label="Last"
  value="" width="100"/>
</cfformgroup>
<cfinput type="text" name="email" label="email"
 bind="{firstName.text}.{lastName.text}@mm.com">
```

© 2014 Adobe Systems Incorporated. All rights reserved.

© 2014 Adobe Systems Incorporated. All rights reserved.

Building Flash forms

You build Flash forms using standard ColdFusion form tags, plus the `cfformgroup` and `cfformitem` tags. These tags create the *elements* of the form, as follows:

- The `cfcalendar`, `cfgrid`, `cfinput`, `cfselect`, `cftextarea`, and `cftree` tags create *controls* for data display and user input.
- The `cfformitem` tag lets you add formatted or unformatted text, spacers, and horizontal and vertical rules without using HTML.
- The `cfformgroup` tag creates *containers*, such as horizontally aligned boxes or tabbed navigators, that let you group, organize, and structure the form contents.
  Flash forms follow a hierarchical structure of containers and *children*.

1. The `cfform` tag is the master container, and its contents are child containers and controls.
2. The `cfformgroup` tag defines a container that organizes its child elements.
3. All other tags create individual controls, including display elements such as rules.For example, the image in the About Flash formssection has the following hierarchical structure of containers and children. (This outline only shows the structure of the page that is visible in the image. It omits the structure of the Preferences tab.)

```
1 cfform
2  cfformgroup type="tabnavigator" -- Tab navigator container
3   cfformgroup type="page" -- Tabbed page container, child of tabnavigator
4    cfformgroup type="horizontal" -- Aligns its two children horizontally
5     cfinput type="text" -- First name input control
5     cfinput type="spacer" -- Space between the name input controls
5     cfinput type="text" -- Last name input control
4    cfformitem type="hrule" -- Displays a rule
4    cfformitem type="html" -- Displays formatted text
4    cffinput type="text" -- E-mail input control
4    cfformitem type="hrule" -- Displays a rule
4    cfinput type="text" -- Phone number input control
4    cfinput type="spacer" -- Space between the phone and date controls
4    cfinput type="datefield" -- Date input control
3   cfinput type="page" -- Second tabbed page container for preferences
   .
   .
2  cfformgroup type="horizontal" -- Follows the tabnavigator in the form
3  cfinput type="submit" -- Submit button control
3  cfinput type="reset" -- Reset button control
```

Adding text, images, rules, and space with the cfformitem tag

Because Flash forms do not support inline HTML, you use the `cfformitem` tag to add text blocks and horizontal and vertical rules to your form. (Flash form controls, such as `cfinput`, use the `label` or `value` attribute to specify text labels.) You can also use the `cfformitem` tag to add spacers between visual form elements.
The `cfformitem type="hrule"` and `cfformitem type="vrule"` tags put horizontal and vertical rules in the form. You can specify the rule thickness, color, and shadow characteristics by using style specifications. For more information on specifying rule styles, see Styles for cfformitem with hrule or vrule type attributes in ColdFusion Flash Form Style Reference in the *CFML Reference*.
The `cfformitem type="spacer"` tag puts a blank space of the specified height and width on the form. This tag type does not use styles; it can be useful in improving the form appearance by helping you control the form layout.
The `cfformitem type="text"` tag lets you insert plain text in the form You can use the `style` attribute to apply a consistent format to the text.

© 2014 Adobe Systems Incorporated. All rights reserved.

The `cfformitem type="html"` tag lets you insert formatted text and images in the form. You can include basic HTML-style formatting tags in the body of this tag to add images and to format and style the text.
You can use the following formatting tags and attributes in the body of a `cfformitem type="html"` tag:

Tag	Valid attributes
`a`	`href` URL to link to.`target` window name; can be a standard HTML window name such as _blank.
`b`	None.
`br`	None.
`font`	`color` Must be in hexadecimal format, such as #FF00AA. Use a single number sign (#) character.`face` Can be a comma-delimited list of font face names; Flash uses the first font that is available on the client system.`size` In pixels; + and -relative values are allowed.
`i`	None.
`img`	See the attribute table for the `img` tag.Note: Close this tag with /> or an </img> tag.
`li`	None.
`p`	`align` Must be one of the following: `left`, `right`, `center`.
`textformat`	See the attribute table for the `textformat` tag.
`u`	None.

The `img` tag supports the following attributes:

Attribute	Description
`src`	(Required) URL or path to a JPEG or SWF file. Images are not displayed until they have downloaded completely. Flash Player does not support progressive JPEG files.
`width`	Width of the image, in pixels.
`height`	Height of the image in pixels.

© 2014 Adobe Systems Incorporated. All rights reserved.

`align`	Horizontal alignment of the embedded image within the text field. Valid values are `left` and `right`. The default value is `left`.
`hspace`	Number of pixels of horizontal space that surround the image where no text appears. The default value is `8`.
`vspace`	Number of pixels of vertical space that surround the image where no text appears. The default value is `8`.

Note

Because of the Flash dynamic sizing rules, to ensure that the image displays properly, you sometimes have to specify the `formitem` tag `height` attribute and the `width` and `height` attri butes for the form or the containing `cfformgroup` tag. Also, the image always displays on a new line, not inline with text, and text that follows the image in your code occupies any available space to the right of the image.

The `textformat` tag is specific to Flash, and has the following attributes:

Attribute	Description
`blockindent`	Block indentation, in points.
`indent`	Indentation from the left margin to the first character in the paragraph.
`leading`	Amount of leading (vertical space) between lines.
`leftmargin`	Left margin of the paragraph, in points.
`rightmargin`	Right margin of the paragraph, in points.
`tabstops`	Custom tab stops as an array of nonnegative integers. To specify tabs in text, use the \t escape character.

For detailed descriptions of these tags, see the Flash documentation.
The following code creates a simple Flash form that consists of a formatted text area surrounded by horizontal rules:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cfform name="myform" height="220" width="400" format="Flash" >
 <!--- Use text formitem tag with style specifications for the heading. --->
 <cfformitem type="text" style="fontWeight:bold; fontSize:14;">
 Flash form with formatted text and rules
 </cfformitem>
 <!--- The spacer adds space between the text and the rule --->
 <cfformitem type="spacer" height="2" />
 <cfformitem type="hrule" />
 <cfformitem type="html">
  <b><font color="#FF0000" size="+4" face="serif">
  This form has formatted text, including:</font></b><br>
  <textformat blockindent="20" leading="2">
   <li>colored text</li>
   <li><i>italic and bold text</i></li>
   <li>a bulleted list in an indented block</li>
  </textformat>
  <p><b>The text is preceded and followed by horizontal rules</b></p>
  It also has a link to a web page.</b><br>
  <a href="http://www.adobe.com/" target="_blank">
  <font color="#0000FF"><u>
  This link displays the Adobe home page in a new browser window
  </u></font></a>
 </cfformitem>
 <cfformitem type="spacer" height="2"/>
 <cfformitem type="hrule"/>
</cfform>
```

Using the cfformgroup tag to structure forms

ColdFusion provides form group container types that provide basic structure to a Flash form. You specify these types in the `type` attribute of the cfformgroup tag. Use the following container types to control the layout of controls and groups of controls:

Type	Description
`horizontal`	Arranges individual controls horizontally, and optionally applies a label to the left of the controls. Use only for arranging ColdFusion form controls, including `cfformitem` controls. As a general rule, do not use to organize `cfformgroup` containers; use the `hbox` attribute instead.If you put other `cfformgroup` tags inside a `horizontal` form group, the controls inside the included `cfformgroup` tag do not align with other controls in the horizontal group.

© 2014 Adobe Systems Incorporated. All rights reserved.

`vertical`	Arranges individual controls vertically, and optionally applies a label to the left (*not* top) of the controls. Use only for groups of ColdFusion form controls, including `cfformitem` controls. As a general rule, do not use to organize `cfformgroup` containers; use the `vbox` attribute instead.If you put other `cfformgroup` tags inside a `vertical` form group, the controls inside the included `cfformgroup` tag do not align with other controls in the vertical group.
`hbox`	Arranges groups of controls horizontally. Does not apply a label. Use this attribute value to arrange other `cfformgroup` containers. This tag does not enforce alignment of child controls that have labels, so you should not use it to align individual controls.
`vbox`	Arranges groups of controls vertically. Does not apply a label. Use this attribute value to arrange other `cfformgroup` containers. This tag does not enforce alignment of child controls that have labels, so do not use it to align individual controls.
`hdividedbox`	Arranges two or more children horizontally, and places divider handles between the children that users can drag to change the relative sizes of the children. Does not apply a label. The direct children of an `hdividedbox` container must be `cfformgroup` tags with `type` attributes other than `horizontal` or `vertical`.
`vdividedbox`	Arranges two or more children vertically, and places divider handles between the children that users can drag to change the relative sizes of the children. Does not apply a label. The direct children of a `vdividedbox` container must be `cfformgroup` tags with `type` attributes other than `horizontal` or `vertical`.
`tile`	Arranges its children in a rectangular grid in row-first order. Does not apply a label.
`panel`	Consists of a title bar containing the `label` attribute text, a border, and a content area with vertically arranged children.
`accordion`	Places each of its child pages in an accordion pleat with a label bar. Displays the contents of one pleat at a time. Users click the labels to expand or contract the pleat pages. Does not apply a label.

© 2014 Adobe Systems Incorporated. All rights reserved.

`tabnavigator`	Places each of its children on a tabbed page. Users click the tabs to display a selected page. Does not apply a label.
`page`	The immediate child of an accordion or tab navigator container. Specifies the label on the pleat bar or tab, and arranges its child containers and controls vertically.

For more information on using the `accordion`, `tabnavigator`, and `page` `cfformgroup` types, see *Creating complex forms with accordion and tab navigator containers* in Building Flash forms.

Example: structuring with the cfformgroup tag

The following example shows a form with an `hdividedbox` container with two `vbox` containers. The left box uses a `horizontal` container to arrange two radio buttons. The right box uses a `tile` container to lay out its check boxes. You can drag the divider handle to resize the two boxes. When you submit the form, the ColdFusion page dumps the Form scope to show the submitted data.

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cfif Isdefined("Form.fieldnames")>
<cfdump var="#form#" label="form scope">
<br><br>
</cfif>

<cfform name="myform" height="200" width="460" format="Flash" skin="HaloBlue">
 <cfformitem type="html" height="20">
  <b>Tell us your preferences</b>
 </cfformitem>
 <!--- Put the pet selectors to the left of the fruit selectors. --->
 <cfformgroup type="hdividedbox" >
 <!--- Group the pet selector box contents, aligned vertically. --->
  <cfformgroup type="VBox"height="130">
   <cfformitem type="text" height="20">
    Pets:
   </cfformitem>
   <cfformgroup type="vertical" height="80">
    <cfinput type="Radio" name="pets" label="Dogs" value="Dogs"
     checked>
    <cfinput type="Radio" name="pets" label="Cats" value="Cats">
   </cfformgroup>
  </cfformgroup>

 <!--- Group the fruit selector box contents, aligned vertically. --->
  <cfformgroup type="VBox" height="130">
   <cfformitem type="text" height="20">
    Fruits:
   </cfformitem>
   <cfformgroup type="tile" height="80" width="190" label="Tile box">
    <--- Flash requires unique names for all controls --->
    <cfinput type = "Checkbox" name="chk1" Label="Apples"
     value="Apples">
    <cfinput type="Checkbox" name="chk2" Label="Bananas"
     value="Bananas">
    <cfinput type="Checkbox" name="chk3" Label="Pears"
     value="Pears">
    <cfinput type="Checkbox" name="chk4" Label="Oranges"
    value="Oranges">
    <cfinput type="Checkbox" name="chk5" Label="Grapes"
     value="Grapes">
    <cfinput type="Checkbox" name="chk6" Label="Cumquats"
     value="Cumquats">
   </cfformgroup>
  </cfformgroup>
 </cfformgroup>
 <cfformgroup type="horizontal">
  <cfinput type="submit" name="submit" width="100" value="Show Results">
  <cfinput type="reset" name="reset" width="100" value="Reset Fields">
 </cfformgroup>
</cfform>
```

Controlling sizes in Flash forms

Sizing elements in a Flash form is something of an art, rather than a science. As a general rule, if you don't specify the `height` and `width` attributes, Flash tends to do a good job of laying out the form. However, keep in mind the

© 2014 Adobe Systems Incorporated. All rights reserved.

following considerations:

- If you do not specify the `height` and `width` attributes in the cfform tag, Flash reserves the full dimensions of the visible browser window, if the form is not in a table, or the table cell, if the form is in a table, even if they are not required for the form contents. Any HTML output that precedes or follows the form causes the output page to exceed the size of the browser window.
- If you do not specify the height or width of a control, including a form group, Flash adjusts the dimensions, trying to fit the controls in the available space. For example, Flash often extends input boxes to the width of the containing control, if not otherwise specified.In general, it is best to use the following process when you design your Flash form.

Determine the sizes of a Flash form and its controls

1. When you first create the form, don't specify any `height` and `width` attributes on the form or its child tags. Run the form and examine the results to determine height and width values to use.
2. Specify `height` and `width` attributes in the `cfform` tag for the desired dimensions of the form. You can specify absolute pixel values, or percentage values relative to the size of the containing window.
3. Specify any `height` or `width` attributes on individual tags. These values must be in pixels.
4. Repeat step 3 for various tags, and possibly step 2, until your form has a pleasing appearance.

Repeating Flash form elements based on query data

The `repeater` `cfformgroup` type tells Flash Player to iterate over a query and create a set of the `cfformgroup` tag's child controls for each row in the query. For each set of child controls, `bind` attributes in the child tags can access fields in the current query row. This `cfformgroup` type lets you create Flash forms where the number of controls can change based on a query, without requiring ColdFusion to recompile the Flash SWF file for the form. This significantly enhances server performance.

> **Note**
>
> For more information on binding data, see Binding data in Flash forms.

Optionally, you can specify a start row and a maximum number of rows to use in the `repeater`. Unlike most ColdFusion tags, `repeater` index values start at 0, not 1. To specify a repeater that starts on the first line of the query object and uses no more than 15 rows, use a tag such as the following:

```
<cfformgroup type="repeater" query="q1" startrow="0" maxrows="15">
```

One example that can use a repeater is a form that lets a teacher select a specific class and update the student grades. Each class can have a different number of students, so the form must have a varying number of input lines. Another example is a shopping cart application that displays the product name and quantity ordered and lets users change the quantity.
The following example uses the `cfformgroup` tag with a `repeater type` attribute value to populate a form. It creates a query, and uses the repeater to iterate over a query and create a firstname and lastname input box for each row in the query.

© 2014 Adobe Systems Incorporated. All rights reserved.

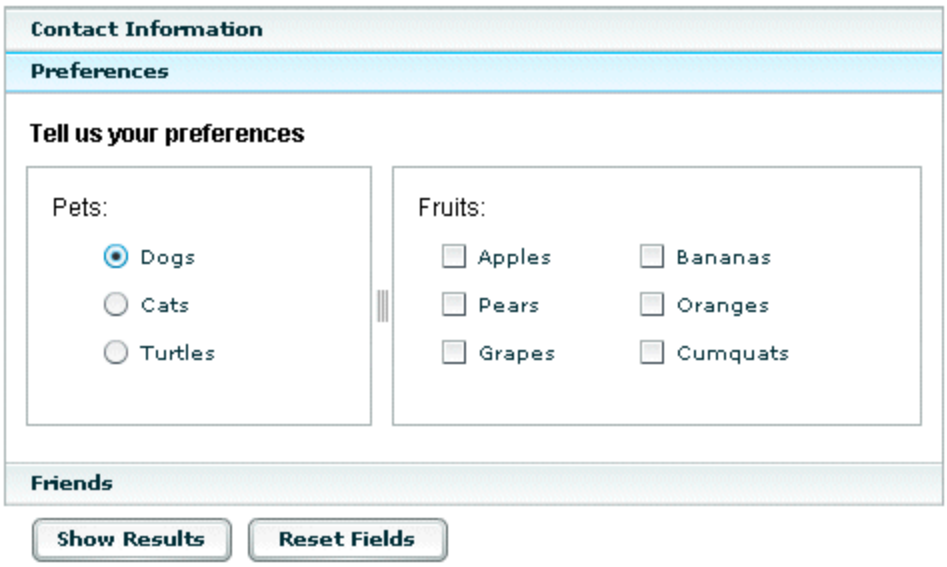
```
<cfif IsDefined("Form.fieldnames")>
 <cfdump var="#form#" label="form scope">
 <br><br>
</cfif>
<cfscript>
 q1 = queryNew("id,firstname,lastname");
 queryAddRow(q1);
 querySetCell(q1, "id", "1");
 querySetCell(q1, "firstname", "Rob");
 querySetCell(q1, "lastname", "Smith");
 queryAddRow(q1);
 querySetCell(q1, "id", "2");
 querySetCell(q1, "firstname", "John");
 querySetCell(q1, "lastname", "Doe");
 queryAddRow(q1);
 querySetCell(q1, "id", "3");
 querySetCell(q1, "firstname", "Jane");
 querySetCell(q1, "lastname", "Doe");
 queryAddRow(q1);
 querySetCell(q1, "id", "4");
 querySetCell(q1, "firstname", "Erik");
 querySetCell(q1, "lastname", "Pramenter");
</cfscript>

<cfform name="form1" format="flash" height="220" width="450">
 <cfselect label="select a teacher" name="sel1" query="q1" value="id"
  display="firstname" width="100" />
 <cfformgroup type="repeater" query="q1">
  <cfformgroup type="horizontal" label="name">
   <cfinput type="Text" name="fname"bind="{q1.currentItem.firstname}">
   <cfinput type="Text" name="lname" bind="{q1.currentItem.lastname}">
  </cfformgroup>
 </cfformgroup>
 <cfinput type="submit" name="submitBtn" value="Send Data" width="100">
</cfform>
```

Creating complex forms with accordion and tab navigator containers

The `accordion` and `tabnavigator` attributes of the `cfformgroup` tag let you construct complex forms that would otherwise require multiple HTML pages. With accordions and tab navigator containers, users can switch among multiple entry areas without submitting intermediate forms. All data that they enter is available until they submit the form, and all form elements load at one time.

An accordion container places each logical form page on an accordion pleat. Each pleat has a label bar; when the user clicks a bar, the current page collapses and the selected page expands to fill the available form space. The following image shows a three-pleat accordion, open to the middle pleat, Preferences:

© 2014 Adobe Systems Incorporated. All rights reserved.



A tab navigator container places each logical form page on a tabbed frame. When the user clicks a tab, the selected page replaces the previous page. The image in About Flash forms shows a tab navigator container.
The following example generates a two-tab navigator container that gets contact information and preferences. You can change it to an accordion container by changing the `type` attribute of the first `cfformgroup` tag from `tabnavigator` to `accordion`. To prevent the accordion from having scroll bars, increase the `cfform` tag `height` attribute to 310 and the `tabnavigator` tag `height` attribute to 260.

```
<cfif IsDefined("Form.fieldnames")>
 <cfdump var="#form#" label="form scope">
 <br><br>
</cfif>
<br>
<cfform name="myform" height="285" width="480" format="Flash" skin="HaloBlue">
 <cfformgroup type="tabnavigator" height="240" style="marginTop: 0">
  <cfformgroup type="page" label="Contact Information">
   <!--- Align the first and last name fields horizontally. --->
   <cfformgroup type="horizontal" label="Your Name">
    <cfinput type="text" required="Yes" name="firstName" label="First"
     value="" width="100"/>
    <cfinput type="text" required="Yes" name="lastName" label="Last"
     value="" width="100"/>
   </cfformgroup>
   <cfformitem type="hrule" />
   <cfformitem type="HTML"><textformat indent="95"><font size="-2">
    Flash fills this field in automatically.
    You can replace the text.
    </font></textformat>
   </cfformitem>
   <!--- The bind attribute gets the field contents from the firstName
     and lastName fields as they get filled in. --->
   <cfinput type="text" name="email" label="email"
    bind="{firstName.text}.{lastName.text}@mm.com">
   <cfformitem type="spacer" height="3" />
   <cfformitem type="hrule" />
   <cfformitem type="spacer" height="3" />

   <cfinput type="text" name="phone" validate="telephone" required="no"
    label="Phone Number">
```

© 2014 Adobe Systems Incorporated. All rights reserved.

```
    <cfinput type="datefield" name="mydate1" label="Requested date">
   </cfformgroup>

   <cfformgroup type="page" label="Preferences" style="marginTop: 0">
    <cfformitem type="html" height="20">
     <b>Tell us your preferences</b>
    </cfformitem>
    <!--- Put the pet selectors to the left of the fruit selectors. --->
    <cfformgroup type="hdividedbox" >
    <!--- Group the pet selector box contents, aligned vertically. --->
     <cfformgroup type="VBox"height="130">
      <cfformitem type="text" height="20">
       Pets:
      </cfformitem>
      <cfformgroup type="vertical" height="80">
       <cfinput type="Radio" name="pets" label="Dogs" value="Dogs"
        checked>
       <cfinput type="Radio" name="pets" label="Cats" value="Cats">
      </cfformgroup>
     </cfformgroup>

    <!--- Group the fruit selector box contents, aligned vertically. --->
     <cfformgroup type="VBox" height="130">
      <cfformitem type="text" height="20">
       Fruits:
      </cfformitem>
      <cfformgroup type="tile" height="80" width="190" label="Tile box">
       <--- Flash requires unique names for all controls. --->
       <cfinput type = "Checkbox" name="chk1" Label="Apples"
        value="Apples">
       <cfinput type="Checkbox" name="chk2" Label="Bananas"
        value="Bananas">
       <cfinput type="Checkbox" name="chk3" Label="Pears"
        value="Pears">
       <cfinput type="Checkbox" name="chk4" Label="Oranges"
        value="Oranges">
       <cfinput type="Checkbox" name="chk5" Label="Grapes"
        value="Grapes">
       <cfinput type="Checkbox" name="chk6" Label="Kumquats"
        value="Cumquats">
      </cfformgroup>
     </cfformgroup>
    </cfformgroup>
   </cfformgroup>
  </cfformgroup>

  <cfformgroup type="horizontal">
   <cfinput type = "submit" name="submit" width="100" value = "Show Results">
   <cfinput type = "reset" name="reset" width="100" value = "Reset Fields">
```

© 2014 Adobe Systems Incorporated. All rights reserved.

```
  </cfformgroup>
 </cfform>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

Setting styles and skins in Flash forms

ColdFusion provides the following methods for controlling the style and appearance of Flash forms and their elements:

- **Skins** provide a simple method for controlling the overall appearance of your form. A single skin controls the entire form.
- **Styles** provide a finer-grained level of control than skins. Each style specifies a particular characteristic for a single control. Many styles are also inherited by a control's children. You can use both techniques in combination: you can specify a skin for your form and use styles to specify the appearance (such as input text font) of individual controls. For detailed information on the style names and values that you can use, see Cold Fusion Flash Form Style Reference in the *CFML Reference*.

Controlling form appearance with Flash skins

The cfform tag takes a `skin` attribute, which lets you select an overall appearance for your form. The skin determines the color used for highlighted and selected elements.
You can select the following Flash skins:

- haloBlue
- haloGreen (the default)
- haloOrange
- haloSilver

About Flash form styles

The ColdFusion Flash form tags have a `style` attribute that lets you specify control characteristics using CSS syntax. You can specify a `style` attribute in the following tags:

- `cfform`
- `cfformgroup`
- `cfcalendar`
- cfformitem, types `hrule` and `vrule`
- `cfgrid`
- `cfinput`
- `cfselect`
- `cftextarea`
- `cftree`}}`The attributes for the {{cfform` and `cfformgroup` generally apply to all the form or form group's children.Flash supports many, but not all, standard CSS styles. ColdFusion Flash forms only support applying specific CSS style specifications to individual CFML tags and their corresponding Flash controls and groups. You cannot use an external style sheet or define a document-level style sheet, as you can for HTML forms.

Flash form style specification syntax

To specify a Flash style, use the following format:

```
style="stylename1: value; stylename2: value; ..."
```

For example, the following code specifies three style values for a text input control:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cfinput type="text" name="text2" label="Last"
 style="borderSyle:inset; fontSize:12; backgroundColor:##FFEEFF">
```

About Flash form style value formats

Style properties can be Boolean values, strings, numbers, or arrays of these values.

Length format

You specify styles that take length or dimension values, including font sizes, in pixels.
The `fontSize` style property lets you use a set of keywords in addition to numbered units. You can use the following keywords when you set the `fontSize` style property. The exact sizes are defined by the client browser.

- `xx-small`
- `x-small`
- `small`
- `medium`
- `large`
- `x-large`
- `xx-large` The following `cfinput` tag uses the `style` attribute with a `fontSize`keyword to specify the size of the text in the input box:

```
<cfinput type="text" name="text1" style="fontSize:X-large" label="Name">
```

Time format

You specify styles that take time values, such as the `openDuration` style that specifies how fast an accordion pleat opens, in milliseconds. The following example shows an `accordion` tag that takes one-half second to change between accordion pleats:

```
<cfformgroup type="accordion" height="260" style="openDuration: 500">
```

Color format

You define color values, such as those for the `backgroundColor` style, in the following formats:

Format	Description
hexadecimal	Hexadecimal colors are represented by a six-digit code preceded by two number sign characters (##). Two # characters are required to prevent ColdFusion from interpreting the character. The range of possible values is `##000000` to `##FFFFFF`.

© 2014 Adobe Systems Incorporated. All rights reserved.

VGA color names	VGA color names are a set of 16 basic colors supported by all browsers that support CSS. The available color names are `Aqua`, `Black`, `Blue`, `Fuchsia`, `Gray`, `Green`, `Lime`, `Maroon`, `Navy`, `Olive`, `Purple`, `Red`, `Silver`, `Teal`, `White`, and `Yellow`. Some browsers support a larger list of color names. VGA color names are not case-sensitive.

Some styles support only the hexadecimal color format.
Some controls accept multiple colors. For example, the tree control's `depthColors` style property can use a different background color for each level in the tree. To assign multiple colors, use a comma-delimited list, as the following example shows:

```
style="depthColors: ##EAEAEA, ##FF22CC, ##FFFFFF"
```

About Flash form style applicability and inheritance

Because of the way Flash control styles are implemented, some common styles are valid, but have no effect, in some tags. Therefore, in the table in Styles valid for all controls in ColdFusion Flash Form Style Reference in the *CFML Reference*, the listed styles do not cause errors when used in controls, but might not have any effect.
Styles can be inheritable or noninheritable. If a style is noninheritable, it only affects the tag, and does not affect any of its children. For example, to maintain a consistent background color in an `hbox` form group and its children tags, specify the color in all tags. If a style is inheritable, it applies to the tag and its children.

Example: applying styles to a Flash form

The following code creates a Flash form that uses a skin and styles to control its appearance.
The code for the form is as follows. Comments in the code explain how formatting controls and styles determine the appearance.

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<!--- Specify the form height and width, use the HaloBlue skin.
   Note: Flash ignores a backgroundColor style set in cfform. --->
<cfform name="myform" height="390" width="440" format="Flash" skin="HaloBlue">
 <!--- The input area is a panel. Styles to specify panel characteristics.
    Child controls inherit the background color and font settings. --->
 <cfformgroup type="Panel" label="Contact Information"
   style="marginTop:20; marginBottom:20; fontSize:14; fontStyle:italic;
   headerColors:##FFFF00, ##999900; backgroundColor:##FFFFEE;
   headerHeight:35; cornerRadius:12">
  <!--- This vbox sets the font size and style, and spacing between and
     around its child controls. --->
  <cfformgroup type="vbox" style="fontSize:12; fontStyle:normal;
    verticalGap:18; marginLeft:10; marginRight:10">
   <!--- Use a horizontal group to align the first and last name fields
      and set a common label. --->
   <cfformgroup type="horizontal" label="Name" >
    <!--- Use text styles to highlight the entered names. --->
    <cfinput type="text" required="Yes" name="firstName" label="First"
     value="" width="120" style="color:##006090; fontSize:12;
     fontWeight:bold" />
    <cfinput type="text" required="Yes" name="lastName" label="Last"
     value="" width="120" style="color:##006090; fontSize:12;
     fontWeight:bold"/>
   </cfformgroup>
   <!--- Horizontal rules surround the e-mail address.
      Styles specify the rule characteristics. --->
   <cfformitem type="hrule" style="color:##999900; shadowColor:##DDDD66;
    strokeWidth:4"/>
   <cfformitem type="HTML" Style="marginTop:0; marginBottom:0">
    <textformat indent="57"> <font size="-1">Flash fills this field in
    automatically. You can replace the text.</font></textformat>
   </cfformitem>
   <cfinput type="text" name="email" label="email"
    bind="{firstName.text}.{lastName.text}@mm.com">
   <cfformitem type="hrule" style="color:##999900; shadowColor:##DDDD66;
    strokeWidth:4"/>
   <cfinput type="text" name="phone" validate="telephone" label="Phone">
   <!--- Styles control the colors of the current, selected, and
    rolled-over dates. --->
   <cfinput type="datefield" name="mydate1" label="Date"
    style="rollOverColor:##DDDDFF; selectionColor:##0000FF;
    todayColor:##AAAAFF">
  </cfformgroup> <!--- vbox --->
 </cfformgroup> <!--- panel --->
 <!--- A style centers the buttons at the bottom of the form. --->
 <cfformgroup type="horizontal"style="horizontalAlign:center">
  <cfinput type = "submit" name="submit" width="100" value = "Show Results">
  <cfinput type = "reset" name="reset" width="100" value = "Reset Fields">
 </cfformgroup>
</cfform>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

Using ActionScript in Flash forms

ActionScript 2.0 is a powerful scripting language that is used in Flash and other related products and is like JavaScript. You can use a subset of ActionScript 2.0 code in your Flash forms.
Information on how to include ActionScript in your Flash forms, and restrictions and additions to ActionScript that apply to ColdFusion Flash forms are described here. However, information on writing ActionScript is not provided. For details on ActionScript and how you can use it, see the Flash ActionScript 2.0 documentation, including the documents available in the Flash and Flex sections of LiveDocs at www.adobe.com/go/learn_cfu_docs_en.

Using ActionScript code in CFML

You can use ActionScript in the following attribute of tags in CFML Flash format forms:

- Form and control events, such as the `onSubmit` attribute of the `cfform` tag, or the `onChange` and `onClick` attributes of the `cfinput` tag. The attribute description on the tag reference pages in the *CFML Reference* lis t the event attributes.
- Bind expressions, which you can use to set field values. For more information on binding data, see Binding data in Flash forms.Your ActionScript code can be inline in the form attribute specification, you can make a call to a custom function that you define, or you can use the ActionScript `include`command in the attribute specification to get the ActionScript from a .as file. The following example shows a simple Fahrenheit to Celsius converter that does the conversion directly on the client, without requiring the user to submit a form to the ColdFusion server.

```
<cfform format="flash" width="200" height="150">
 <cfinput type="text" name="fahrenheit" label="Fahrenheit" width="100"
  value="68">
 <cfinput type="text" name="celsius" label="Celsius" width="100">
 <cfinput type="button" name="convert" value="Convert" width="100"
  onClick="celsius.text = Math.round((farenheit.text-32)/1.8*10)/10">
</cfform>
```

Note

You do not use the text property (for example, *`fieldname`*`.text`) to access hidden fields. To access a hidden field, use the format `formname.fieldname = 'value'`.

Custom ActionScript functions

Custom ActionScript functions are the equivalent of CFML UDFs. You can define your own functions in ColdFusion by using the `cfformitem` tag with a `type` attribute value of `script`, or you can define the functions in an ActionScript (.as) file. Also, ColdFusion includes a small number of predefined custom ActionScript functions that you can use in your Flash form controls.
You can use the following custom functions in the ActionScript for all form controls to reset or submit the form:

- `resetForm()`
- `submitForm()`
  You can use the following custom functions in `cfgrid` tags only to insert and delete rows in the grid:

- `GridData.insertRow(`*gridName*`)`
- `GridData.deleteRow(`*gridName*`)`
  The following example shows how you can use the two GridData functions to add custom buttons that add and delete rows from a Flash form. These buttons are equivalent to the buttons that ColdFusion creates if you

© 2014 Adobe Systems Incorporated. All rights reserved.

specify `insert="yes"` and `delete="yes"` in the `cfgrid` tag, but they allow you to specify you own button text and placement. This example puts the buttons on the side of the grid, instead of below it and uses longer than standard button labels.

```
<cfform format="flash" height="265" width="400">
 <cfformitem type="html">
  You can edit this grid as follows:
  <ul>
  <li>To change an item, click the field and type.</li>
  <li>To add a row, click the Insert Row button and type in the fields
   in the highlighted row.</li>
  <li>To delete a row, click anywhere in the row and click the
   Delete Row button</li>
  </ul>
  <p><b>When you are done, click the submit button.</b></p>
 </cfformitem>
 <!--- The hbox aligns the grid and the button vbox horizontally --->
 <cfformgroup type="hbox" style="verticalAlign:bottom;
   horizontalAlign:center">
  <!--- To make all elements align properly, all of the hbox children must
   be containers, so we must put the cfgrid tag in a vbox tag. --->
  <cfformgroup type="vbox">
   <!-- An editable grid with hard coded data (for simplicity).
    By default, this grid does not have insert or delete buttons. --->
   <cfgrid name="mygrid" height="120" width="250" selectmode="edit">
    <cfgridcolumn name="city">
    <cfgridcolumn name="state">
    <cfgridrow data="Rockville,MD">
    <cfgridrow data="Washington,DC">
    <cfgridrow data="Arlington,VA">
   </cfgrid>
  </cfformgroup>
  <!--- Group the Insert and Delete buttons vertically;
    use a vbox to ensure correct alignment. --->
  <cfformgroup type="vbox" name="buttons"style="verticalAlign:bottom;
    horizontalAlign:center">
   <!--- Use a spacer to position the buttons. --->
   <cfformitem type="spacer" height="18" />
   <!--- Use the insertRow method in the onClick event to add a row. --->
   <cfinput type="button" name="ins" value="Insert a new row" width="125"
    onClick="GridData.insertRow(mygrid);">
   <!--- Use the deleteRow method in the onClick event to delete
    the selected row --->
   <cfinput type="button" name="del" value="Delete selected row"
     width="125" onClick="GridData.deleteRow(mygrid)">
   <cfinput type="submit" name="f1" value="Submit" width="125">
  </cfformgroup>
 </cfformgroup>
</cfform>

<!--- Dump the form if it has been submitted. --->
<cfif IsDefined("form.fieldnames")>
<cfdump var="#form#"><br>
</cfif>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

Creating Skinnable XML Forms

You can create XML skinnable forms, which are forms that generate XForms-compliant XML and are normally formatted using an XSLT (extensible stylesheet language transformations) skin.
You can use XML skinnable forms with the skins that Adobe ColdFusion provides without having any knowledge of either XML or XSLT. For information on using XML with ColdFusion, see Using XML and WDDX.

© 2014 Adobe Systems Incorporated. All rights reserved.

About XML skinnable forms

A ColdFusion form with a `format="XML"` attribute is an *XML skinnable form*. When ColdFusion processes an XML skinnable form, it generates the form as XML. By default, it applies an XML Stylesheet Language Transform (XSLT) skin to the XML and generates a formatted HTML page for display on the user's browser. Optionally, you can specify an XSLT file, or you can process the raw XML in your ColdFusion page.
By using XML skinnable forms, you can control the type and appearance of the forms that ColdFusion generates and displays. ColdFusion provides a set of standard skins, including a default skin that it uses if you do not specify another skin (or tell it not to apply a skin). You can also create your own XSLT skin and have ColdFusion use it to give your forms a specific style or appearance.

ColdFusion forms and XForms

ColdFusion skinnable forms conform to and extend the W3C XForms specification. This specification provides an XML syntax for defining interactive forms using a syntax that is independent of form appearance. ColdFusion forms tags include attributes that provide information that does not correspond directly to the XForms model, such as appearance information, validation rules, and standard HTML attributes. ColdFusion skinnable forms retain this information in XForms extensions so that an XSL transformation can use the values to determine appearance or do other processing.
For more information on XML structure of ColdFusion skinnable forms, see ColdFusion XML format.

The role of the XSLT skin

An XSLT skin and associated cascading style sheet (CSS) determine how an XML skinnable form is processed and displayed, as follows:

- The XSLT skin tells ColdFusion how to process the XML, and typically converts it to HTML for display. The skin specifies the CSS style sheet to use to format the output.
- The CSS style sheet specifies style definitions that determine the appearance of the generated output.
  XSLT skins give you extensive freedom in the generated output. They let you create a custom appearance for your forms, or even different appearances for different purposes. For example, you could use the same form in an intranet and on the Internet, and change only the skin to give a different appearance (or even select different subsets of the form for display). You can also create skins that process your form for devices, such as wireless mobile devices.

How ColdFusion processes XML skinnable forms

When ColdFusion processes a `cfform` tag that specifies XML format and an XSLT skin, it does the following to the form:

1. Converts the CFML form tags into an XForms-compliant XML text format and makes it available in a variable with the same name as the form. ColdFusion ignores inline text or HTML tags in the form, and does not pass them to the XML. (It does process HTML `option` tags that are children of a `cfselect` tag.)
2. Applies an XSLT skin to the XML; for example, to convert the form into HTML. The XSLT file specifies the CSS style sheet.
3. Returns the resulting, styled, form to the client, such as a user's browser.
   If you omit the `cfform` tag `skin` attribute, ColdFusion uses a default skin.
   If you specify `skin="none"`, ColdFusion performs the first step, but omits the remaining steps. Your application must handle the XML version of the form as needed. This technique lets you specify your own XSL engine, or incorporate the form as part of a larger form.

ColdFusion XSL skins

ColdFusion provides the following XSLT skins:

- basic

© 2014 Adobe Systems Incorporated. All rights reserved.

- basiccss
- basiccss_top
- beige
- blue
- default
- lightgray
- red
- silver

  The XSLT skin files are located in the *cf_webroot\CFIDE\scripts\xsl* directory, and the CSS files that they use for style definitions are located in the *cf_webroot\CFIDE\scripts\css* directory.

  The default skin and the basic skin format forms identically. ColdFusion uses the default skin if you do not specify a `skin` attribute in the `cfform` tag. The default and basic skins are simple skins that use tables for arranging the form contents. The basic skin uses `div` and `span` tags to arrange the elements. The skins with names of colors are like the basic skin, but make more use of color.

Example: a simple skinnable form

The following image shows a simple XML skinnable form that uses the default skin to format the output:

This form is used in the examples and description.

© 2014 Adobe Systems Incorporated. All rights reserved.

Building XML skinnable forms

You build ColdFusion XML skinnable forms using standard ColdFusion forms tags, including `cfformgroup` and `cfformitem` tags. These tags create the *elements* of the form, the building blocks of the form.
ColdFusion converts the following tags to XML for processing by the XSLT:

- **Standard ColdFusion form data control tags** The `cfgrid`, `cfinput`, `cfselect`, `cfslider`, `cftextarea`, and `tree` tags specify the controls that the form displays.
- **cfformitem tags** Add individual items to your form, such as text or rules. The valid types depend on the skin.
- **cfformgroup tags** Group, organize, and structure the form contents. The valid types depend on the skin.These tags are designed so you can develop forms in a hierarchical structure of *containers* and *children*. Using this model, the `cfform` tag is the master container, and its contents are *children* containers and controls. Each `cfformgroup` tag defines a container that organizes its child elements.The specific tags and attributes that you use in your form depend on the capabilities of the XSLT skin. You use only the tag and attribute combinations that the skin supports. If you are using a skin provided by a third party, make sure that the supplier provides information on the supported attributes.

Using standard ColdFusion form tags

You use standard ColdFusion form tags, such as `cfinput` or `cftree`, as you normally do in standard CFML forms to generate form input elements. ColdFusion maps most of these tags and their subtags (such as `option` tags in the `cfselect` tag) to equivalent XForms elements. ColdFusion maps applet and Flash format `cfgrid` and `cftree` tags to ColdFusion XML extensions that contain Java applet or Flash objects. It converts XML format `cfgrid` and `cftree` tags to ColdFusion XML extension.
The specific attributes you can use and their meanings can depend on the skins.
Using ColdFusion skins: The skins that are supplied with ColdFusion support the attributes that you can use with HTML forms. You can also use `label` attributes to provide labels for the following tags:

- `cfinput` with `type` attribute values of `text`, `button`, `password`, and `file`
- `cfselect`
- `cfslider`
- `cftextarea`
  Using other skins: If you use any other skin, some attributes are not supported, or the skin supports custom attributes. Get the information about the supported attributes from the XSLT skin developer.

Using cfformitem tags

ColdFusion does not process inline text or standard HTML tags when it generates an XML form; therefore, you use the `cfformitem` tag to add formatted HTML or plain text blocks and any other display elements, such as horizontal and vertical rules, to your form.
ColdFusion converts all `cfformitem` `type` attribute values to all-lowercase. For example, if you specify `type="MyType"` ColdFusion converts the type name to `"mytype"`.
ColdFusion makes no other limitations on the `cfformitem` `type` attributes that you can use in a form, but the XSLT skin must process the attributes to display the items.
Using ColdFusion skins: The skins provided in ColdFusion support the following `cfformitem` types:

- `hrule`
- `text`
- `html`
  The `hrule` type inserts an HTML `hr` tag, and the `text` type displays unformatted plain text.
  The `html` type displays HTML-formatted text. You can include standard HTML text markup tags, such as `strong`, `p`, `ul`, or `li`, and their attributes. For example, the following text from the a simple skinnable form section shows how you could use a `cfformitem` tag to insert descriptive text in a form:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cfformitem type="html">
 <b>We value your input</b>.<br>
 <em>Please tell us a little about yourself and your thoughts.</em>
</cfformitem>
```

Using other skins: If you use any other skin, the supported attributes and attribute values depend on the skin implementation. Get the information about the supported attributes and attribute values from the XSLT skin developer.

Using cfformgroup tags

The `cfformgroup` tag lets you structure forms by organizing its child tags, for example, to align them horizontally or vertically. Some skins use `cfformgroup` tags for more complex formatting, such as tabbed navigator or accordion containers. ColdFusion makes no limitations on the `type` attributes that you can use in a form, but the XSLT must process the resulting XML to affect the display.
Using ColdFusion* skins:* The skins provided in ColdFusion support the following `type` attribute values:

- `horizontal`
- `vertical`
- `fieldset`
  The `horizontal` and `vertical` types arrange their child tags in the specified direction and place a label to the left of the group of children. The following text from the *a simple skinnable form* section shows how you could use a `cfformgroup` tag to apply a *Name* label and align first and last name fields horizontally:

```
<cfformgroup type="horizontal" label="Name">
 <cfinput type="text" name="firstname" label="First" required="yes">
 <cfinput type="text" name="lastname" label="Last" required="yes">
</cfformgroup>
```

The `fieldset` type corresponds to the HTML `fieldset` tag, and groups its children by drawing a box around them and replacing part of the top line with legend text. To specify the legend, use the `label` attribute. To specify the box dimensions, use the `style` attribute with `height` and `width` values.
The following code shows a simple form group with three text controls. The cfformgroup type="vertical" tag ensures that the contents of the form is consistently aligned. The cfformgroup type="horizontal" aligns the firstname and lastname fields horizontally.

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cfform name="comments" format="xml" skin="basiccss" width="400"
  preservedata="Yes" >
 <cfformgroup type="fieldset" label="Contact Information">
  <cfformgroup type="vertical">
   <cfformgroup type="horizontal" label="Name">
    <cfinput type="text" size="20" name="firstname" required="yes">
    <cfinput type="text" size="25" name="lastname" required="yes">
   </cfformgroup>
   <cfinput type="text" name="email" label="E-mail" validation="email">
  </cfformgroup>
  </cfformgroup>
</cfform>
```

Note

Because XML is case-sensitive, but ColdFusion is not, ColdFusion converts `cfformgroup` and `cfformitem` attributes to all-lowercase letters. For example, if you specify `cfformgroup type="Random",` ColdFusion converts the type to `random` in the XML.

Using other skins: If you use any other skin, the supported attributes and attribute values depend on the skin implementation. Get the information about the supported attributes and attribute values from the skin developer.

Example: CFML for a skinnable XML form

The following CFML code creates the form shown in the image in About XML skinnable forms. It shows how you can use CFML to structure your form.

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cfform name="comments" format="xml" skin="basiccss" width="400" preservedata="Yes"
>
 <cfinput type="hidden" name="revision" value="12a">
 <cfformgroup type="fieldset" label="Basic Information">
  <cfformgroup type="vertical">
  <cfformgroup type="horizontal" label="Name">
   <cfinput type="text" size="20" name="firstname" required="yes">
   <cfinput type="text" size="25" name="lastname" required="yes">
  </cfformgroup>
  <cfinput type="text" name="email" label="E-mail" validate="email" maxlength="35">
  <cfselect name="satisfaction" style="width:120px" multiple="false"
label="Satisfaction">
   <option selected>very satisfied</option>
   <option>somewhat satisfied</option>
   <option>somewhat dissatisfied</option>
   <option>very dissatisfied</option>
   <option>no opinion</option>
  </cfselect>
  </cfformgroup>
  </cfformgroup>
 <cfformitem name="html1" type="html">
 <p><b>We value your input</b>.<br>
 <em>Please tell us a little about yourself and your thoughts.</em></p>
 </cfformitem>
 <cftextarea name="thoughts" label="Additional Comments" rows="5" cols="66">We
really want to hear from you!</cftextarea>
 <cfformgroup type="horizontal">
  <cfinput type="submit" name="submit" style="width:80" value="Tell Us">
  <cfinput type="reset" name="reset" style="width:80" value="Clear Fields">
 </cfformgroup>
</cfform>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

ColdFusion XML format

The XML generated from a ColdFusion `cfform` tag and its children are described here. It provides a building block toward creating your own XSL skins.

XML namespace use

The XML that ColdFusion generates for forms uses elements and attributes in several XML *namespaces*. Namespaces are named collections of names that help ensure that XML names are unique. They often correspond to a web standard, a specific document type definition (DTD), or a schema. In XML, the namespace name and a colon (:) precede the name of the tag that is defined in that namespace; for example `xf:model` for the XForms namespace `model` tag.
ColdFusion uses several standard XML namespaces defined by the World Wid Web Consortium (W3C). These namespaces correspond to specifications for standard XML dialects such as XHTML, XForms, and XML Events. ColdFusion XML forms also use a custom namespace for skinnable forms XML extensions. The following table lists the namespaces in the XML that ColdFusion generates.

Prefix	URL	Used for
html	http://www.w3.org/1999/xhtml	Form tag information, including action, height, width, and name. XHTML compliant.
xf	http://www.w3.org/2002/xforms	XForms model (including initial field values) and XForms elements that correspond to cfform tags.
ev	http://www.w3.org/2001/xml-events	System events. Used for the `cfinput type="reset".`
cf		All ColdFusion extensions, including passthrough of attributes that do not correspond to XForms elements or attributes.

XML structure

For each CFML tag, ColdFusion converts attributes and element values to XML in the XForms `xf:model` element, or in individual control elements, such as the XForms `xf:input`, `xf:secret`, or `xf:group` elements.
ColdFusion generates XForms XML in the following format. The numbers on each line indicate the level of nesting of the tags.

© 2014 Adobe Systems Incorporated. All rights reserved.

```
1 form tag
2  XForms model element
3   XForms instance element
4    cf:data element
3   XForms submission element
3   XForms bind element
3   XForms bind element
3  .
3  .
3  .
2  (end of model element)
2  XForms or ColdFusion extension control element
2  XForms or ColdFusion extension control element
   .
   .
   .
1 (end of form)
```

Data model

The XForms data model specifies the data that the form submits. It includes information on each displayed control that can submit data, including initial values and validation information. It does not contain information about `cfformgroup` or `cfformitem` tags. The data model consists of the following elements and their children:

- One `xf:instance` element
- One `xf:submission` element
- One `xf:bind` element for each form control that can submit data

xf:instance element

The XForms `xf:instance` element contains information about the form data controls. Any control that can submit data has a corresponding instance element. If the control has an initial value, the instance element contains that value.

The `xf:instance` element contains a single `cf:data` element that contains an element for each data control: `cfgrid`, most `cfinput` tag types, `cfselect`, `cfslider`, `cftextarea`, and `cftree`. Each element name is the corresponding CFML tag's `name` attribute. For applet and Flash format `cfgrid` and `cftree` tags, the element name is the value of the `cf_param_name` parameter of the tree or grid's Java applet object. Only `cfinput` tags of types `submit`, `image`, `reset, andbutton` do not have instance data, because they cannot submit data.

The body of each element contains the initial control data from the CFML tag's `value` attribute or its equivalent. For example, for a `cfselect` tag, the `xf:instance` element body is a comma-delimited list that contains the `name` attributes of all the `option` tags with a `selected` attribute. For `submit` and `image` buttons, the body contains the `name` attribute value.

The following example shows the `xf:instance` element for the form shown in the image in [About XML skinnable forms](#):

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<xf:instance>
 <cf:data>
   <firstname/>
   <lastname/>
   <email/>
   <revision>Comment Form revision 12a</revision>
   <satisfaction>very satisfied</satisfaction>
   <thoughts>We really want to hear from you!</thoughts>
 </cf:data>
</xf:instance>
```

xf:submission element

The `xf:submission` element specifies the action when the form is submitted, and contains the values of the `cfform` `action` and `method` attributes.:
The following example shows the XML for the form shown in the image in About XML skinnable forms:

```
<xf:submission action="/_MyStuff/phase1/forms/XForms/FrameExamples/Figure1.cfm"
 method="post"/>
```

xf:bind elements

The `xf:bind` elements provide information about the input control behavior, including the control type and any data validation rules. The XML has one bind element for each instance element that can submit data. It does not have bind elements for controls such as `cfformitem` tags, or `cfinput` tags with submit, input, reset, or image types. Each element has the following attributes:

Attribute	Description
`id`	CFML tag `name` attribute value
`nodeset`	XPath expression with the path in the XML to the instance element for the control
`required`	CFML tag `required` attribute value

Each `xf:bind` element has an `xf:extension` element with ColdFusion specific information, including type and validation values. The following table lists the cf namespace elements that are used here.

Element	Description
`cf:attribute name="type"`	Control type. One of the following: CHECKBOX, FILE, IMAGE, PASSWORD, RADIO, SELECT, SUBMIT TEXT, CFSLIDER.The TEXT type is used for `cfinput type="text"` and `cftextinput`.

© 2014 Adobe Systems Incorporated. All rights reserved.

`cf:attribute name="onerror"`	JavaScript function specified by the control's `onError` attribute, if any.
`cfargument name="maxlength"`	Value of the control's `maxlength` attribute, if any.
`cf:validate type="valiadationtype"__`	Data validation information. Has one attribute, `type`, the validation type, and one or more `cf:argument` and `cf:trigger` children. ColdFusion generates a `cf:validate` element for each of the following: • `cfinput` or `cftextarea` `validation` attribute • `cfinput` or `cftextarea` `range` attribute • `cfslider`: the `range` and `message` attributes are specified by a `cf:validate type="range"` element
`cf:argument` (in the body of a `cf:validate` element)	Data validation specification. Has one attribute, `name`, and body text. Each `cf:validate` element can have multiple `cf:argument` children, corresponding to the validation-related CFML tag attribute values, such as maximum length, and maximum and minimum range values. The element body contains the CFML attribute value. Valid `name` values are as follows. Unless specified otherwise, the name is identical to the corresponding CFML tag attribute name. • `max` • `message` • `min` • `pattern`
`cf:trigger` (in the body of a `cf:validate` element)	When to do the validation; specifies a form element `validateAt` attribute value.Has one attribute, `event`, which can be one of the following: • `onBlur` • `onSubmit` • `onServer` If a `validateAt` attribute specifies multiple validation triggers, the XML has one `cf:trigger` element for each entry in the list.

The following example shows the `xf:bind` element of the form shown in the image in About XML skinnable forms:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<xf:bind id="firstname"
  nodeset="//xf:model/xf:instance/cf:data/firstname"
  required="true()">
 <xf:extension>
  <cf:attribute name="type">TEXT</cf:attribute>
  <cf:attribute name="onerror">_CF_onError</cf:attribute>
 </xf:extension>
</xf:bind>
<xf:bind id="lastname"
  nodeset="//xf:model/xf:instance/cf:data/lastname"
  required="true()">
 <xf:extension>
  <cf:attribute name="type">TEXT</cf:attribute>
  <cf:attribute name="onerror">_CF_onError</cf:attribute>
 </xf:extension>
</xf:bind>
<xf:bind id="email"
  nodeset="//xf:model/xf:instance/cf:data/email" required="false()">
 <xf:extension>
  <cf:attribute name="type">TEXT</cf:attribute>
  <cf:attribute name="onerror">_CF_onError</cf:attribute>
 </xf:extension>
</xf:bind>
<xf:bind id="satisfaction"
  nodeset="//xf:model/xf:instance/cf:data/satisfaction"
  required="false()">
 <xf:extension>
  <cf:attribute name="type">SELECT</cf:attribute>
  <cf:attribute name="onerror">_CF_onError</cf:attribute>
 </xf:extension>
</xf:bind>
<xf:bind id="thoughts"
  nodeset="//xf:model/xf:instance/cf:data/thoughts" required="false()">
 <xf:extension>
  <cf:attribute name="type">TEXT</cf:attribute>
  <cf:attribute name="onerror">_CF_onError</cf:attribute>
 </xf:extension>
</xf:bind>
```

Control elements

The XML tags that follow the `xf:bind` element specify the form controls and their layout. The XML includes one element for each form control and `cfformitem` or `cfformgroup` tag.

CFML to XML tag mapping

ColdFusion maps CFML tags to XForms elements and ColdFusion extensions as the following table shows:

CFML tag	XML tag
cfinput type="text"	xf:input
cfinput type="password"	xf:secret

© 2014 Adobe Systems Incorporated. All rights reserved.

cfinput type="hidden"	None: instance data only
cfinput type="file"	xf:upload
cfinput type="radio"	xf:select1
cfinput type="checkbox"	xf:select
cfinput type="button"	xf:trigger
cfinput type="image"	xf:submit
cfinput type="reset"	xf:submit
cfinput type="submit"	xf:submit
cfselect multiple="false"	xf:select1
cfselect multiple="true"	xf:select
cftextarea	xf:textarea
cfslider	xf:range
cfgrid	cf:grid
cftree	cf:tree
cfformitem type="text"	xf:output
cfformitem type="html"	xf:output
cfformitem type="*" (all but text, html)	xf:group appearance="*"
cfformgroup type="*"	xf:group appearance="*"

ColdFusion converts `cfformitem` tags with text and html `type` attributes to XForms `output` elements with the tag body in a <![CDATA[section. It converts all other `cfformitem` tags to XForms `group` elements, and sets each element's `appearance` attribute to the `cfformitem` tag's `type` attribute. The XSLT must process these elements to produce meaningful output. For example, the ColdFusion default skin transform displays the `xf:output` text blocks and processes the `xf:group appearance="hrule"` element, but it ignores all other `xf:group` elements.

General control element structure

Each control element that a standard XForms control element can represent has the following general structure.

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<xf:tagname bind="bindid" id="bindid">
 <xf:label>label</xf:label>
 <xf:extension>
  <cf:attribute name="type">controltype</cf:attribute>
  <cf:attribute name="attribname>attribvalue</cf:attribute>
  <cf:attribute name="attribname>attribvalue</cf:attribute>
  .
  .
  .
 </xf:extension>
</xf:tagname>
```

The following table describes the variable parts of this structure:

Part	Description
`tagname`	The xf or cf namespace element name, as identified in the table in CFML to XML tag mapping.
`bindid`	ID attribute of the model `xf:bind` element for this control. Specified by the control's CFML tag `name` attribute.
`label`	Control label text. Specified by one of the following: • The CFML tag `label` attribute • The `value` attribute of the `radiobutton`, `radio`, `submit`, and `reset` `cfinput` tags • The tag body content of `cfselect` `option` subtags, • Not used for `cfgrid` and `cftree` tags.
`controltype`	Type of control. One of the following: • The `cfinput` `type` attribute • `Select`, `slider`, or `textarea`, for the `cfselect`, `cfslider`, or `cftextarea` tags, respectively. • Not used for `cfgrid` and `cftree` tags.
`attribname`	Name of a CFML tag attribute. There is a `cf:attribute` tag for each attribute specified in the CFML code that does not otherwise have an entry in the XML.
`attribvalue`	Value of a CFML tag attribute.

Tag-specific element structure

The information described here about the tag-specific features of the XML for several types of input tags is not all-inclusive. For the specific structure of any ColdFusion form tag, see the XML generated from the tag by ColdFusion.

© 2014 Adobe Systems Incorporated. All rights reserved.

Selection tags

Tags that are used for selection, `cfselect`, `cfinput type="radio"`, and `cfinput type="checkbox"` are converted to XForms `select` and `select1` elements. These elements include an `xf:choices` element, which in turn has an `xf:item` element for each item a user can choose. Each item normally has an `xf:label` element and an `xf:value` element. Check boxes have a single item; select and radio button controls have more than one.
The following example shows the CFML code for a group of two radio buttons, followed by the generated XML control elements. This example also shows the use of a `cfformgroup` tag to arrange and label the radio button group.

h7. CFML

```
<cfformgroup type="horizontal" label="Accept?">
 <cfinput type = "Radio" name = "YesNo" value = "Yes" checked>
 <cfinput type = "Radio" name = "YesNo" value = "No">
</cfformgroup>
```

h7. XML

```
<xf:group appearance="horizontal">
 <xf:label>Accept?</xf:label>
 <xf:extension/>
 <xf:select1 appearance="full" bind="YesNo" id="YesNo">
  <xf:extension>
   <cf:attribute name="type">radio</cf:attribute>
  </xf:extension>
  <xf:choices>
   <xf:item>
    <xf:label>Yes</xf:label>
    <xf:value>Yes</xf:value>
    <xf:extension>
     <cf:attribute name="checked">checked</cf:attribute>
    </xf:extension>
   </xf:item>
   <xf:item>
    <xf:label>No</xf:label>
    <xf:value>No</xf:value>
    <xf:extension/>
   </xf:item>
  </xf:choices>
 </xf:select1>
</xf:group>
```

cfgrid tags

ColdFusion represents a `cfgrid` tag using the `cf:grid` XML tag. This tag has four attributes: `format`, which can be Flash, Applet, or XML; and the `id`, `name`, and `bind` attributes, which all have the value of the `cfgrid` tag `name` attribute.

© 2014 Adobe Systems Incorporated. All rights reserved.

For applet and Flash format grids, ColdFusion inserts `cfgrid` controls in the XML as HTML embed objects in <![CDATA[sections in the body of a `cf:grid` tag. The controls can be Java applets or in SWF file format.
For XML format grids, ColdFusion converts the CFML to XML in the following format:

```
<cf:grid bind="gridname" name="gridname" format="xml" id="gridname>
 <metadata>
  <cfgridAttribute1>attributeValue</cfgridAttribute1>
  ...
  (There are an entry for attributes with a specified or default value.)
 </metadata>
 <columns>
  <column cfgridcolumnAttribute1="value" ... />
  ...
 </columns>
 <rows>
  <row>
   <column1Name>row1Column1Value</column1Name>
   <column2Name>row1Column2Value</column2Name>
   ...
  </row>
  <row>
   <column1Name>row2Column1Value</column1Name>
   <column2Name>row2Column2Value</column2Name>
  </row>
  ...
 </rows>
</cf:grid>
```

The following example shows a minimal grid with two nodes.

h7. CFML

```
<cfgrid name="mygrid" Format="xml" selectmode="Edit" width="350 ">
 <cfgridcolumn name="CorName" header="Course Name" >
 <cfgridcolumn name="Course_ID" header="ID">
 <cfgridrow data="one0,two0">
 <cfgridrow data="one1,two1">
</cfgrid>
```

h7. XML
Most metadata lines are omitted for brevity:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cf:grid bind="mygrid" format="XML" id="mygrid" name="mygrid">
 <metadata>
  <autowidth>false</autowidth>
  <insert>false</insert>
  <delete>false</delete>
  <sort>false</sort>
  <italic>false</italic>
  <bold>false</bold>
  <appendkey>true</appendkey>
  <highlughthref>true</highlughthref>
  <griddatalines>Left</griddatalines>
  <gridlines>true</gridlines>
  <rowheaders>true</rowheaders>
  <rowheaderalign>Left</rowheaderalign>
  <rowheaderitalic>false</rowheaderitalic>
  <rowheaderbold>false</rowheaderbold>
  <colheaders>true</colheaders>
  <colheaderalign>Left</colheaderalign>
  <colheaderitalic>false</colheaderitalic>
  <colheaderbold>false</colheaderbold>
  <selectmode>Edit</selectmode>
  <notsupported>&lt;b&gt; Browser must support Java to view ColdFusion Java
   Applets&lt;/b&gt;</notsupported>
  <picturebar>false</picturebar>
  <insertbutton>insert</insertbutton>
  <deletebutton>delete</deletebutton>
  <sortAscendingButton>SortAsc</sortAscendingButton>
  <sortDescendingButton>SortDesc</sortDescendingButton>
 </metadata>
 <columns>
  <column bold="false" display="true" header="Course Name"
   headerBold="false" headerItalic="false" italic="false"
   name="CorName" select="true"/>
  <column bold="false" display="true" header="ID"
   headerBold="false" headerItalic="false" italic="false"
   name="Course_ID" select="true"/>
 </columns>
 <rows>
  <row>
   <CorName>one0</CorName>
   <Course_ID>two0</Course_ID>
  </row>
  <row>
   <CorName>one1</CorName>
   <Course_ID>two1</Course_ID>
  </row>
 </rows>
</cf:grid>
```

The cftree tags

For applet and Flash format trees, ColdFusion inserts `cftree` controls in the XML as HTML embed objects in <![CDATA[sections in the tag body. The controls can be Java applets or in Flash SWF file format. The `cf:tree` XM L tag has two attributes: `format`, which can be Flash or Applet, and `id`.

© 2014 Adobe Systems Incorporated. All rights reserved.

For XML format trees, ColdFusion converts the CFML to XML in the following format:

```
cf:tree format="XML" id="treename>
 <metadata>
  <cftreeAttribute1>attributeValue</cftreeAttribute1>
  ...
 </metadata>
 <node cfml tree item attributes>
  <node //nested node with no children
   cfml tree item attributes />
  ...
 </node>
 ...
</cf:tree>
```

The following example shows a minimal tree with two nodes:

h7. CFML

```
<cfform name="form2" Format="XML" >
<cftree name="tree1" hscroll="No" vscroll="No" format="xml"
 border="No">
 <cftreeitem value="Divisions">
 <cftreeitem value="Development"
  parent="Divisions" img="folder">
</cftree>
</cfform>
```

h7. XML
The following code shows only the XML that is related to the tree appearance:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cf:tree format="xml" id="tree1">
 <metadata>
  <fontWeight/>
  <align/>
  <lookAndFeel>windows</lookAndFeel>
  <delimiter>\</delimiter>
  <completePath>false</completePath>
  <border>false</border>
  <hScroll>false</hScroll>
  <vScroll>false</vScroll>
  <appendKey>true</appendKey>
  <highlightHref>true</highlightHref>
  <italic>false</italic>
  <bold>false</bold>
 </metadata>
 <node display="Divisions" expand="true" href="" img=""
  imgOpen="" parent="" path="Divisions" queryAsRoot="true"
     value="Divisions">
  <node display="Development" expand="true" href=""
   img="folder" imgOpen="" parent="Divisions"
   path="Divisions\Development" queryAsRoot="true"
      value="Development"/>
 </node>
</cf:tree>
```

The cfformgroup and cfformitem tags

All `cfformgroup` tags and all `cfformitem` tags, except `type="html"` and `type="text",` generate `xf:group` elements. The following rules determine the element structure:

- The CFML tag `type` attribute determines the `xf:group` `appearance` attribute.
- ColdFusion converts `type` attribute values to all-lowercase characters.
- For `cfformgroup` tags only, the CFML `label` attribute determines the `xf:group` `label` attribute.
- All other CFML attributes are placed in `cf:attribute` elements in a `xf:extension` element.
- The `cfformitem` tags generate an `xf:output` element with the body text in a <![CDATA[section.
  The following example shows two `cformitem` tags, and the resulting XML:

h7. CFML

```
<cfformitem name="text1" type="text" style="color:green">
 Please tell us a little about yourself and your thoughts.
</cfformitem>
<cfformitem type="hrule" height="3" width="200" testattribute="testvalue" />
```

h7. XML

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<xf:output><![CDATA[Please tell us a little about yourself and your
thoughts.]]>
 <xf:extension>
  <cf:attribute name="style">color:green</cf:attribute>
 </xf:extension>
</xf:output>
<xf:group appearance="hrule">
 <xf:extension>
  <cf:attribute name="width">200</cf:attribute>
  <cf:attribute name="height">3</cf:attribute>
  <cf:attribute name="testattribute">testvalue</cf:attribute>
 </xf:extension>
</xf:group>
```

Example: control element XML

The following code shows the XML for the input controls for the form shown in the image in About XML skinnable forms. This code immediately follows the end of the `xf:model` element.

```
<xf:group appearance="horizontal">
 <xf:label>name</xf:label>
 <xf:extension/>
 <xf:input bind="firstname" id="firstname">
  <xf:label>First</xf:label>
  <xf:extension>
   <cf:attribute name="type">text</cf:attribute>
   <cf:attribute name="size">20</cf:attribute>
  </xf:extension>
 </xf:input>
 <xf:input bind="lastname" id="lastname">
  <xf:label>Last</xf:label>
  <xf:extension>
   <cf:attribute name="type">text</cf:attribute>
   <cf:attribute name="size">25</cf:attribute>
  </xf:extension>
 </xf:input>
</xf:group>
<xf:input bind="email" id="email">
 <xf:label>Email</xf:label>
 <xf:extension>
  <cf:attribute name="type">text</cf:attribute>
  <cf:attribute name="validation">email</cf:attribute>
 </xf:extension>
</xf:input>
<xf:output><![CDATA[<b>We value your input</b>.<br>
 <em>Please tell us a little about yourself and your thoughts.</em>]]>
 <xf:extension/>
</xf:output>
<xf:group appearance="vertical">
 <xf:extension/>
 <xf:select1 appearance="minimal" bind="satisfaction" id="satisfaction">
  <xf:label>Satisfaction</xf:label>
  <xf:extension>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

```
    <cf:attribute name="type">select</cf:attribute>
    <cf:attribute name="style">width:200</cf:attribute>
   </xf:extension>
   <xf:choices>
    <xf:item>
     <xf:label>very satisfied</xf:label>
     <xf:value>very satisfied</xf:value>
    </xf:item>
    <xf:item>
     <xf:label>somewhat satisfied</xf:label>
     <xf:value>somewhat satisfied</xf:value>
    </xf:item>
    <xf:item>
     <xf:label>somewhat dissatisfied</xf:label>
     <xf:value>somewhat dissatisfied</xf:value>
    </xf:item>
    <xf:item>
     <xf:label>very dissatisfied</xf:label>
     <xf:value>very dissatisfied</xf:value>
    </xf:item>
    <xf:item>
     <xf:label>no opinion</xf:label>
     <xf:value>no opinion</xf:value>
    </xf:item>
   </xf:choices>
  </xf:select1>
  <xf:textarea bind="thoughts" id="thoughts">
   <xf:label>Additional Comments</xf:label>
   <xf:extension>
    <cf:attribute name="type">textarea</cf:attribute>
    <cf:attribute name="rows">5</cf:attribute>
    <cf:attribute name="cols">40</cf:attribute>
   </xf:extension>
  </xf:textarea>
 </xf:group>
 <xf:group appearance="horizontal">
  <xf:extension/>
  <xf:submit id="submit" submission="comments">
   <xf:label>Tell Us</xf:label>
   <xf:extension>
    <cf:attribute name="type">submit</cf:attribute>
    <cf:attribute name="name">submit</cf:attribute>
   </xf:extension>
  </xf:submit>
  <xf:submit id="reset">
   <xf:label>Clear Fields</xf:label>
   <reset ev:event="DOMActivate"/>
   <xf:extension>
    <cf:attribute name="name">reset</cf:attribute>
   </xf:extension>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

```
  </xf:submit>
 </xf:group>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

Creating XSLT skins

You can create your own XSLT skins to process the XML that ColdFusion generates. You must be familiar with XSLT and CSS programming. General information on writing XSLT transformations or CSS styles is not provided here. However, information about the following areas is provided:

- How ColdFusion passes form attribute values to the XML file
- How to extend XSLT skins that ColdFusion provides as templates
- Basic techniques for extending the basic.xsl file to support additional `cfformgroup` and `cfformitem` tag `type` attributes
- How to extend the ColdFusion CSS files to enhance form appearance.

XSLT skin file locations

If you specify an XSLT skin by name and omit the .xsl extension, ColdFusion looks for the file in the `cfform` script source directory and its subdirectories. You can specify the script source directory in your `cfform` tag `scriptsrc` attribute, and you can set a default location on the Settings page in the ColdFusion Administrator. When you install ColdFusion, the default location is set to /CFIDE/scripts/ (relative to the web root).
You can also use a relative or absolute file path, or a URL, to specify the XSLT skin location. ColdFusion uses the directory of the CFML page as the root of relative paths. The following formats are valid:

Format	Location
<cfform format="xml" skin="basic">	Searches for XML/CSS in the default directory and its subdirectories.
<cfform format="xml" skin="c:\foo\bar\basic.xsl">	Uses the absolute path.
<cfform format="xml" skin="basic.xsl">	Searches in the current directory.
<cfform format="xml" skin="..\basic.xsl:">	Searches the parent of the current directory.
<cfform format="xml" skin="http://anywhereOnTheWeb/basic.xsl">	Uses the specified URL.

Note

Hosting companies sometimes move the default skin location folder out of CFIDE. Doing this lets them secure the CFIDE while giving site developers access to the files that you need for `cfform`.

Attribute and value passthrough

ColdFusion passes form tag attributes or attribute values that it does not specifically process directly to the XML, as follows:

- It converts `cfformitem` and `cfformgroup` `type` attributes to `xf:group` element `appearance` attributes.
- It passes the name and value of tag attributes that it does not recognize or process in `cf:attribute` elements.
  This passthrough feature lets you create custom versions of any of the following items for your XSLT to process:

© 2014 Adobe Systems Incorporated. All rights reserved.

- The `cfformitem` types, such as rules, spacers, or other display elements
- The `cfgroup` types, such as divided boxes or tabbed dialog boxes
- The custom `cfinput` types, such as a custom year chooser element
- ColdFusion tag attributes, such as those used to control validation

Extending ColdFusion XSLT skins

ColdFusion provides basic XSLT transforms that you can use as templates and extend for making your own skin. Each skin has a base XSL file, which include several utility XSL files. Utility filenames start with an underscore (*), and multiple base skins share the files. The following table describes the XSL files, which are located in the _cf_webroot*
x

File	Description
default.xsl	The default transform that ColdFusion uses if you do not specify a `skin` attribute for an XML format form. Identical to the basic.xsl file.
basic.xsl	A basic form format that arranges form elements using a table.
basiccss.xsl	A basic form format that arranges form elements using HTML `div` and `span` tags.
colorname.xsl	A basic form format that arranges form elements using a table and applies a color scheme determined by the *colorname* to the form. Based on the basic.xsl file.
_cfformvalidation.xsl	Applies ColdFusion validation rules. Used by all skins.
_formelements.xsl	Transformation rules for form elements except for those defined using `cfformgroup` tags. Used by all skins
grouptype.xsl_group_*typetable.xsl_grouptype*_css.xsl	Transformation rules for `cfformgroup` tags. The tag *type* attribute is part of the filename. Files with *table* in the name are used by basic.xsl and its derivatives. Files with css in the name are used by basiccss.xsl.

All skins support the same set of CFML tags and tag types, and do a relatively simple transformation from XML to HTML. For example, they do not support horizontal or vertical rules.
The ColdFusion skin XSL files have several features that you can use when designing and developing your own transformation. They do the following:

- Provide an overall structure and initial templates for implementing custom transformations.
- Show how you can handle the various elements in the ColdFusion-generated XML.
- Use a structure of included files that can form a template for your XSLT code.
- The base XSL files include a separate file, _cfformvalidation.xsl, with complete code for generating the hidden fields required for ColdFusion onServer validation and the JavaScript for performing ColdFusion onSubmit and onBlur validation. You can include this file without modification to do ColdFusion validation in your XSLT template, or you can change it to add other forms of validation or to change the validation rules.

© 2014 Adobe Systems Incorporated. All rights reserved.

- The base XSL files include files, that implement several form groups, laying out the child tags and applying a label to the group. These files can serve as templates for implementing additional form group types or you can expand them to provide more sophisticated horizontal and vertical form groups.
- You can add custom `cfformgroup` and `cfformitem type` attributes by including additional XSL files.

Extending basic.xsl cfformgroup and cfformitem support

The following procedure describes the steps for extending the basic.xsl file to support additional `cfformgroup` and `cfformitem` types. You can use similar procedures to extend other xsl files.

Add support for cfformgroup and cfformitem types to the basic.xsl

1. Create an XSL file.
2. For each `type` attribute that you want to support, create an `xsl:template` element to do the formatting. The `match`attribute of the element must have the following format:

   ```
   match="xf:group[@appearance='type_attribute_name']"
   ```

   For example, to add a panel `cfformgrouptype`, add an element with a start tag such as the following:

   ```
   <xsl:template match="xf:group[@appearance='panel']">
   ```

3. Deploy your XSL file or files to the *cf_webroot*
   x
4. Add an include statement to the basic.xsl file at the end of the Supported groups section; for example, if you create a my_group_panel.xsl file to handle a panel `cfformgrouptype`, your basic.xsl file would include the following lines:

   ```
   <!-- include groups that will be supported for this skin-->
   <xsl:include href="_group_vertical_table.xsl" />
   <xsl:include href="_group_horizontal_table.xsl" />
   <xsl:include href="_group_fieldset.xsl"/>
   <xsl:include href="my_group_panel.xsl" />
   ```

Styling forms by extending the ColdFusion CSS files

Each ColdFusion skinnable form XSL file uses a corresponding CSS style sheet to specify the form style and layout characteristics. The following CSS files are located in the *cf_webroot*\CFIDE\scripts\css directory:

File	Description
basic_style.cssdefault_style.css	Provides a plain style for ColdFusion XSL files that use table-based formatting. These files are identical and are used by the basic.xsl and default.xsl transforms. ColdFusion uses the default_style.css if you do not specify a skin in your `cfform` tag.

© 2014 Adobe Systems Incorporated. All rights reserved.

basic2_style.css	The basic_style with limited positioning changes for use with XSL files that have div-based formatting. Used by the basiccss.xsl transform.
css_layout.css	Style specifications for laying out forms that use div-based formatting. Used by the basiccss.xsl transform.
*colorname*_style.css	Used by the color-formatted ColdFusion skins. Defines the same classes as basic_style.css, with additional property specifications.

The ColdFusion XSL files and their corresponding CSS style sheets use classes extensively to format the form. The basic.xsl file, for example, has only one element style; all other styles are class-based. Although the CSS files contain specifications for all classes used in the XSL files, they do not always contain formatting information. The horizontal class definition in basic_style.css, which is used for horizontal form groups, for example, is empty.
You can enhance the style of XML skinnable forms without changing the XSL transform by enhancing the style sheets that ColdFusion provides.

#back to top

© 2014 Adobe Systems Incorporated. All rights reserved.

Using Ajax User Interface Components and Features

Use Adobe ColdFusion Ajax-based layout and form controls and other Ajax-based user interface capabilities to create a dynamic application.

For information about how ColdFusion uses the Ajax framework in general, or how to use ColdFusion Ajax data and programming capabilities, including binding to form data and managing JavaScript resources, see Using Ajax Data and Development Features.

© 2014 Adobe Systems Incorporated. All rights reserved.

About Ajax and ColdFusion user interface features

Ajax (Asynchronous JavaScript and XML) is a set of web technologies for creating interactive web applications. Ajax applications typically combine:

- HTML and CSS for formatting and displaying information.
- JavaScript for client-side dynamic scripting
- Asynchronous communication with a server using the XMLHttpRequest function.
- XML or JSON (JavaScript Object Notation) as a technique for serializing and transferring data between the sever and the client.
  ColdFusion provides many tools that simplify using Ajax technologies for dynamic applications. By using ColdFusion tags and functions, you can easily create complex Ajax applications.

ColdFusion Ajax features

ColdFusion provides two types of Ajax features:

- Data and development features
- User interface features

Data and development features

ColdFusion data and development features help you develop effective Ajax applications that use ColdFusion to provide dynamic data. They include many features that you can use with other Ajax frameworks, including Spry. The following data and development features are important for use with form and layout tags:

- ColdFusion supports data binding in many tags. Binding allows form and display tags to dynamically display information based on form input. In the simplest application, you display form data directly in other form fields. But usually, you pass form field data as parameters to CFC or JavaScript functions or CFM pages, and use the results to control the display.
- The `cfajaximport` tag specifies the location of the JavaScript and CSS files that a ColdFusion page imports or to selectively import files required by specific tags. The ability to change the file location lets you support a wide range of configurations and use advanced techniques, such as application-specific styles.
  For more information about the data and development features and how to use them, see Using Ajax Data and Development Features.

User Interface tags and features

Several ColdFusion user interface elements incorporate Ajax features. The tags and tag-attribute combinations can be divided into the following categories:

- Container tags that lay out or display contents
- File management tags that handle files
- Form tags that dynamically display data
- A menu tag that lets you create menu bars and pull-down menus
- User assistance features that provide tool tips and form completion
- Four other tags for using geographical maps, progress bar, media player, and message box.
  The following table lists the basic tags and attributes that display the Ajax-based features. For information on additional forms-specific features, see Using Ajax form controls and features.

Tag/attribute	Description
Container tags	

© 2014 Adobe Systems Incorporated. All rights reserved.

`cfdiv`	An HTML `div` region that can be dynamically populated by a bind expression. Forms in this region submit asynchronously.
`cflayout`	A horizontal or vertical box, a tabbed region, or a set of bordered regions that can include a top, bottom, left, right, and center regions.
`cflayoutarea`	An individual region within a `cflayout` area, such as the display that appears in a tabbed layout when the user select a tab. Forms in this region submit asynchronously.
`cfpod`	An area of the browser window with an optional title bar and a body that contains display elements. Forms in this region submit asynchronously.
`cfwindow`	A pop-up window within the browser. You can also use the `ColdFusion.Window.createWindow` function to create a pop-up window. Forms in this region submit asynchronously.
File management tags	
cffileupload	A dialog for uploading multiple files from the user's system.
Forms tags	
`cfgrid format="html"`	A dynamic, editable, sortable, data grid.
`cfinput type="datefield"`	An input control that users can fill by selecting a date from a pop-up calendar.
`cftextarea richtext="yes"`	A text area with a set of controls that let users format the displayed text.
`cftree format="html"`	A dynamic tree-format representation of data.
`cfslider`	A slider control, for selecting a numeric value from a range, in a ColdFusion form.
Menu tags	
`cfmenu`	A menu bar or the root of a drop-down menu.
`cfmenuitem`	An individual item in a menu, or the root of a submenu.

© 2014 Adobe Systems Incorporated. All rights reserved.

User assistance tags and attributes	
`cfinput type="text" autosuggest="bind}}{{expression"`	A drop-down autofill suggestion box. As the user types, a list appears with completion suggestions based on the text the user has typed.
`cftooltip` tag, and the `tooltip` attribute on `cfinput`, `cfselect`, `cftextarea` controls	A textual description of a control or region that appears when the user hovers the mouse over the control or region.
Other tags	
cfprogressbar	A progress bar to indicate the progress of an activity such as a file download.
cfmap	A geographical map within a ColdFusion web page.
cfmediaplayer	An inbuild media player.
cfmessagebox	A control for displaying pop-up messages.

In addition to the tags and attributes, ColdFusion provides many JavaScript functions that let you control and manage the display. Many functions control the display of specific tags. For example, you can use JavaScript functions to dynamically display and hide the window. There are also several utility tags, such as the `ColdFusion.getElementValue` function that gets the value of a control attribute, or the `ColdFusion.navigate` function that displays the results of a URL in a container tag. For a complete list of all ColdFusion Ajax JavaScript functions, and detailed function descriptions, see Ajax JavaScript Functions in the *CFML Reference*.

Using ColdFusion Ajax user interface features

ColdFusion Ajax user interface features let you create data-driven pages that update dynamically without requiring multiple HTML pages or page refreshes or non-HTML display tools such as Flash forms. Many user interface features use data binding to dynamically get data based on other data values: form field values, form control selections, and selections in Spry data sets.
ColdFusion Ajax user interface controls and features can be divided into two major categories:

- Display layout
- Data interaction
  Display layout controls include the `cflayout`, `cfpod`, and `cfwindow` controls. Some of the data interaction features include the HTML `cfgrid` control, the `cfmenu` control, and dynamic autosuggest lists for text input controls. Most display layout and data interaction features can use data binding to dynamically interact with the user.
  ColdFusion Ajax user interface features are based on the [Yahoo
  User Interface Library|http://developer.yahoo.com/yui/] and the Ext JavaScript Library. Also, the `cftextarea` rich text editor is based on the FCKeditor text editor. In most situations, you require only ColdFusion tags and functions (including JavaScript functions) to create and manage the interface. However, advanced developers can modify the library code, particularly the CSS styles, to customize the controls in more complex ways.

© 2014 Adobe Systems Incorporated. All rights reserved.

Controlling Ajax user interface layout

The following layout tags let you dynamically control the display:

- cfdiv
- cflayout
- cfpod
- cfwindow
  For information about how you can use these tags to submit form contents asynchronously, see *Using Ajax containers for form submission* in Using Ajax form controls and features.

Using the cfdiv tag

The cfdiv tag is a general-purpose container that lets you use a bind expression to specify its contents. It therefore lets you dynamically refresh any arbitrary region on the page based on bind events. By default, the tag creates an HTML `div` region, but it can create any HTML tag with body contents. Unlike other ColdFusion Ajax container tags, you can use any type of bind expression to populate contents: CFC or JavaScript function, URL, or a string with bind parameters. As a result, the `cfdiv` tag provides substantial flexibility in dynamically populating the page contents.
The `cfdiv` tag is also useful if you want a form to submit asynchronously. That is, whether or not you use a bind expression to populate the tag. If you submit a form that is inside a `cfdiv` tag (including in HTML returned by a bind expression), the form submits asynchronously. The response from the form submission populates the `cfdiv` region. (The `cflayoutarea`, `cfwindow`, and `cfpod` tags have the same behavior.) For example, you could have a page with a form that includes a list of artists, and lets you add artists. If the form is in a `cfdiv` tag, when the user submits the form, the entire page is not refreshed, only the region inside the `cfdiv` tag. For an example of using container controls for asynchronous forms, see *Using Ajax containers for form submission* in Using Ajax form controls and features.
One use case for a `cfdiv` tag is an application where a `cfgrid` tag displays an employee list. Details of the selected row in the grid are displayed inside a `cfdiv` tag with a bind expression that specifies the `cfgrid` in a bind parameter. As users click through different employees on the grid, they get the employee details in the `cfdiv` regio n.
The following simple example shows how you can use the `cfdiv` tag to get data using a bind expression. It uses binding to display the contents of a text input field in an HTML div region. When a user enters text in the input box and tabs out of it, or clicks another region of the application, the div region displays the entered text.
The cfdiv tag.cfm file, the main application file, has the following contents.

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>cfdiv Example</title>
</head>

<body>
<cfform>
 <cfinput name="tinput1" type="text">
</cfform>

<h3> using a div</h3>
<cfdiv bind="url:divsource.cfm?InputText={tinput1}" ID="theDiv"
  style="background-color:##CCffFF; color:red; height:350"/>
</body>
</html>
```

The divsource.cfm file that defines the contents of the div region has the following code:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<h3>Echoing main page input:</h3>
<cfoutput>
 <cfif isdefined("url.InputText")>
  #url.InputText#
 <cfelse>
  No input
 </cfif>
</cfoutput>
```

Using layouts

The cflayout tag controls the appearance and arrangement of one or more child `cflayoutarea` regions. The cflayoutarea regions contain display elements and can be arranged in one of the following ways:

- Horizontally or vertically.
- In a free-form bordered grid (panel layout) with up to five regions: top, bottom, left. right, and center. You can optionally configure the layout so that users can resize or collapse any or all of the regions, except the center region. The center region grows or shrinks to take up any space that other regions do not use. You can also dynamically show or hide individual regions, or let users collapse, expand, or close regions.
- As a tabbed display, where selecting a tab changes the display region to show the contents of the tab's layout area. You can dynamically show and hide, and enable and disable tabs, and optionally let users close tabs.
  You can configure a layout area to have scroll bars all the time, only when the area content exceeds the available screen size, or never. You can let layout area contents extend beyond the layout area. You can also nest layouts inside layout areas to create complex displays.
  You can define the layout area content in the `cflayoutarea` tag body. But, you can also use a bind expression to dynamically get the content by calling a CFC function, requesting a CFML page, or calling a JavaScript function.
  ColdFusion provides many JavaScript functions for managing layouts, including functions to collapse, expand, show, and hide border areas; and to create, enable, disable, select, show, and hide tabs. For a complete list of functions, see Ajax JavaScript Functions in the *CFML Reference*.
  The following example shows the use of a tabbed layout, including the use of JavaScript functions to enable and disable a tab, and to show and hide a tab.

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
</head>

<body>
<!--- The tabheight attribute sets the height of all tab content areas and therefore
the
   layout height. The width style controls the layout width. --->
<cflayout type="tab" name="mainTab" tabheight="300px" style="width:400px">

 <!--- Each layoutarea is one tab. --->
 <cflayoutarea title="First Tab" name="tab1">
 <h2>The First Tab</h2>
 <p>
 Here are the contents of the first tab.
 </p>
 </cflayoutarea>

 <cflayoutarea title="Second Tab" name="tab2">
 <h2>The Second Tab</h2>
 <p>
 This is the content of the second tab.
 </p>
 </cflayoutarea>
</cflayout>

<p>
Use these links to test selecting tabs via JavaScript:<br />
<a href="" onClick="ColdFusion.Layout.selectTab('mainTab','tab1');return false;">
 Click here to select tab 1.</a><br />
<a href="" onClick="ColdFusion.Layout.selectTab('mainTab','tab2');return false;">
 Click here to select tab 2.</a><br />
</p>

<p>
Use these links to test disabling/enabling via JavaScript. Notice that you cannot
disable
 the currently selected tab.<br />
<a href="" onClick="ColdFusion.Layout.enableTab('mainTab','tab1');return false;">
 Click here to enable tab 1.</a><br />
<a href="" onClick="ColdFusion.Layout.disableTab('mainTab','tab1');return false;">
 Click here to disable tab 1.</a><br />
</p>

</body>
</html>
```

For an example that uses a bordered layout with `cfpod` children, see the next section. For another example of a tab layout, see the cflayoutarea tag in the *CFML Reference*. For an example of a bordered layout nested inside a layout area of a vertical layout, see cflayout in the *CFML Reference*.

Styling layouts

The `cflayout` and `cflayoutarea` tags have `style` attributes. The `cflayout` tag `style` attribute controls the style of the layout container, and sets default values for many, but not all, styles for the layout areas. For example,

© 2014 Adobe Systems Incorporated. All rights reserved.

the color and background color styles of the `cflayout` tag set the default text and background colors in the layout areas. But the `cflayout` tag `border` style sets only the color of the border around the entire layout, not the layout area borders. The `cflayoutarea` tag `style` attribute controls the style of the individual layout area and overrides any corresponding settings in the `cflayout` tag.

As is often the case with complex controls, the effects of layout and layout area styles can vary. For example, do not often specify the `height` style in the `cflayout` tag; instead, specify height styles on each of the `cflayoutarea` ta gs.

The following simple example shows a tab layout with two layout areas. The layout has a light pink background color, and the layout areas have three pixel-wide red borders.:

```
<cflayout name="layout1" type="tab" style="background-color:##FFCCCC">
 <cflayoutarea title="area1" style="border:3px solid red">
  Layout area 1
 </cflayoutarea>
 <cflayoutarea title="area1" style="border:3px solid red">
  Layout area 2
 </cflayoutarea>
</cflayout>
```

Using pods

The cfpod control creates a content region with a title bar and surrounding border. You can define the pod content in the `cfpod` tag body, or you can use a bind expression to dynamically get the content from a URL. Pods are frequently used for portlets in a web portal interface and for similar displays that are divided into independent, possibly interactive, regions.

You control the pod header style and body style independently by specifying CSS style properties in the `headerStyle` and `bodyStyle` attributes.

The following example uses multiple pods inside `cflayoutarea` tags to create a simple portal. The time pod gets the current time from a CFML page. The contents of the other pods is defined in the `cfpod` bodies for simplicity. Each pod uses the `headerStyle` and `bodyStyle` attributes to control the appearance.

The cfpodExample.cfm application has the following code:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<html>
<head>
</head>
<body>
<cflayout name="theLayout" type="border" style="height:300;">
 <cflayoutarea position="left" size="300" style="float:right;">
  <cfpod width="300" name="theNews" title="All the latest news"
    headerstyle="background-color:##DDAADD; font-size:large;
     font-style:italic; color:black"
    bodyStyle="background-color:##FFCCFF; font-family:sans-serif;
     font-size:80%">
   Contents of a news feed would go here.
  </cfpod>
 </cflayoutarea>
 <cflayoutarea position="center" align="center" >
  <cfpod name="theSports" width="500"
    title="What's new in your favorite sports"
    headerstyle="background-color:##AADDDD; font-size:large;
     font-style:italic; color:black"
    bodyStyle="background-color:##CCFFFF; font-family:sans-serif;
     font-size:90%">
   Contents of a sports feed would go here.
  </cfpod>
 </cflayoutarea>
 <cflayoutarea position="right" size="302">
  <cfpod width="300" height="20" name="thetime" title="The Weather"
    source="podweather.cfm"
    headerstyle="background-color:##DDAADD; font-style:italic;
     color:black"
    bodyStyle="background-color:##FFCCFF; font-family:sans-serif;
     font-size:80%" />
  <cfpod width="300" name="thestocks" title="What's new in business"
    headerstyle="background-color:##DDAADD; font-size:large;
     color:black; font-style:italic"
    bodyStyle="background-color:##FFCCFF; font-family:sans-serif;
     font-size:80%">
   Contents of a news feed would go here.
  </cfpod>
 </cflayoutarea>

</cflayout>
</body>
</html>
```

In this example, the podweather.cfm page contains only the following line. A more complete example would dynamically get the weather from a feed and format it for display.

```
Partly Cloudy, 76 degrees
```

Using pop-up windows

ColdFusion HTML pop-up windows have the following characteristics:

© 2014 Adobe Systems Incorporated. All rights reserved.

- They have title bars
- They float over the browser window and can be placed at an arbitrary location over the window.
- They can be modal (users cannot interact with the main window when the pop-up window is displayed) or non-modal (users can interact with both windows).
- You can specify that the user can drag, close, or resize the window.
- You can create and show a window independently. After you create the window, you can use JavaScript functions to show and hide it multiple times without having to create it again.

Display and hide windows

You display a window in the following ways:

- By using a ColdFusion `cfwindow` tag with an `initShow` attribute value of to create and show the window.
- By using a ColdFusion `cfwindow` tag with an `initShow` attribute value of `false` and calling the ColdFusion.Window.show JavaScript function to display it.
- By using ColdFusion.Window.create and ColdFusion.Window.show JavaScript functions.
  You can hide a window that is currently showing by calling the ColdFusion.Window.hide function. You can use the `ColdFusion.Window.onShow` and `ColdFusion.Window.onhide` functions to specify JavaScript functions to run when a window shows or hides.
  The following example shows how you can create, display, and hide a window. It also shows several of the configuration options that you can set, including whether the user can close, drag, or resize the window. When you run the application, the `cfwindow` tag creates and shows Window 1. You can then hide it and reshow it. To show Window 2, click the Create Window 2 button, followed by the Show Window 2 button. You can then hide and show it.
  The following example shows the main application page:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<html>
<head>
<script>
 <!--
 //Configuration parameters for window 2.
 var config =
 {x:250,y:300,height:300,width:300,modal:false,closable:false,
  draggable:true,resizable:true,initshow:false,minheight:200,minwidth:200
 }
 -->
</script>

</head>
<body>

<!--- Create a window with a title and show it. Don't allow dragging or resizing.
--->
<cfwindow name="window1" title="CFML Window" draggable="false"
  resizable="false" initshow="true" height="250" width="250" x=375 y=0>
 <p>
 This content was defined in the cfwindow tag body.
 </p>
</cfwindow>

<form>
<!--- Use the API to show and hide Window 1. --->
 <input type="button" value="Show Window1"
   onClick="ColdFusion.Window.show('window1')">
 <input type="button" value="Hide Window1"
   onClick="ColdFusion.Window.hide('window1')"><br />

<!--- Use the API to create, show, and hide Window 2 --->
 <input type="button" value="Create Window2"
   onClick="ColdFusion.Window.create('window2', 'JavaScript Window',
   'window2.cfm', config)">
 <input type="button" value="Show Window2"
   onClick="ColdFusion.Window.show('window2')">
 <input type="button" value="Hide Window2"
  onClick="ColdFusion.Window.hide('window2')">
</form>

</body>
</html>
```

The window2.cfm file with the contents of Window 2 has the following contents:

```
<cfoutput>
<p>
This content was loaded into window 2 from a URL.<br />
</p>
</cfoutput>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

Use the window show and hide events

You can use the `onShow` and `onHide` events that are triggered each time a window shows and hides to control your application. To do so, call the ColdFusion.Window.onShow and ColdFusion.Window.onHide functions to specify the event handlers. Both functions take the window name and the handler function as parameters. The event handler functions can take a single parameter, the window name.

The following example displays an alert message when a window hides or shows. The alert message includes the window name. The alert does not show when the window first appears, because the `cfwindow` tag uses the `initShow` attribute to initially display the window. An alert message does appear when the user hides the window by clicking the Toggle Window button or the close button on the window.

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<html>
<head>
 <script language="javascript">
  //Boolean value tacking the window state.
  var shown=true;

  //Functions to display an alert box when
  function onshow(name) {
   alert("window shown = " + name);
  }
  function onhide(name) {
   alert("window hidden = " + name);
  }

  //Initialize the window show/hide behavior.
  function initWindow() {
   ColdFusion.Window.onShow("testWindow", onshow);
   ColdFusion.Window.onHide("testWindow", onhide);
  }

  //Show or hide the window, depending on its current state.
  function toggleWindow() {
   if (shown) {
    ColdFusion.Window.hide("testWindow");
    shown = false;
   }
   else {
    ColdFusion.Window.show("testWindow");
    shown = true;
   }
  }

 </script>
</head>
<!-- The body tag onLoad event calls the window show/hide initializer function. -->
<body onLoad="initWindow()">

<cfwindow name="testWindow" initshow=true title="test window" closable=true> Window
contents
</cfwindow>

<cfform>
 <cfinput name="button" value="Toggle Window" onclick="javascript:toggleWindow()"
type="button"/>
</cfform>
</body>
</html>
```

Control container contents

ColdFusion provides a variety of ways to set and change container tag contents:

- You can use bind expressions in the container tag source (or for `cfdiv`, `bind`) attribute. The container then dynamically updates any time a bound control changes.
- You can call the `ColdFuson.navigate` function to change the container body to be the contents returned

© 2014 Adobe Systems Incorporated. All rights reserved.

by a specified URL. This function lets you specify a callback handler to do additional processing after the new content loads, and also lets you specify an error handler.The callback handler can be useful to provide information about a successful navigation operation. For example, you could make a pod's title bar italic to indicate loading (just before the navigate call), and use the callback handler to switch it back to normal once the navigate completes. Similarly, if a pod is showing pages from a book, the callback handler could update a page number in a separate field once a page loads

- You can use the special *controlNamebody variable to access and change the body contents for `cfpod` and `cfwindow` controls. For example, you can use the _controlNamebody`.innerHTML` property to set the body HTML. For `cfpod` and `cfwindow` tags, you can also use the _controlName_`title` to get or set the title bar contents of the control.
  These different techniques provide you with flexibility in writing your code. For example, the `ColdFuson.navigate` function and the *controlNamebody variable provide similar functionality. However, with the _controlNamebody technique, you make explicit Ajax requests to get markup for the body, and the JavaScript functions in the retrieved markup might not work properly. `ColdFusion.navigate` takes care of these issues. Therefore, limit use of the _controlName_*`body` technique to simpler use cases.
  The following example shows how you can use various techniques to change container contents. It consists of a main page and a second windowsource.cfm page with text that appears in a main page window when you click a button. The main page has a `cfpod` control, two `cfwindow` controls, and the following buttons:

- The "Simple navigate" button calls a `ColdFusion.navigate` function to change the contents of the second window.
- The "Change w2 body & title" button replaces the second window's body and title innerHTML values directly to specific strings.
- The "Change pod body" button changes the pod body innerHTML to the value of the second window's title innerHTML.
  The following example shows the main page:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<html>
<head>
<!--- Callback handler puts text in the window.cfm callback div block. --->
<script language="javascript">
 var mycallBack = function(){
  document.getElementById("callback").innerHTML = "<br><br>
 <b>This is printed by the callback handler.</b>";
 }

<!--- The error handler pops an alert with the error code and message. --->
 var myerrorHandler = function(errorCode,errorMessage){
  alert("[In Error Handler]" + "\n\n" + "Error Code: " + errorCode + "\n\n" +
   "Error Message: " + errorMessage);
 }
</script>
</head>

<body>
<cfpod height="50" width="200" title="The Title" name="theTitle">
 This is a cfpod control.
</cfpod><br>

<!--- Clicking the link runs a ColdFusion.navigate function that replaces the second
window's
   contents with windowsource.cfm. The callback handler then updates the window
   contents further. --->
<cfwindow name="w1" title="CF Window 1" initShow=true
  x=10 y=200 width="200">
 This is a cfwindow control.<br><br>
 <a href="javascript:ColdFusion.navigate('windowsource.cfm','w2',
  mycallBack,myerrorHandler);">Click</a> to navigate Window 2</a>
</cfwindow>

<cfwindow name="w2" title="CF Window 2" initShow=true
  x=250 y=200 width="200">
 This is a second cfwindow control.
</cfwindow>

<cfform>
 <!--- This button only replaces the second window body with the body of the
    windowsrc.cfm page. --->
 <cfinput type="button" name="button" value="Simple navigate"
   onClick="ColdFusion.navigate('windowsource.cfm','w2');">
 <!--- This button replaces the second window body and title content. --->
 <cfinput type="button" name="button2" value="Change w2 body & title"
   onClick="w2_body.innerHTML='New body inner HTML';w2_title.innerHTML=
    'New Title inner HTML'">
 <!--- This button puts the second window title in the pod body. --->
 <cfinput type="button" name="button3" value="Change pod body"
   onClick="theTitle_body.innerHTML=w2_title.innerHTML;">
</cfform>
</body>
</html>
```

The following example shows the windowsource.cfm page:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
This is markup from "windowsource.cfm"
<!--- The callback handler puts its output in the following div block. --->
<div id="callback"></div>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

Using menus and toolbars

The cfmenu and cfmenuitem tags let you create vertical menus and horizontal toolbars.

Define menus and toolbars

- Use a single `cfmenu` tag to define the general menu characteristics.
- Create a horizontal (toolbar) menu or vertical menu by specifying a `cfmenu type` attribute value of `horizontal` or `vertical`.
- Menus can have submenus, but only the top menu can be horizontal. All children of a horizontal menu are vertical.
- The top-level menu shows initially, a submenu shows when the user moves the mouse over the menu root in the parent menu.
- Use `cfmenuitem` tags to specify individual menu items.
- To create submenus, nest `cfmenuitem` tags. The parent tag becomes the root of the submenu.
- All `cfmenuitem` tags, except tags for dividers, must have a `display` attribute, which defines the text to show on the menu item, and can optionally have an `image` attribute.
- A horizontal menu has dividers between all items. You place dividers in vertical menus by specifying a `cfmenuitem` tag with a `divider` attribute.
- To make a menu item active, specify a `href` attribute with a URL or a JavaScript function to call when the user clicks the menu item.
  The following example shows a simple horizontal menu with submenus that uses JavaScript to change the display contents. When the user selects an end item in a menu, the text in the `div` block below the menu shows the path to the selected menu.

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<html>
<head>
</head>
<body>

<!--- The selected function changes the text in the selectedItemLabel div block to
show the
  selected item. --->
<script type="text/javascript">
 function selected(item) {
  var el = document.getElementById("selectedItemLabel");
  el.innerHTML = "You selected: " + item;
 }
</script>

<!--- A horizontal menu with nested submenus. Clicking an end item calls the
selected
  function. --->
<cfmenu name="hmenu" bgcolor="##9999ff" selectedfontcolor="##0000dd"
  selecteditemcolor="##ddddff">
 <cfmenuitem display="Home" href="javascript:selected('Home');" />
 <cfmenuitem display="File">
  <cfmenuitem display="Open...">
   <cfmenuitem display="Template" href="javascript:selected('File &gt;
    Open... &gt; Template');" />
   <cfmenuitem divider="true" />
   <cfmenuitem display="CSS" href="javascript:selected('File &gt; Open... &gt;
    CSS');" />
  </cfmenuitem>
  <cfmenuitem display="Close" href="javascript:selected('File &gt; Close');" />
 </cfmenuitem>
 <cfmenuitem display="Help">
  <cfmenuitem display="About" href="javascript:selected('Help &gt; About');" />
 </cfmenuitem>
</cfmenu>

<!--- A div with initial text.
  The selected function changes the text by resetting the innerHTML. --->
<div style=" margin-top: 100; margin-left: 10;"><span id="selectedItemLabel">
 Please select an item!</span></div>

</body>
</html>
```

Styling menus

The cfmenu and cfmenuitem tags have several attributes that let you easily control the menu appearance. These attributes consist of two types: basic and CSS style. Basic attributes, such as the `cfmenu` tag `fontColor` attribute, control individual menu characteristics. CSS style attributes let you specify a CSS style specification for a whole menu or part of a menu. The following information describes how the CSS style specifications interact and affect the menu style. For descriptions of all style-related attributes, see the cfmenu and cfmenuitem descriptions in the *CFML Reference*.

The `cfmenu` and `cfmenuitem` tags provide a hierarchy of CSS style attributes that affect different parts of the menu. The following table describes these attributes in hierarchical order:

© 2014 Adobe Systems Incorporated. All rights reserved.

Attribute	Description
cfmenu attributes	
`menuStyle`	Applies to the menu, including any parts of the menu that surround the menu items. If you do not override this style in a `cfmenu` tag `childStyle` attribute or by specifying style information in the `cfmenuitem` tags, this attribute controls the style of the top-level items.
`childStyle`	Applies to the items in the top level menu and all child menu items, including the children of submenus. This attribute lets you use a single style specification for all menu items.
cfmenuitem attributes	
`style`	Applies to the current menu item only. It is not overridden by the `childStyle`{{}}attribute.
`menuStyle`	Controls the overall style of any submenu of this menu item. This attribute controls the submenu of the current menu item, but not to any child submenus of the submenu.
`childStyle`	Applies to all child menu items of the current menu item, including the children of submenus.

In addition to these styles, consider any style-related attributes, such as `bgcolor`, that you set on the `cfmenu` tag. When you design your menu, keep in mind the following issues:

- Keep font sizes at 20 pixels or smaller. Larger sizes can result in menu text in vertical menus exceeding the menu boundaries.
- Consider how the style attributes interact. Because each menu and submenu consists of a surrounding menu area and individual child items, be careful when you choose background colors. For example, if you specify different `background-color` styles in the `cfmenu` tag's `menuStyle` and `childStyle` attributes, the menu items are one color and the surrounding menu area are a different color.
  For an application that shows some of the effects of menu style attributes, see the example in the cfmenuitem tag in the *CFML Reference*.
  ColdFusion attributes provide most style options that you are likely to require. However, you can, if necessary, modify the basic menu styles for all menus by editing the menu-related styles in the CSS files in the yui.css file. This file is located by default in the *web_root*/CFID/scripts/ajax/resources/yui directory. For more information about these styles, see the [Yahoo!
  User Interface Library menu documentation|http://developer.yahoo.com/yui/menu/].

© 2014 Adobe Systems Incorporated. All rights reserved.

Uploading files

The `cffileupload` tag lets you select multiple files and upload them to a server.

Working with the cffileupload

The `cffileupload` tag displays a dialog that lets you upload multiple files. The following are the file upload features:

- Uses callback and error handlers that lets provides control over file upload process after upload completion or if errors occur.
- Lets you style the file upload control
- Provides option to stop or continue with upload in case of errors
- Provides option to send custom response to callback and error handlers

Sending custom response to the callback and error handlers

The page/URL that handles the upload operation on the server can send back a `struct` with the keys status and message as shown here:

```
<cffile action = "upload"
destination = "#Expandpath('./upload')#"
nameconflict="makeunique">
<cfset str.STATUS = 200>
<cfset str.MESSAGE = "File Upload Successful">
<cfoutput>#serializeJSON(str)#</cfoutput>
```

The following example illustrates the error handler:

```
<cftry>
<cffile action = "upload"
destination = "#Expandpath('./upload')#">
<cfcatch type="any">
<cfset str.STATUS = 500>
<cfset str.MESSAGE = "Error occurred while uploading the file">
<cfoutput>#serializeJSON(str)#</cfoutput>
</cfcatch>
</cftry>
```

If the user tries to upload a file already present in the upload directory, it results in an error. The status and message are set to the specifications in the catch block.

Using styles

The attributes `headercolor`, `textcolor`, `bgcolor`, `titletextalign`, `titletextcolor`, and `rollovercolor` lets you style the file upload control.
The following example illustrates the styling of file upload control:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cffileupload
    url="uploadAll.cfm"
    name="myuploader3"
    align="right"
 style="headercolor:silver;textcolor:1569C7;titletextalign:right;titletextcolor:blac
k;bgcolor:74BBFB;"/>
```

The following code shows how the attribute `onUploadComplete` is used:

```
<!--- upload.cfm --->
<!--- <cffile action = "upload" destination = "#Expandpath('./upload')#"
nameconflict="makeunique"> --->
<script language="javascript">
 var uploadCompleteHandler = function(obj){
  var result = "Upload Details:" + "\n\n";
  for(var i=0;i < obj.length; i++){
   result = result + "FILENAME: " + obj[i].FILENAME + "\n" + "STATUS: " +
obj[i].STATUS + "\n" + "MESSAGE: " + obj[i].MESSAGE + "\n\n";
  }
 }
</script>
<br>
<cffileupload
 url="uploadall.cfm"
 name="myuploader"
 onUploadComplete="uploadCompleteHandler"
 maxUploadSize=100
 stopOnError=false
 />
```

Securing your uploads by default (by verifying the MIME type)

When you use the tag `cffile` for actions `upload` and `uploadAll`, the enhancements let you:

- Use fittribute `accept`.
- Check for proper MIME type of the file using `strict=true` by default.

Note

If you use multi-file uploader, there can be instances where the upload fails (for example, when MIME type or extension check fails) but the status in the uploader shows "Done". In such scenarios, you must trap the errors in the upload page and send back a serialized struct with MESSAGE and STATUS keys set to the error condition.

© 2014 Adobe Systems Incorporated. All rights reserved.

Using Ajax form controls and features

ColdFusion HTML forms and controls provide the following Ajax-based features:

- The `cfgrid`, `cfinput`, `cfselect`, `cftextarea`, and `cftree` controls support binding to get control contents.
- ColdFusion functions support asynchronous submission of forms without refreshing the entire page. When a form is in an Ajax container control, it is done automatically. Also, the `ColdFusion.Ajax.SubmitForm` JavaScript function and Ajax proxy `setForm` function support manual asynchronous submissions.
- The `cfgrid` and `cftree` tags provide HTML grids and trees that do not require a Java applet or Flash.
- The `cftextarea` control has a rich text editor option. The text editor is configurable.
- The `cfinput` tag supports a `datefield` type with an Ajax-based pop-up calendar from which user can select the date.
- The `cfinput` tag with `text` type supports an `autosuggest` attribute that lets you dynamically supply a drop-down list of field completions based on the current user input.
- The `cfinput`, `cfselect`, and `cftextarea` tags support a `tooltip` attribute that specifies a pop-up tool tip to display when the user moves the mouse over the control. The `cftooltip` tag displays a tool over any region of a page, not just a form control.

Using Ajax form controls

ColdFusion Ajax-based form controls let you submit Ajax forms in your applications without refreshing the entire page.

> **Note**
>
> When you use Ajax to submit forms asynchronously from the page, you cannot use `cfinput` or `input` tags to upload files.

Using Ajax containers for form submission

The ColdFusion Ajax container tags, `cfdiv`, `cflayoutarea`, `cfpod`, and `cfwindow`, automatically submit any forms that they contain asynchronously. When the form is submitted, the result returned by the action page replaces the contents of the container, but has no effect on the rest of the page.
The following example shows this behavior in the submitSimple.cfm page:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
</head>
<body>
<cflayout type="vbox" name="layout1">
 <cflayoutarea>
  <h3>This area is not refreshed when the form is submitted.</h3>
  <br />
 </cflayoutarea>

 <cflayoutarea>
  <h3>This form is replaced by the action page</h3>
  <cfform name="myform" format="html" action="showName.cfm">
   <cfinput type = "Text" name = "name">
    <cfinput type = "submit" name = "submit" value = "Enter name">
  </cfform>
 </cflayoutarea>
</cflayout>

</body>
</html>
```

In the following example, when you enter a name in the text input and click the Enter name button, the entered text replaces the form on the page, but the rest of the page is not refreshed. This example shows the showName.cfm action page:

```
<cfif IsDefined("Form.name")>
 <cfoutput>The Name is : <strong>#Form.name#</strong></cfoutput>
</cfif>
```

Using the cfajaxproxy SetForm function

The `SetForm` function of the proxy object created by the `cfajaxproxy` tag causes the proxy to pass the form values as arguments to the next CFC function that you call after the `SetForm` function. This way, you can pass the current values of fields in a form to a CFC function, which can then do the necessary processing and return a result. When you use the `SetForm` function, the following rules apply to the arguments in the called CFC function:

- The function does not need to specify the form fields in `cfargument` tags, and the function gets the field values passed by name.
- Form fields that have the same names as CFC arguments override the CFC argument values.
- If you do not specify form fields in the `cfargument` tags, they do not necessarily follow any declared arguments, when you use positional (array) notation to access them in the `arguments` structure.
- The `arguments` scope in the CFC function includes two fields that ColdFusion uses to control its behavior. These fields are intended for internal use, and their names might change in future releases. Both field values are set to `true`:
  - `_CF_NODEBUG` tells ColdFusion not to return debugging output in the call response.
  - `_CF_NOCACHE` tells ColdFusion to send a no cache header on the response, which prevents the browser from caching the response and ensures that every Ajax request results in a network call.
    The following example shows how to use the `SetForm` tag to submit the contents of a login form. When the user clicks the Login! button, the doLogin function calls the proxy `setForm` function and

© 2014 Adobe Systems Incorporated. All rights reserved.

then the `AuthenticationSystem.cfc validateCredentials` method. The `validateCredentials` method checks the password of the user and if it is valid, returns `true` to the proxy. Because the proxy is synchronous (the default), the `doLogin` method gets the returned value. If the value is `true`, it hides the login window; the user can then access the page contents. If the return value is `false`, the `doLogin` function displays a message in the login window title bar.

The following example shows the setForm.cfm application:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<html>
<head>
  <script type="text/javascript">
   function doLogin() {
    // Create the Ajax proxy instance.
    var auth = new AuthenticationSystem();

    // setForm() implicitly passes the form fields to the CFC function.
    auth.setForm("loginForm");
    //Call the CFC validateCredentials function.
    if (auth.validateCredentials()) {
     ColdFusion.Window.hide("loginWindow");
    } else {
     var msg = document.getElementById("loginWindow_title");
     msg.innerHTML = "Incorrect username/password. Please try again!";
    }
   }
  </script>
</head>

<body>
<cfajaxproxy cfc="AuthenticationSystem" />

<cfif structKeyExists(URL,"logout") and URL.logout>
 <cflogout />
</cfif>

<cflogin>
 <cfwindow name="loginWindow" center="true" closable="false"
    draggable="false" modal="true"
    title="Please login to use this system"
    initshow="true" width="400" height="200">
  <!--- Notice that the form does not have a submit button.
   Submit action is performed by the doLogin function. --->
  <cfform name="loginForm" format="xml">
   <cfinput type="text" name="username" label="username" /><br />
   <cfinput type="password" name="password" label="password" />
   <cfinput type="button" name="login" value="Login!" onclick="doLogin();" />
  </cfform>
 </cfwindow>
</cflogin>

<p>
This page is secured by login.
You can see the window containing the login form.
The window is modal; so the page cannot be accessed until you log in.
<ul>
 <li><a href="setForm.cfm">Continue using the application</a>!</li>
 <li><a href="setForm.cfm?logout=true">Logout</a>!</li>
</ul>
</p>
</body>
</html>
```

The following example shows the AuthenticationSystem.cfc file:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cfcomponent output="false">

 <cffunction name="validateCredentials" access="remote" returntype="boolean"
   output="false">
  <cfargument name="username" type="string"/>
  <cfargument name="password" type="string"/>

  <cfset var validated = false/>
  <!--- Ensure that attempts to authenticate start with new credentials. --->
  <cflogout/>

  <cflogin>
   <cfif arguments.username is "user" and arguments.password is "secret">
    <cfloginuser name="#arguments.username#"
      password="#arguments.password#" roles="admin"/>
    <cfset validated = true/>
   </cfif>
  </cflogin>

  <cfreturn validated/>
 </cffunction>
</cfcomponent>
```

Using the ColdFusion.Ajax.submitForm function

You can use the `ColdFusion.Ajax.submitForm` function to submit form contents to a CFML page (or other active page) at any time. For example, you could use this function to automatically save a partially completed form. This function does not support uploading a file attachment to the form.
When you use this function, you pass it the name of the form to submit and the URL of the page that processes the form. You can also specify the following optional parameters:

- A callback function that handles the returned results
- An error handler that takes two parameters, an HTTP error code and a message
- The HTTP method (by default, `POST`)
- Whether to submit the form asynchronously (by default, `true`)
  The following proof of concept example uses the `ColdFusion.Ajax.submitForm` function to submit two form fields to an asyncFormHandler.cfm page, which simply echoes the form values. The callback handler displays an alert with the returned information.

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<html>
<head>
<!--- The cfajaximport tag is required for the submitForm function to work
   because the page does not have any Ajax-based tags. --->
<cfajaximport>

<script>
 function submitForm() {
  ColdFusion.Ajax.submitForm('myform', 'asyncFormHandler.cfm', callback,
   errorHandler);
 }

 function callback(text)
 {
  alert("Callback: " + text);
 }

 function errorHandler(code, msg)
 {
  alert("Error!!! " + code + ": " + msg);
 }
</script>

</head>
<body>

<cfform name="myform">
 <cfinput name="mytext1"><br />
 <cfinput name="mytext2">
</cfform>

<a href="javascript:submitForm()">Submit form</a>
</body>
</html>
```

The asynchFormHandler.cfm page consists of a single line, as follows:

```
<cfoutput>Echo: #form.mytext1# #form.mytext2#</cfoutput>
```

Using the ColdFusion.navigate function to submit a form

The `ColdFusion.navigate` JavaScript function can submit a form to a URL and have the returned output appear in a specified container control, such as a `cfdiv`, `cflayout`, `cfpod`, or `cfwindow` tag. This function lets you populate a control other than the one that contains the form when the user submits the data. You can also use the function to submit the form asynchronously when a user performs an action outside the form, such as clicking a menu item.
For an example that uses this function, see the ColdFusion.navigate function in the *CFML Reference*.

Using HTML grids

The ColdFusion HTML `cfgrid` control lets you use a bind expression to dynamically populate the grid. HTML grids

© 2014 Adobe Systems Incorporated. All rights reserved.

that use bind expressions are paged; as users navigate from page to page of the grid, the grid dynamically gets the data for only the required page from the data source. You also use bind expressions when you let users edit form contents, and other ColdFusion controls can bind to the grid. Also, HTML grids provide several JavaScript functions that you can use to manage and manipulate the grids.
You can also create a static HTML grid by specifying a `cfgrid` tag that does not use a bind expression. With static grids, all data is initially available.
In HTML mode, if you move the mouse over a column heading, a down arrow button appears. Clicking the button displays a list with the following options:

- Sort the grid in ascending or descending order based on the column's content.
- Select the columns to display.
- If the grid has a `groupfield` attribute: turn grouping off and on and group by the column value.
  If you specify `selectMode="edit"` for an HTML grid, the grid displays Insert, Save, Cancel, and Delete buttons to the bottom bar. The Insert button opens a new editable row. The Save button commits any changes to the bind source. The Cancel button rolls back any changes that have not been saved. The Delete button deletes a selected row. You need not press the Save button after clicking the Delete button.

Dynamically filling form data

HTML grids can dynamically fill the grid data by using a `bind` attribute with a bind expression that calls a CFC or JavaScript function, or a URL. The bind expression uses bind parameters to specify dynamic information provided by the grid and the values of any other form field attributes.
Pass the following bind parameters to the bind expression. If you omit any of the parameters in the function call or URL, you get an error. These parameters send information about the grid and its state to the data provider function. The data for these parameters is provided automatically. You do not set any values manually.

Parameter name	Description
`cfgridpage`	The number of the page for which to retrieve data.
`cfgridpagesize`	The number of rows of data in the page. The value of this parameter is the value of the `pageSize` attribute.
`cfgridsortcolumn`	The name of the column that determines the sorting order of the grid. This value is set only after the user clicks a column heading.
`cfgridsortdirection`	The direction of the sort, may be 'ASC' (ascending) or 'DESC' (descending). This value is set only after the user clicks a column heading.

Note

The `cfgridsortcolumn` and `cfgridsortdirection` parameters can be empty if the user or application has not sorted the grid, for example, by clicking a grid column header.

For more information on binding and bind parameters, see Using Ajax Data and Development Features in the *CFML Reference*.
You can use optional parameters to specify additional information to pass to the called function. These parameters provide data that the called function requires to determine the data to return. For example, if the function returns the cities in a state, you would pass it the state name. Any or all of the optional function parameters can be bind

© 2014 Adobe Systems Incorporated. All rights reserved.

parameters. A state name, for example, could come from the selection in a states `cfselect` control.
If you do not want the grid to refresh automatically when other controls change, you can use the `@none` specifier on all optional bind parameters. Doing this, prevents automatic updating of the grid based on the bound control values. Use the ColdFusion.Grid.refresh JavaScript function to explicitly refresh the grid contents. For more information on the use of the `@none` specifier and explicitly refreshing the control, see *Specifying bind parameters* in Binding data to form fields.
If the grid supports user sorting of the data (the `sort` attribute is `true`), the function called by the bind expression must return data in the desired sorted order, and must use the values of the `cfgridsortcolumn` and `cfgridsortdirection` bind parameters to determine the order. Even if you do not allow user sorting, still pass these parameters to the function; otherwise, you get an error. Also, your function or action page must handle cases where these parameters are empty strings, because their values are not set until the user selects a column header to sort the grid, or you call the JavaScript `ColdFusion.Grid.sort` function.
The format of the returned data depends on how you get the data:

Bind type	Return value
CFC	A ColdFusion structure. ColdFusion automatically converts the structure for return to the caller. Alternatively, you can return a JSON representation of the structure.
URL	A JSON representation of a structure. No other body contents is allowed.
JavaScript	A JavaScript object.

When you specify a CFC in the `bind` attribute, use the `queryConvertForGrid` function to convert a query directly into a structure that you can use as your CFC return value.
When you specify a CFML page in the `bind` attribute, use the `queryConvertForGrid` function to convert a query into a structure, and then use the `serializeJSON` function to convert the structure into a JSON representation.
If you manually create a JavaScript object or its JSON representation, it must have two top-level keys:

- `TOTALROWCOUNT`: The total number of rows in the query data set being returned. This value is the total number of rows of data in all pages in the grid, and not the number of rows in the current page.
- `QUERY`: The contents of the query being returned{{.}} The `QUERY`value must also be an object with two keys:
  - `COLUMNS`: An array of the column names.
  - `DATA`: A two-dimensional array, where the first dimension corresponds to the rows and the second dimension corresponds to the field values, in the same order as the `COLUMNS` array.

Note

If a CFC manually creates a return structure, the `QUERY` value can be a ColdFusion query object; ColdFusion automatically converts it for remote access.

The following example defines an object that a JavaScript bind function can return to provide the data for a `cfgrid` tag:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
var myobject =
 {"TOTALROWCOUNT":6,"QUERY":{"COLUMNS":["EMP_ID","FIRSTNAME",
  "EMAIL"],"DATA":[[1,"Carolynn","CPETERSON"],
  [2,"Dave","FHEARTSDALE"], [3,"Linda","LSTEWART"],
  [4,"Aaron","ASMITH"], [5,"Peter","PBARKEN"],
  [6,"Linda","LJENNINGS"],]}};
```

The following example uses a bind expression and a CFC to populate a dynamic, paged, data grid. The CFML page contains the following form:

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
</head>

<body>
<cfform name="form01">
 <cfgrid format="html" name="grid01" pagesize=5 sort=true
   bind="cfc:places.getData({cfgridpage},{cfgridpagesize},
   {cfgridsortcolumn},{cfgridsortdirection})">
  <cfgridcolumn name="Emp_ID" display=true header="eid" />
  <cfgridcolumn name="FirstName" display=true header="Name"/>
  <cfgridcolumn name="Email" display=true header="Email" />
 </cfgrid>
</cfform>
</body>
</html>
```

The places.cfc file looks as follows. Notice that the query gets the full data set each time the function gets called. the `QueryConvertForGrid` function selects and returns only the required page of data:

```
<cfcomponent>
 <cffunction name="getData" access="remote" output="false">
  <cfargument name="page">
  <cfargument name="pageSize">
  <cfargument name="gridsortcolumn">
  <cfargument name="gridsortdirection">
  <cfquery name="team" datasource="cfdocexamples">
   SELECT Emp_ID, FirstName, EMail
   FROM Employees
   <cfif gridsortcolumn neq "" or gridsortdirection neq "">
    order by #gridsortcolumn# #gridsortdirection#
   </cfif>
  </cfquery>
  <cfreturn QueryConvertForGrid(team, page, pageSize)>
 </cffunction>
</cfcomponent>
```

The following example is equivalent to the previous one, but uses a URL bind expression in the main page and a CFML page to return the data.

© 2014 Adobe Systems Incorporated. All rights reserved.

The main page contains the following form:

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
</head>

<body>
<cfform name="form01">
 <cfgrid format="html" name="grid01" pagesize=5 sort=true
  bind="url:getdata.cfm?page={cfgridpage}&pageSize={cfgridpagesize}
   &sortCol={cfgridsortcolumn}&sortDir={cfgridsortdirection}">
  <cfgridcolumn name="Emp_ID" display=true header="eid" />
  <cfgridcolumn name="FirstName" display=true header="Name"/>
  <cfgridcolumn name="Email" display=true header="Email" />
 </cfgrid>
</cfform>
</body>
</html>
```

The following example shows the getdata.cfm page:

```
<!--- Empty string; the default end of the query SQL. --->
<cfset queryEnd="">

<cfquery name="team" datasource="cfdocexamples">
 SELECT Emp_ID, FirstName, EMail
 FROM Employees
 <cfif sortcol neq "" or sortdir neq "">
  order by #sortcol# #sortdir#
 </cfif>
</cfquery>

<!--- Format the query so that the bind expression can use it. --->
<cfoutput>#serializeJSON(QueryConvertForGrid(team, page, pageSize))#
</cfoutput>
```

If your database lets you specify SQL to retrieve only the required page of data in a query, you can optimize efficiency by using such a query. Do not use the QueryConvertForGrid function. Instead, manually create the return structure and return only the single page of data. Ensure that you set the TotalRowCount field to the number of rows in the entire data set, not the number of rows in the returned page of data.

Using the bindOnLoad attribute

The `bindOnLoad` attribute causes a control to execute its bind expression immediately when it loads, and not wait until the event that normally triggers the bind expression evaluation to occur. This way, the control can be filled with an initial value. This attribute is `false` by default for all ColdFusion Ajax controls that have the attribute, except `cfd iv` and `cfgrid`, for which it is `true` by default. Having a `true` `bindOnLoad` value on these controls ensures that they are populated when they load.
When a control with a `true` `bindOnLoad` attribute is bound to a control that also binds when the page loads, the first and second control load themselves at the `onLoad` page event. Then the first control loads itself again in response to a change event from the second control when that control completes loading. So, the first control makes

© 2014 Adobe Systems Incorporated. All rights reserved.

two Ajax calls, whereas it must make only one, when the second control finished loading.
Because the `cfinput`, `cfselect`, and `cftextarea` control `bindOnLoad` attributes are `false` by default, you do not encounter any problems if a `cfgrid` or `cfdiv` tag binds to any of these controls and you do not explicitly set the `bindOnLoad` attributes. However, if the control does set its `bindOnLoad` attribute to `true`, set the `cfgrid` or `cfdiv` attribute to `false` to ensure that the control only fetches data when the control that it is bound to returns.
You can also get a double loading if a grid binds to a Spry data set. By default, the grid and data set load data at page load, and then the grid loads data again in response to a selection change event from the data set when it sets focus to its first row. Set `bindOnLoad` to `false` to ensure that the grid fetches data only when it receives a selection `change` event from the data set.

Dynamically editing grid contents

When you use a bind expression to get `cfgrid` data dynamically, you can also update the data source dynamically with user input, without requiring the user to submit the form. You can use dynamic updating to update or delete data in the data source. (To edit `cfgrid` data, select the contents of a field and type the new value; to delete a row, select a field in the row and click the delete button at the bottom of the grid.)
You *cannot* insert new rows directly in a grid that uses a bind expression. To add rows, enter the data in a form, and make sure that the grid refreshes after the form has been submitted.
To update or delete data dynamically, do the following:

- Specify `selectmode="edit"` in the `cfgrid` tag. This lets the user edit the grid.
- Specify an `onChange` attribute in the `cfgrid` tag. The attribute must use a bind expression to specify a CFC method, JavaScript function, or URL of a page that updates the data source. The bind expression has the same format as the bind expression described in *Dynamically filling form data* below; however, it must take the following bind parameters that the grid automatically passes. These parameters send information about the grid and its state to the `onChange` function.

Parameter name	Description
`cfgridaction`	The action performed on the grid. 'U' for update, or 'D' for delete.
`cfgridrow`	A structure or JavaScript Object whose keys are the column names and values are the original values of the updated or deleted row.
`cfgridchanged`	A structure or JavaScript Object with a single entry, whose key is the name of the column with the changed value, and whose value is the new value of the field. If the grid action is delete, this structure exists but is empty.

When you update data dynamically, you can also use the `onError` attribute to specify the name of a JavaScript function to handle any errors that result in a CFC or URL returning an HTTP error status. The method must take two parameters: the HTTP error code and a text message that describes the error. The following example shows an `onError` handler function:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<script type="text/javascript">
 function errorhandler(id,message) {
  alert("Error while updating \n Error code: "+id+" \nMessage:
  "+message);}
</script>
```

The following `example` displays the members of a department and lets users edit the data in the fields. When the focus leaves the edited field an onChange event triggers and the form calls the `editData` CFC function to update the data source.

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<script type="text/javascript">
 function errorhandler(id,message) {
  alert("Error while updating\n Error code: "+id+"\n Message: "+message);
 }
</script>
</head>

<body>
<cfform name="form01">
 <cfgrid format="html" name="grid01" pagesize=11
   stripeRows=true stripeRowColor="gray"
   bind="cfc:places.getData({cfgridpage},{cfgridpagesize},
    {cfgridsortcolumn},{cfgridsortdirection})"
   delete="yes" selectmode="edit"
   onchange="cfc:places.editData({cfgridaction},{cfgridrow},{cfgridchanged})">
  <cfgridcolumn name="Emp_ID" display=true header="Employee ID"/>
  <cfgridcolumn name="FirstName" display=true header="Name"/>
  <cfgridcolumn name="Email" display=true header="Email"/>
 </cfgrid>
</cfform>
</body>
</html>
```

The `getData` function is identical to the `getData` function in *Dynamically filling form data* below. This example shows the `editData` function in the CFC:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cffunction name="editData" access="remote" output="false">
 <cfargument name="gridaction">
 <cfargument name="gridrow">
 <cfargument name="gridchanged">

 <cfif isStruct(gridrow) and isStruct(gridchanged)>
  <cfif gridaction eq "U">
   <cfset colname=structkeylist(gridchanged)>
   <cfset value=structfind(gridchanged,#colname#)>
   <cfquery name="team" datasource="cfdocexamples">
    update employees set <cfoutput>#colname#</cfoutput> =
     '<cfoutput>#value#</cfoutput>'
    where Emp_ID = <cfoutput>#gridrow.Emp_ID#</cfoutput>
   </cfquery>
  <cfelse>
   <cfquery name="team" datasource="cfdocexamples">
    delete from employees where emp_id = <cfoutput>#gridrow.Emp_ID#
     </cfoutput>
   </cfquery>
  </cfif>
 </cfif>
</cffunction>
```

Binding controls to grid contents

You can bind the contents of a form control to the data in a grid field by specifying a bind parameter as the form control `bind` attribute value. To do so, use the following syntax:

```
<cfinput name="name" type="text" bind="{gridName.columnName}">
```

By default, each time the selected row in the grid changes, the bind parameter is re-evaluated, and the control value changes to the value of the specified column of selected grid cell.

Grid JavaScript functions

You can use the following JavaScript functions to manage an HTML format grid:

Function	Description
`ColdFusion.Grid.getGridObject`	Gets the underlying Ext JS JavaScript library object.
`ColdFusion.Grid.refresh`	Manually refreshes a displayed grid.
`ColdFusion.Grid.sort`	Sorts the grid.

For more information, see the ColdFusion.Grid.getGridObject, ColdFusion.Grid.refresh, and ColdFusion.Grid.sort functions in the *CFML Reference*.

Using the Boolean column and grouping

© 2014 Adobe Systems Incorporated. All rights reserved.

The example in this section shows how to use the Boolean column. The code also illustrates how to group data in a grid on a selected grid column.
This example uses all types of Boolean representations. For grouping, `groupField` is set to active and the grid data is grouped accordingly.

```
<cfset emps = querynew("firstname,department, salary,active")>
<cfset queryaddrow(emps,10)>
<cfset querysetcell(emps,"firstname","Debra",1)>
<cfset querysetcell(emps,"department","Accounting",1)>
<cfset querysetcell(emps,"salary","100000",1)>
<cfset querysetcell(emps,"active","Y",1)>
<cfset querysetcell(emps,"firstname","Doherty",2)>
<cfset querysetcell(emps,"department","Finance",2)>
<cfset querysetcell(emps,"salary","120000",2)>
<cfset querysetcell(emps,"active","Yes",2)>
<cfset querysetcell(emps,"firstname","Ben",3)>
<cfset querysetcell(emps,"department","Law",3)>
<cfset querysetcell(emps,"salary","200000",3)>
<cfset querysetcell(emps,"active","true",3)>
<cfset querysetcell(emps,"firstname","Aaron",4)>
<cfset querysetcell(emps,"department","Accounting",4)>
<cfset querysetcell(emps,"salary","200000",4)>
<cfset querysetcell(emps,"active","1",4)>
<cfset querysetcell(emps,"firstname","Josh",5)>
<cfset querysetcell(emps,"department","CF",5)>
<cfset querysetcell(emps,"salary","400000",5)>
<cfset querysetcell(emps,"active",true,5)>
<cfset querysetcell(emps,"firstname","Peterson",6)>
<cfset querysetcell(emps,"department","Accounting",6)>
<cfset querysetcell(emps,"salary","150000",6)>
<cfset querysetcell(emps,"active","0",6)>
<cfset querysetcell(emps,"firstname","Damon",7)>
<cfset querysetcell(emps,"department","Finance",7)>
<cfset querysetcell(emps,"salary","100000",7)>
<cfset querysetcell(emps,"active","N",7)>
<cfset querysetcell(emps,"firstname","Tom",8)>
<cfset querysetcell(emps,"department","CF",8)>
<cfset querysetcell(emps,"salary","100000",8)>
<cfset querysetcell(emps,"active","false",8)>
<cfset querysetcell(emps,"firstname","Adam",9)>
<cfset querysetcell(emps,"department","CF",9)>
<cfset querysetcell(emps,"salary","300000",9)>
<cfset querysetcell(emps,"active",false,9)>
<cfset querysetcell(emps,"firstname","Sean",10)>
<cfset querysetcell(emps,"department","CF",10)>
<cfset querysetcell(emps,"salary","250000",10)>
<cfset querysetcell(emps,"active","No",10)>
<cfform name="form01">
 <cfgrid format="html"  insert="yes" insertButton="Add Row"
  name="grid01"
  selectmode="edit"
  width=600
  collapsible="true"
  title="Employees"
  autowidth="yes"
  query="emps"
  sort="yes"
```

© 2014 Adobe Systems Incorporated. All rights reserved.

```
  groupField="active">
   <cfgridcolumn name="FirstName" header="FirstName"/>
   <cfgridcolumn name="Department" header="Department" />
   <cfgridcolumn name="Salary" display=true header="Salary" type="numeric"
values="1000000,1200000" valuesdisplay="1000000,1200000"/>
   <cfgridcolumn name="Active"  display=true header="Contract" type="boolean" />
```

© 2014 Adobe Systems Incorporated. All rights reserved.

```
 </cfgrid>
</cfform>
```

Using the date column

The following example shows how to use the date column. In the code, `startdatecolumn` is of type `date`. A `mask` , `Y/m/d` is used. `Y` is year in four digits, `m` months with leading zero, and `d` the days with leading zero.

```
<!--- using cfgridcolum type="date" --->
<cfset emps = querynew("firstname,department, salary,startdate")>
<cfset queryaddrow(emps,3)>
<cfset querysetcell(emps,"firstname","Debra",1)>
<cfset querysetcell(emps,"department","Accounting",1)>
<cfset querysetcell(emps,"salary","100000",1)>
<cfset querysetcell(emps,"startdate","2009/1/1",1)>
<cfset querysetcell(emps,"firstname","Doherty",2)>
<cfset querysetcell(emps,"department","Finance",2)>
<cfset querysetcell(emps,"salary","120000",2)>
<cfset querysetcell(emps,"startdate","2005/2/21",2)>
<cfset querysetcell(emps,"firstname","Ben",3)>
<cfset querysetcell(emps,"department","Law",3)>
<cfset querysetcell(emps,"salary","200000",3)>
<cfset querysetcell(emps,"startdate","2008/03/03",3)>
<cfform name="form01">
 <cfgrid format="html"  insert="yes" insertButton="Add Row"
  name="grid01"
  selectmode="edit"
  width=600
  collapsible="true"
  title="Employees"
  autowidth="yes"
  query="emps"
  sort="yes"
  groupField="department">
   <cfgridcolumn name="FirstName" header="FirstName"/>
   <cfgridcolumn name="Department" header="Department" />
   <cfgridcolumn name="Salary" display=true header="Salary" type="numeric"
values="1000000,1200000" valuesdisplay="1000000,1200000"/>
   <cfgridcolumn name="StartDate"  display=true header="StartDate" type="date"
mask="Y/m/d"/>
 </cfgrid>
</cfform>
```

For details of various types of masks that can be used, see the *CFML Reference Guide*.

Datehandling when the attribute mask is used in ColdFusion 9.0.1

If the attribute `mask` is applied to a datefield column in an HTML grid, ColdFusion converts the date to an intermediate format as shown here:
`MMMMM, dd yyyy HH:mms`
for example,
`January, 19 2005 07:35:42`
This is required for proper date conversion and is applicable both when data is sent to the server (for example, when

© 2014 Adobe Systems Incorporated. All rights reserved.

using an onChange grid event) and when data is received from the server (for example, populating a date field in a grid). Therefore, in some cases, users might have to format the date if they are updating a date column in the database.

> **Note**
>
> Date values which are NULL are sent as empty strings when the form is submitted. In such cases, set the value to NULL explicitly while updating the date column in the database.

Using HTML trees

An HTML cftree tag creates an Ajax-based tree data representation that you can populate from a query or a bind expression. The behavior with a query is equivalent to the behavior of applet or Flash trees. Bind expressions let you populate the tree based on the values of other controls or Spry data sets. Also, when you use a bind expression, the tree loads dynamically, getting only the data required for the current display.

Populating the tree using a bind expression

You use the `bind` attribute and bind expressions to dynamically and incrementally load and display tree data as the user navigates the tree. The child tree items do not exist until the parent node expands. This behavior avoids prefilling a tree with large amounts of data. It lets the tree children change dynamically (you can optionally get the children each time the item expands) and can enhance application responsiveness.
For more information about binding and bind parameters, see Binding data to form fields.
Bind expressions in trees work in the following ways:

- If you use a bind expression, the `cftree` tag can have only a single `cftreeitem` tag. Therefore, the function or URL called by the bind expression must be able to populate all levels of the tree.
- When a tree item expands, the CFC or JavaScript function or active page specified by the `bind` attribute returns an array with the values for the child nodes of the item. The dynamic tree code on the client constructs the child items by using these values.
- When a control to which the tree is bound generates an event that the tree is listening for, the tree is refreshed. For example, if the tree uses a bind expression that includes a select box as a bind parameter, the tree collapses to the root nodes when the selected value in the select box changes.
  When you use a bind expression to populate a `cftree` control, specify a CFC function, JavaScript function, or URL, and pass the following bind parameters. If you omit either of the parameters from your function call or URL, you get an error. These parameters provide information about the tree and its state, and are automatically provided by the control.

Bind parameter	Description
`{cftreeitempath}`	Passes the path of the current (parent) node to the method, which uses it to generate the next node.
`{cftreeitemvalue}`	Passes the current tree item value (normally the `value` attribute)

The called function or URL cannot return nested arrays and structures, that is, it can only return a single level of items.
When a function or URL is first called to populate the root-level tree items, the value passed in the cftreeitemvalue variable is the empty string. Your bind function can test for an empty string to determine that it is populating the root level of the tree.
The `@none` event specifier is also useful if you use the ColdFusion.Tree.refresh JavaScript function to manually

© 2014 Adobe Systems Incorporated. All rights reserved.

refresh the tree. When you call the `Refresh` function, the bind expression fetches data from all bind parameters, including `@none` parameters. If you specify `@none` in all bind parameters that specify other controls, the tree does not respond automatically to changes in the other controls, but it does pick up data from the bind parameters when you use the `ColdFusion.Tree.Referesh` function to explicitly refresh the tree.
The format of the data that the function or URL in a bind expression must return depends on the type of bind expression

Bind type	Return value
CFC	A ColdFusion array of structures. ColdFusion automatically converts the structure to JSON format when it returns the result to the caller. Alternatively, you can return a JSON representation of the structure.
JavaScript	A JavaScript Array of Objects.
URL	A JSON representation of an array of structures. No other body content is allowed.

Each structure in the array of structures or objects defines the contents and appearance of the node for a child item. Each structure must have a VALUE field, and can have the following fields. Except for LEAFNODE, these structure keys correspond to cftreeitem attributes.

- DISPLAY
- EXPAND
- HREF
- IMG
- IMGOPEN
- LEAFNODE
- TARGET

Note

If a CFC does not return a value field, you do not get an error, but the tree does not work properly.

The LEAFNODE structure element is only used in the bind response structures. It must be a Boolean value that identifies whether the node is a leaf. If the value is `true`, the tree does not show a +/- expansion indicator in front of the node, and users cannot expand the node.
If your bind expression specifies a JavaScript function, the function must use all-uppercase letters for the field names; for example, use `VALUE` and `DISPLAY`, not `value` and `display`. ColdFusion uses all capital letters in the structure key names. ColdFusion is not case-sensitive, so CFCs can use lowercase letters for the field names; JavaScript is case-sensitive, so the JavaScript function must match the uppercase field names.
If you use a URL to get the tree items from a CFML page, you can use the `serializeJSON` function to convert the array to JSON format. If the array with the tree items is named itemsArray, for example, the following line specifies the page output:

```
<cfoutput>#serializeJSON(itemsArray)#</cfoutput>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

Example 1: a simple tree

The following simple example creates a simple hierarchical tree of unlimited depth, with one node per level. Each node label (specified by the `display` attribute) identifies the node depth:
The following example shows the CFML page:

```
<cfform name="testform">
 <cftree name="t1" format="html">
  <cftreeitem bind="cfc:makeTree.getNodes({cftreeitemvalue},{cftreeitempath})">
 </cftree>
</cfform>
```

The following example shows the maketree.cfc file with the `getNodes` method that is called when the user expands a node:

```
<cfcomponent>
 <cffunction name="getNodes" returnType="array" output="no" access="remote">
  <cfargument name="nodeitemid" required="true">
  <cfargument name="nodeitempath" required="true">
  <!--- The initial value of the top level is the empty string. --->
  <cfif nodeitemid IS "">
   <cfset nodeitemid =0>
  </cfif>
  <!--- Create an array with one element defining the child node. --->
  <cfset nodeArray = ArrayNew(1)>
  <cfset element1 = StructNew()>
  <cfset element1.value = nodeitemid + 1>
  <cfset element1.display = "Node #nodeitemid#">
  <cfset nodeArray[1] = element1>
  <cfreturn nodeArray>
 </cffunction>
</cfcomponent>
```

Handling leaf nodes

Code that returns the information for leaf nodes of the trees must always set the LEAFNODE structure field to `true`. This prevents the tree from displaying a + expansion indicator in the tree leaf node tree entries and from attempting to expand the node. The following example shows how you use the LEAFNODE field.

Example 2: a more complex tree with leaf node handling

The following tree uses the cfartgallery database to populate a tree where the top level is the art medium, the second level is the artist, and the leaf nodes are individual works of art. When the user clicks an artwork, the application shows the art image.
This example shows how to generate return values that are specific to the level in the tree and the parent value. It also shows the use of the LEAFNODE return structure element.
In this application, the CFC return structure keys are specified in lowercase letters, and ColdFusion automatically converts them to uppercase. Notice that the database contains entries only for the painting, sculpture, and photography categories, so just those top-level tree nodes have child nodes.
The following example shows the main application page:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<!--- The loadimage function displays the image of the selected art.
  It is called when the user clicks the image item. --->
<script>
 function loadImage(img) {
  var imgURL = '<img src="/cfdocs/images/artgallery/'+img+'">';
  var imgDiv = document.getElementById('image');
   imgDiv.innerHTML = imgURL;
 }
</script>

</head>
<body>

<!--- The form uses a table to place the tree and the image. --->
<cfform name="ex1" action="ex1.cfm" method="post">
 <table>
  <tr valign="top">
   <td>
    <cftree name="mytree" format="html">
     <!--- When you use a bind expression, you must have only one
       cftreeitem, which populates the tree level. --->
     <cftreeitem bind="cfc:tree.getItems({cftreeitempath},
      {cftreeitemvalue})">
    </cftree>
   </td>
   <td>
    <div id="image"></div>
   </td>
  </tr>
 </table>
</cfform>
</body>
</html>
```

The following example shows the tree.cfc file:

```
<cfcomponent output="false">

<cfset variables.dsn = "cfartgallery">

<!--- Function to populate the current level of the tree. --->
<cffunction name="getItems" returnType="array" output="false" access="remote">
 <cfargument name="path" type="string" required="false" default="">
 <cfargument name="value" type="string" required="false" default="">
 <cfset var result = arrayNew(1)>
 <cfset var q = "">
 <cfset var s = "">

 <!--- The cfif statements determine the tree level. --->
 <!--- If there is no value argument, the tree is empty. Get the media types. --->
 <cfif arguments.value is "">
  <cfquery name="q" datasource="#variables.dsn#">
```

© 2014 Adobe Systems Incorporated. All rights reserved.

```
  SELECT mediaid, mediatype
  FROM media
  </cfquery>
  <cfloop query="q">
   <cfset s = structNew()>
   <cfset s.value = mediaid>
   <cfset s.display = mediatype>
   <cfset arrayAppend(result, s)>
  </cfloop>

 <!--- If the value argument has one list entry, it is a media type. Get the artists
for
   the media type.--->
 <cfelseif listLen(arguments.value) is 1>
  <cfquery name="q" datasource="#variables.dsn#">
  SELECT artists.lastname, artists.firstname, artists.artistid
  FROM art, artists
  WHERE art.mediaid = <cfqueryparam cfsqltype="cf_sql_integer"
    value="#arguments.value#">
  AND art.artistid = artists.artistid
  GROUP BY artists.artistid, artists.lastname, artists.firstname
  </cfquery>
  <cfloop query="q">
   <cfset s = structNew()>
   <cfset s.value = arguments.value & "," & artistid>
   <cfset s.display = firstName & " " & lastname>
   <cfset arrayAppend(result, s)>
  </cfloop>

 <!--- We only get here when populating an artist's works. --->
 <cfelse>
  <cfquery name="q" datasource="#variables.dsn#">
  SELECT art.artid, art.artname, art.price, art.description,
    art.largeimage, artists.lastname, artists.firstname
  FROM art, artists
  WHERE art.mediaid = <cfqueryparam cfsqltype="cf_sql_integer"
    value="#listFirst(arguments.value)#">
  AND art.artistid = artists.artistid
  AND artists.artistid = <cfqueryparam cfsqltype="cf_sql_integer"
    value="#listLast(arguments.value)#">
  </cfquery>
  <cfloop query="q">
   <cfset s = structNew()>
   <cfset s.value = arguments.value & "," & artid>
   <cfset s.display = artname & " (" & dollarFormat(price) & ")">
   <cfset s.href = "javaScript:loadImage('#largeimage#');">
   <cfset s.children=arrayNew(1)>
   <!--- leafnode=true prevents node expansion and further calls to the
    bind expression. --->
   <cfset s.leafnode=true>
   <cfset arrayAppend(result, s)>
  </cfloop>

 </cfif>

 <cfreturn result>
</cffunction>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

```
</cfcomponent>
```

Binding other controls to a tree

ColdFusion tags that use bind expressions can bind to the selected node of a tree by using the following formats:

- {_form_*tree*.node} retrieves the value of the selected tree node.
- {_form_*tree*.path} retrieves the path of the selected tree node. If the `completePath` attribute value is `true`, the bound path includes the root node.
  The bind expression is evaluated each time a `select` event occurs on an item in the tree. If you specify any other event in the bind parameter, it is ignored.

Tree JavaScript functions

You can use the following JavaScript functions to manage an HTML tree:

Function	Description
`ColdFusion.Tree.getTreeObject`	Gets the underlying Yahoo User Interface Library TreeView JavaScript object.
`ColdFusion.Tree.refresh`	Manually refreshes a tree.

For more information, see the ColdFusion.Tree.getTreeObject and ColdFusion.Tree.refresh functions in the *CFML Reference*.

Using the rich text editor

The ColdFusion rich text editor lets users enter and format rich HTML text by using an icon-driven interface based on the open source FCKeditor Ajax widget. The editor includes numerous formatting controls, and icons for such standard operations as searching, printing, and previewing text. Text editor controls are not covered. For detailed information on the editor icons and controls, see http://wiki.fckeditor.net/UsersGuide.

> **Note**
>
> Do not bind to a rich text area on load of a page, for example, from another control, such as a text box.

The following example shows a simple rich text editor. When a user enters text and clicks the Enter button, the application refreshes and displays the formatted text above the editor region.

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
</head>

<body>
<!--- Display the text if the form has been submitted with text. --->
<cfif isdefined("form.text01") AND (form.text01 NEQ "")>
 <cfoutput>#form.text01#</cfoutput><br />
</cfif>

<!--- A form with a basic rich text editor and a submit button. --->
<cfform name="form01" >
 <cftextarea richtext=true name="text01" />
 <cfinput type="submit" value="Enter" name="submit01"/>
</cfform>
</body>
</html>
```

Note

If you use the rich text editor in your pages, you cannot configure your web server to have ColdFusion process files with the .html or .htm extensions. The default HTML processor must handle pages with these extensions.

Configuring the rich text editor

You can customize the rich text editor in many ways. The `cftextarea` attributes support some basic customization techniques. For more detailed information, see http://wiki.fckeditor.net/.

Defining custom toolbars

You can use the following techniques to control the appearance of the toolbar:

- Specify the toolbar name in the `toolbar` attribute
- Create custom toolbars in the fckconfig.js file.
  The editor has a single toolbar consisting of a set of active icons and fields, and separators. The `toolbar` att ribute lets you select the toolbar configuration. The attribute value specifies the name of a toolbar set, which you define in a FCKConfig.ToolbarSets entry in the *cf_webRoot*/CFIDE/scripts/ajax/FCKEditor/fckconfig.js file.
  The rich text editor comes configured with two toolbar sets: the Default set, which contains all supported editing controls, and a minimal Basic set. By default, the editor uses the Default set. To create a custom toolbar named BasicText with only text-editing controls, create the following entry in the fckconfig.js file, and specify `toolbar="BasicText"` in the `textarea` tag.

© 2014 Adobe Systems Incorporated. All rights reserved.

```
FCKConfig.ToolbarSets["BasicText"] = [
  ['Source','DocProps','-','NewPage','Preview'],
  ['Cut','Copy','Paste','PasteText','PasteWord','-','Print','SpellCheck'],
  ['Undo','Redo','-','Find','Replace','-','SelectAll','RemoveFormat'],
  ['Bold','Italic','Underline'],
  ['Outdent','Indent'],
  ['JustifyLeft','JustifyCenter','JustifyRight','JustifyFull'],
  '/',
  ['Style','FontFormat','FontName','FontSize'],
  ['TextColor','BGColor'],
  ['FitWindow','-','About']
 ];
```

This configuration defines a toolbar with two rows that contain a subset of the full tool set designed to support basic text editing.
Follow these rules when you define a toolbar:

- Start the definition with FCKConfig.ToolbarSets.
- Specify the toolbar name in double quotation marks and brackets (["""]). Use this name, case correct, in the `cf textarea` tag `toolbar` attribute.
- Follow the toolbar name with an equal sign (=).
- Place all the toolbar controls inside a set of brackets, and follow the definition with a semicolon (;).
- Group controls in brackets.
- Place each entry in single quotation marks (') and separate the entries with commas (,).
- Use the hyphen (-) character to specify a separator.
- Use a forward slash (\/) character to start a new row.
  For a complete list of the valid toolbar entries, see the `Default` configuration in fckconfig.js.

Defining custom styles

You can add custom styles that users choose in the Styles selector and apply to selected text. To create a custom style, add a `Style` element to /CFIDE/scripts/ajax/FCKEditor/fckstyles.xml. The `Style` XML element has the following format:

- The `name` attribute specifies the name that appears in the Style selector.
- The `element` attribute specifies the HTML element that surrounds the text.
- Each `Attribute` child element defines the `name` and `value` of an attribute of the HTML tag.
  For example, the following definition creates a style that makes the selected text bold and underlined:

```
<Style name="Custom Bold And Underline " element="span">
 <Attribute name="style" value="font-weight: bold; text-decoration: underline;"/>
</Style>
```

If you use a custom XML file, instead of fckstyles.xml, to define your styles, specify the filepath in the `stylesXML` att ribute.

Defining custom templates

The editor includes a set of basic templates that insert HTML formatting into the `textarea` control. For example, the `Image and Title` template places a placeholder for an image on the left of the area, and a title and text to the

© 2014 Adobe Systems Incorporated. All rights reserved.

right of the image. Then right-click the image area to specify the image source and other properties, and replace the placeholder title and text.
You create your own templates by creating entries in *cf_webRoot*/CFIDE/scripts/ajax/FCKEditor/fcktemplates.xml file. Each template XML entry has the following format:

```
<Template title="template title" image="template image">
  <Description>template description</Description>
  <Html>
   <![CDATA[
    HTML to insert in the text area when the user selects the template.
   ]]>
  </Html>
 </Template>
```

The template title, image, and description appear in the Templates dialog box that appears when the user clicks the template icon on the rich text editor toolbar.
The following example template defines a title followed by text:

```
<Template title="Title and Text" image="template1.gif">
 <Description>A Title followed by text.</Description>
 <Html>
  <![CDATA[
   <h3>Type the title here</h3>
   Type the text here
  ]]>
 </Html>
</Template>
```

The name "Title and Text" and the template1.gif image appear in the template selection dialog box.
If you use a custom XML file, instead of fcktemplates.xml, to define your templates, specify the file path in the `templatesXML` attribute.

Defining custom skins

To create a custom skin that you specify in the `skin` attribute, create a subdirectory of the *cf_webRoot*/CFIDE/scripts/ajax/FCKeditor/editor/skins directory. The name of this subdirectory is the name that you use to specify the skin in the `skin` attribute. The custom skin directory must contain an images subdirectory and have the following files:

- **fck_editor.css**: Defines the main interface, including the toolbar, its items (like, buttons, panels) and the context menu.
- **fck_dialog.css**: Defines the basic structure of dialog boxes (standard for all dialogs).
- **fck_strip.gif**: Defines the Default toolbar buttons and context menu icons. It is a vertical image that contains all icons placed one above the other. Each icon must correspond to a 16x16 pixels image. You can add custom images to this strip.
- **images/toolbar.buttonarrow.gif:** Defines the small arrow image used in the toolbar combos and panel buttons.
  Place all other images used by the skin (that are specified in the CSS files) in the images subfolder.
  The most common way of customizing the skin is to change the fck_editor.css and fck_dialog.css files. For information on the skin format and contents, see the comments in those files.

Using the datefield input control

© 2014 Adobe Systems Incorporated. All rights reserved.

The HTML `cfinput` control with a type value of `datefield` lets users select dates from a pop-up calendar or enter the dates directly in the input box. When you use the control, keep in mind the following considerations:

- To correctly display label text next to the control in both Internet Explorer and Firefox, surround the label text in a `<div style="float:left;">` tag and place three `<br>` tags between each line.
- Consider specifying an `overflow` attribute with a value of `visible` in the `cflayoutarea` tag so that if the pop-up calendar exceeds the layout area boundaries, it appears completely.
- If you use a `mask` attribute to control the date format, it does not prevent the user from entering dates that do not conform to the mask. The `mask` attribute determines the format for dates that users select in the pop-up calendar. Also, if the user types a date in the field and opens the pop-up calendar, the calendar displays the selected date only if the entered text follows the mask pattern. If you do not specify a `mask` attribute, the pop-up only matches the default matching pattern.
- If the user types a date with a month name or abbreviation in the control, instead of picking a date from the calendar, the selected date appears in the pop-up calendar only if both of the following conditions are true:
  - The month position and name format match the mask pattern.
  - The month name matches, case correct, the month names specified by the `monthNames` attribute, or, for an `mmm` mask, their three-letter abbreviations.
- If the date mask specifies yy for the years, the pop-up calendar uses dates in the range 1951-2050, so if the user enters 3/3/49 in the text field, the calendar displays March 3, 2049.
- If the user enters invalid numbers in a date, the pop-up calendar calculates a valid date that corresponds to the invalid input. For example, if the user enters 32/13/2007 for a calendar with a dd/mm/yyyy mask, the pop-up calendar displays 01/02/2008.
  The following example shows a simple tabbed layout where each tab contains a form with several datefield controls.:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<html>
<head>
</head>

<body>
<cflayout type="tab" tabheight="250px" style="width:400px;">
 <cflayoutarea title="test" overflow="visible">
  <br>
  <cfform name="mycfform1">
   <div style="float:left;">Date 1: </div>
   <cfinput type="datefield" name="mydate1"><br><br><br>
   <div style="float:left;">Date 2: </div>
   <cfinput type="datefield" name="mydate2" value="15/1/2007"><br><br><br>
   <div style="float:left;">Date 3: </div>
   <cfinput type="datefield" name="mydate3" required="yes"><br><br><br>
   <div style="float:left;">Date 4: </div>
   <cfinput type="datefield" name="mydate4" required="no"><br><br><br>
  </cfform>
 </cflayoutarea>
 <cflayoutarea title="Mask" overflow="visible">
  <cfform name="mycfform2">
   <br>
   <div style="float:left;">Date 1: </div>
   <cfinput type="datefield" name="mydate5" mask="dd/mm/yyyy">
    (dd/mm/yyyy)<br><br><br>
   <div style="float:left;">Date 2: </div>
   <cfinput type="datefield" name="mydate6" mask="mm/dd/yyyy">
    (mm/dd/yyyy)<br><br><br>
   <div style="float:left;">Date 3: </div>
   <cfinput type="datefield" name="mydate7" mask="d/m/yy">
    (d/m/yy)<br><br><br>
   <div style="float:left;">Date 4: </div>
   <cfinput type="datefield" name="mydate8" mask="m/d/yy">
    (m/d/yy)<br><br><br>
  </cfform>
 </cflayoutarea>
</cflayout>

</body>
</html>
```

Note

In Internet Explorer versions previous to IE 7, this example shows the calendars for the first three fields in a page behind the following input controls.

Using autosuggest text input fields

When you create a text input (`type="text"`) in HTML, use the `autosuggest` attribute to specify a static or dynamic source that provides field completion suggestions as the user types. Use the `autosuggestMinLength` att ribute to specify the number of characters the user must type before first displaying any suggestions.

© 2014 Adobe Systems Incorporated. All rights reserved.

Note

To place label text next to a `cfinput` control that uses an `autosuggest` attribute and have it display correctly in both Internet Explorer and Firefox, surround the label text in an HTML `div` ta g with a `style="float: left"` attribute. Also if you have multiple controls, and place them on separate lines, follow the input controls with three `<br>` tags, as in the following example. Otherwise, the label and control do not lay out properly.

```
<div style="float: left"> Name: </div>
<cfinput name="userName" type="text" autosuggest="Andrew, Jane, Robert">
<br><br><br>
```

The control suggests entries from a static list of values. To use a static suggestion list, specify the list entries in the `a utosuggest` attribute, and separate the entries by the character specified by the `delimiter` attribute (by default, a comma), as the following example shows:

```
<cfinput type="text"
 autosuggest="Alabama\Alaska\Arkansas\Arizona\Maryland\Minnesota\Missouri"
 name="city" delimiter="\">
```

In this example, if you type the character *a* (in uppercase or lowercase) in the `cfinput` control, the list of states that start with A appears in a drop-down list. You navigate to a selection by using the arrow keys, and press Enter to select the item.

You also have the control suggest values from a dynamically generated suggestion list. To use a dynamic list, specify a CFC function, JavaScript function, or URL in the `autosuggest` attribute. Use the `autosuggestBindDel ay` attribute to specify the minimum time between function invocations as the user types. This limits the number of requests that are sent to the server. If you use a dynamic list, the input field has an icon to its right that animates while suggestions are fetched.

When you use a bind expression you must include a {`cfautosuggestvalue`} bind parameter in the function call or URL parameters. This parameter binds to the user input in the input control and passes it to the function or page.

A CFC or JavaScript autosuggest function must return the suggestion values as a one-dimensional array or as a comma-delimited list.

The HTTP response body from a URL must consist only of the array or list of suggestion values in JSON format. In ColdFusion, you can use the `serializeJSON` function to convert an array to JSON format. If an array with the suggestions is named nodeArray, for example, the following line would specify the only output on a CFML page that is called by using a bind expression with a URL:

```
<cfoutput>#serializeJSON(nodeArray)#</cfoutput>
```

You do not have to limit the returned data to values that match the `cfautosuggestvalue` contents, because the client-side code displays only the values that match the user input. In fact, the called function or page does not even have to use the value of the `cfautosuggestvalue` parameter that you pass to it. However, use the parameter if

© 2014 Adobe Systems Incorporated. All rights reserved.

the returned data would otherwise be long.

The following example shows how to use a bind expression to populate autosuggest lists. The Last Name text box displays an autosuggest list with all last names in the database that match the keys typed in the box. The First Name text box uses binding to the Last Name text box to display only the first names that correspond to the last name and the text entered in the box. The database query limits the responses to only include results that match the autosuggest criteria, so the autosuggest list displays all the returned results, and the suggestions only match if the database entry has a case-correct match.

To test this example with the cfdocexamples database, type S in the first box and the autosuggest list shows Smith and Stewart. If you select Smith and enter A or J in the First Name box, you get a name suggestion.

The following example shows the application:

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
</head>
<body>

<cfform>
 Last Name:<br />
 <cfinput type="text" name="lastName"
  autosuggest="cfc:suggestcfc.getLNames({cfautosuggestvalue})"><br />
 <br />
 First Name:<br />
 <cfinput type="text" name="firstName"
  autosuggest="cfc:suggestcfc.getFNames({cfautosuggestvalue},{lastName})">
</cfform>
</body>
</html>
```

The following example shows the suggestcfc.cfc file:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cfcomponent>

 <cffunction name="getLNames" access="remote" returntype="array" output="false">
  <cfargument name="suggestvalue" required="true">
  <!--- The function must return suggestions as an array. --->
  <cfset var myarray = ArrayNew(1)>
  <!--- Get all unique last names that match the typed characters. --->
  <cfquery name="getDBNames" datasource="cfdocexamples">
  SELECT DISTINCT LASTNAME FROM Employees
  WHERE LASTNAME LIKE <cfqueryparam value="#suggestvalue#%"
   cfsqltype="cf_sql_varchar">
  </cfquery>
  <!--- Convert the query to an array. --->
  <cfloop query="getDBNames">
   <cfset arrayAppend(myarray, lastname)>
  </cfloop>
  <cfreturn myarray>
 </cffunction>

 <cffunction name="getFNames" access="remote" returntype="array"
   output="false">
  <cfargument name="suggestvalue" required="true">
  <cfargument name="lastName" required="true">
  <cfset var myarray = ArrayNew(1)>
  <cfquery name="getFirstNames" datasource="cfdocexamples">
  <!--- Get the first names that match the last name and the typed characters. --->
  SELECT FIRSTNAME FROM Employees
  WHERE LASTNAME = <cfqueryparam value="#lastName#"
   cfsqltype="cf_sql_varchar">
  AND FIRSTNAME LIKE <cfqueryparam value="#suggestvalue & '%'#"
   cfsqltype="cf_sql_varchar">
  </cfquery>
  <cfloop query="getFirstNames">
   <cfset arrayAppend(myarray, Firstname)>
  </cfloop>
  <cfreturn myarray>
 </cffunction>

 </cfcomponent>
```

Issue with numeric data

For CFC methods that returns numeric data with a leading zero, for example, zip code `02674`, the zero is interpreted by the bind expression as an octal number and its decimal equivalent (in this case `1468`) even if you set `returnformat="string"`. To resolve this issue, for URL binds or binds routed by way of a JavaScript function (for example, using cfajaxproxy), you can set `returnformat=plain` to retain the numeric value. Also, leading zeros are stripped from the suggestion list for autosuggest controls.

Using the cfslider tag

The `cfslider` tag places a slider control, for selecting a numeric value from a range, in a ColdFusion form. The slider can be used within a `cfform` tag for forms in HTML and applet format. The `cfslider` is not supported with Flash forms.
In HTML forms, you can specify maximum, minimum, and increment values, to help you quickly filter complex

© 2014 Adobe Systems Incorporated. All rights reserved.

results. The available slider controls are: Vertical, with controls that can be adjusted to the top or bottom; Horizontal, with controls that can be adjusted to the left or right; Tip, that displays the values as data tips; and Snapping, that moves in incremental values.

Manipulating slider at runtime

The following code illustrates how you can manipulate a slider at runtime:

```
<script language="javascript">
 //use Coldfusion AJAX functions
 var sliderChange = function(slider,value)
 {
  //get slider name
  slidername = slider.getId();
  //get slider value
  currValue = ColdFusion.Slider.getValue(slidername);
  //set a new slider value
  newValue = parseInt(currValue+10);
  ColdFusion.Slider.setValue(slidername,newValue);
  //hide slider
  if(confirm("Do you want to hide slider?"))
  {
   ColdFusion.Slider.hide(slidername);
  }
  //show slider
  if(confirm("Do you want to show slider?"))
  {
    ColdFusion.Slider.show(slidername);
  }
  //disable slider
  if(confirm("Do you disable the slider?"))
  {
   ColdFusion.Slider.disable(slidername);
  }
  //enable slider
  if(confirm("Do you enable the slider?"))
  {
   ColdFusion.Slider.enable(slidername);
  }
 }
 var sliderDrag = function(slider)
 {
  //get slider name
  slidername = slider.getId();
  document.getElementById('currentSliderValue').innerHTML =  "Current Slider value :
<font color='red'><strong>" + ColdFusion.Slider.getValue(slidername) +
"<strong></font>";
 }
</script>
<br>
<cfform name="frm1">
 <p>
 <span id="currentSliderValue">Current Slider Value: <font
color="red"><strong>50</strong></font></span><br>
 </p>
</cfform>
<p>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<br><b>Volume</b>:
<cfslider
 name="s"
 format="html"
 min=1
 max=100
 value="50"
 tip="yes"
 onChange="sliderChange"
 onDrag = "sliderDrag"
 vertical="no"
 width="200pt"
```

© 2014 Adobe Systems Incorporated. All rights reserved.

```
  >
</p>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

Using geographical maps

The `cfmap` tag lets you embed a geographical map within your ColdFusion page. The following are the supported map types:

- earth
- terrain
- satellite
- hybrid
- map (default)

Using the marker window

The marker window opens when you click the marker icon in the map. It is used to provide information pertaining to the locations in the map, for example address or latitude and longitude. The marker window can be populated with static or dynamic content.

Populating data using static content

To manually populate data in the marker window, specify the value in the `markerwindowcontent` attribute.

Populating dynamic data using bind expression

To dynamically populate data, use the `markerbind` attribute with a bind expression that calls a CFC, JavaScript function, or a URL. The bind expression uses bind parameters to specify dynamic information and the values of any other form field attributes.
Pass the bind parameters to the bind expression. If you omit any of the parameters in the function call or URL, you get an error. These parameters send information about the map and its state to the data provider function. The data for these parameters is provided automatically. You do not set any values manually.
Provide the data as provided in the following code:

```
<br>
<cfoutput>
<table>
 <tr>
  <td bgcolor='red'><h4><font color='white'>URL Bind Example</font></td>
 </tr>
</table>
Map Name: #cfmapname#<br>
Latitude, Longitude:
(#DecimalFormat(cfmaplatitude)#,#DecimalFormat(cfmaplongitude)#)<br>
Address: #cfmapaddress#<br>
</cfoutput>
```

The following table provides details of the parameters:

Parameter name	Description
`cfmapname`	The name of the map.
`cfmaplatitude`	The latitude value for the location, in degrees. This value is set as the center of the map.

© 2014 Adobe Systems Incorporated. All rights reserved.

`cfmaplongitude`	The longitude value for the location, in degrees. This value is set as the center of the map.
`cfmapaddress`	The address of the location, which is set as the center of the map.

The format of the returned data depends on how you get the data:

Bind type	Return value
CFC	A ColdFusion structure. ColdFusion automatically converts the structure for return to the caller. Alternatively, you can return a JSON representation of the structure.
URL	A JSON representation of a structure. No other body contents is allowed.
JavaScript	A JavaScript object.

Use the `showmarkerwindow` attribute to control the display of the window.
The following example uses a bind expression and a CFC to populate dynamic data using a CFC bind expression.
The CFML page contains the following:

```
<br>
<cfmap
  centeraddress="Hobart, Tasmania, Australia"
  name="map1"
  type="map"
  tip="Hobart, Tasmania, Australia"
  zoomControl="small3d"
  markerbind="cfc:maps.getMapData({cfmapname}, {cfmaplatitude}, {cfmaplongitude},
{cfmapaddress})"
  showmarkerwindow = true>
 <cfmapitem name="m1" address="Taj Mahal, Agra, India" tip="Taj Mahal, Agra, India">

 <cfmapitem name="m2" latitude="40.46" longitude="117.05" showmarkerwindow=true
tip="Great Wall of China, Bejing">
 <cfmapitem name="m3" address="Stonehenge, England" tip="Stonehenge, England"
showmarkerwindow = false>
</cfmap>
```

The map.cfc is as follows:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cfcomponent>
 <cffunction name="getMapData" access="remote">
  <cfargument name="cfmapname">
  <cfargument name="cfmaplatitude">
  <cfargument name="cfmaplongitude">
  <cfargument name="cfmapaddress">
  <cfsavecontent variable="markup">
   <br>
   <cfoutput>
   <table>
    <tr>
     <td bgcolor='red'><h4><font color='white'>CFC Bind Example</font></td>
    </tr>
   </table>
   Map Name: #cfmapname#<br>
   Latitude, Longitude:
(#DecimalFormat(cfmaplatitude)#,#DecimalFormat(cfmaplongitude)#)<br>
   Address: #cfmapaddress#<br>
   </cfoutput>
  </cfsavecontent>
  <cfreturn markup>
 </cffunction>
</cfcomponent>
```

The following example shows how to populate dynamic data using a JavaScript bind expression:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<script language="JavaScript">
 var getMapData = function(cfmapname, cfmaplatitude, cfmaplongitude, cfmapaddress){
  var msg = "";
  msg = msg + "Map Name: " + cfmapname + "<br>";
  msg = msg + "Latitude,longitude: " + "("  + cfmaplatitude + "," + cfmaplongitude +
")" + "<br>";
  msg = msg + "Address: " + cfmapaddress + "<br>";
  //alert(msg);
  return "<br><table><tr><td bgcolor='red'><h4><font color='white'>" + "Javascript
Bind Example" + "</font></td></tr></table><hr>" + msg;
 }
</script>
<cfmap
 centeraddress="Hobart, Tasmania, Australia"
 name="map1"
 type="map"
 tip="Hobart, Tasmania, Australia"
 zoomControl="small3d"
 markerbind="javascript:getMapData({cfmapname}, {cfmaplatitude}, {cfmaplongitude},
{cfmapaddress})"
 showmarkerwindow = true>
 <cfmapitem name="m1" address="Taj Mahal, Agra, India" tip="Taj Mahal, Agra, India">

 <cfmapitem name="m2" latitude="40.46" longitude="117.05" showmarkerwindow=true
tip="Great Wall of China, Bejing">
 <cfmapitem name="m3" address="Stonehenge, England" tip="Stonehenge, England"
showmarkerwindow = false>
</cfmap>
```

The following example shows how to populate dynamic data using a URL bind expression:

```
<cfmap
 centeraddress="Hobart, Tasmania, Australia"
 name="map1"
 type="map"
 tip="Hobart, Tasmania, Australia"
 zoomControl="small3d"
markerbind="url:mapdata.cfm?cfmapname={cfmapname}&cfmaplatitude={cfmaplatitude}&cfma
plongitude={cfmaplongitude}&cfmapaddress={cfmapaddress}"
 showmarkerwindow = true >
 <cfmapitem name="m1" address="Taj Mahal, Agra, India" tip="Taj Mahal, Agra, India">

 <cfmapitem name="m2" latitude="40.46" longitude="117.05" showmarkerwindow=true
tip="Great Wall of China, Bejing">
 <cfmapitem name="m3" address="Stonehenge, England" tip="Stonehenge, England"
showmarkerwindow = false>
</cfmap>
```

Specifying Google map key

The Google Maps API key is required to embed Google Maps in your web pages.
The following URL provides details of how to sign up for the Google Maps API key:

© 2014 Adobe Systems Incorporated. All rights reserved.

http://code.google.com/apis/maps/signup.html
Currently, ColdFusion supports only embedding of Google map. To generate a map, provide a valid Google map API key, and specify the latitude and longitude of the location, or the address of the location. The Google map API key can be specified in the following ways:

- Using the `cfajaximport` tag. You specify the map API key in the `params` attribute as follows:`<cfajaximport params="#{googlemapkey='Map API Key'}#"`
- Using Application.cfc as follows:`<cfset this.googlemapkey="Map API Key">`
- Using the Settings page in the ColdFusion Administrator. Specify the map API key in the Google Map Key field. You can also specify the map API key in runtime.cfc.

Styling markers

You can specify the following:

- Custom marker icon: Specify the path to the icon using the `markericon` attribute. Ensure that you specify an image of appropriate size.
- Marker icon color: Use the `markercolor` attribute. You can specify a color of your preference only for the default icon and not for others.
- Map title: Use the `title` attribute.

© 2014 Adobe Systems Incorporated. All rights reserved.

Using the cfprogressbar tag

The `cfprogressbar` tag has the following characteristics:

- Automatically runs the progress bar for a duration that you specify.
- Dynamically loads data using bind expressions
- Lets styling of the progress bar
- Uses callback and error handlers that give control to the users after the progress bar completes processing or if it encounters any exceptions.
- Lets programmatic control over progress bar using JavaScript APIs.

Progress bar modes

The progress bar supports the following two modes:

Dynamic mode

User specifies the bind expression to provide data for the progress bar to display. The `bind` attribute specifies a function that determines the indicator length.The following CFM code shows how to use a CFC bind expression:

```
<cfajaxproxy cfc="pbar" jsclassname="pbar">
<head>
 <script>
  var utils = new pbar();
  var count = 0;
  var init = function()
  {
   document.getElementById('cfpbLabel').style.display = 'block';
   ColdFusion.ProgressBar.show('pBar');
   ColdFusion.ProgressBar.start('pBar');
  }
  var hideProgessBar = function()
  {
   document.getElementById('cfpbLabel').style.display = 'none';
   ColdFusion.ProgressBar.hide('pBar');
   utils.resetStatus();
  }
 </script>
</head>
<cfform>
 <div id="cfpbLabel" style="display:none">
  Saving File:
 </div>
 <cfprogressbar
  name="pBar"
  autodisplay=false
  bind="cfc:pbar.getProgessData()"
  onComplete="hideProgessBar"
  width="400"  >
 <cfset ajaxOnLoad('init')>
</cfform>
```

The following pb.cfc has the function that returns data for the progressbar:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cfcomponent>
 <cffunction name="resetStatus" access="remote">
  <!---
  Clear count from session so that next time the progress bar runs from the start
time.
  --->
  <cfif session.count gte 10>
   <cfset structdelete(session,"count")>
  </cfif>
 </cffunction>
 <cffunction name="getProgessData" access="remote">
  <!--- use a count to track progress --->
  <cfif not isdefined('session.count')>
   <cfset session.count = 1>
  <cfelse>
   <cfset session.count = session.count + 1 >
  </cfif>
  <!--- struct with status and message components of the progressbar --->
  <cfset data = {status=session.count * 0.1,message=(session.count * 10)  & "%"}>
  <cfreturn data>
 </cffunction>
</cfcomponent>
```

The following CFM code explains how to use the URL bind expression:

```
<head>
 <script>
  var init = function()
  {
   document.getElementById('cfpbLabel').style.display = 'block';
   ColdFusion.ProgressBar.show('pBar');
   ColdFusion.ProgressBar.start('pBar');
  }
  var hideProgessBar = function()
  {
   document.getElementById('cfpbLabel').style.display = 'none';
   ColdFusion.ProgressBar.hide('pBar');
  }
 </script>
</head>
<cfform>
 <div id="cfpbLabel" style="display:none">
  Saving File:
 </div>
 <cfprogressbar
  name="pBar"
  autodisplay=false
  bind="url:progressdata.cfm"
  onComplete="hideProgessBar"
  width="400"  >
 <cfset ajaxOnLoad('init')>
</cfform>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

The following is the Progressdata.cfm:

```
<!--- use a count to indicate progress --->
<cfif not isdefined('session.count')>
 <cfset session.count = 1>
<cfelse>
 <cfset session.count = session.count + 1 >
</cfif>
<!--- the struct to be sent back; using the populate the status and message
components of the progressbar --->
<cfset data = {status=session.count * 0.1,message=(session.count * 10)  & "%"}>
<!--- clear count from session to start afresh the next time the program is run --->

<cfif session.count eq 10>
 <cfset structdelete(session,"count")>
</cfif>
<!--- data sent back via URL binds must use SerializeJSON() --->
<cfoutput>#SerializeJSON(data)#</cfoutput>
```

The following CFM code has the JavaScript bind expression:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<head>
 <script>
  var count = 0;
  var init = function()
  {
   document.getElementById('cfpbLabel').style.display = 'block';
   ColdFusion.ProgressBar.show('pBar');
   ColdFusion.ProgressBar.start('pBar');
  }
  var hideProgessBar = function()
  {
   document.getElementById('cfpbLabel').style.display = 'none';
   ColdFusion.ProgressBar.hide('pBar');
  }
  var getProgessData = function()
  {
   count++;
   if(count > 10)
    return {STATUS:1,MESSAGE:"Done"}
   else
    return {STATUS:count*0.1,MESSAGE:(count * 10) + "%"}
  }
 </script>
</head>
<cfform>
 <div id="cfpbLabel" style="display:none">
  Saving File:
 </div>
 <cfprogressbar
  name="pBar"
  autodisplay=false
  bind="javascript:getProgessData()"
  onComplete="hideProgessBar"
  width="400"
  >
 <cfset ajaxOnLoad('init')>
</cfform>
```

Manual mode

In the manual mode, you specify the duration the progress bar takes to complete the display of progress. In the following example, `autodisplay` is set to `false` as a result of which the progress bar is not shown when the page is first loaded. When the page is loaded, `init` function is invoked and the function displays and runs the progress bar. The default interval used in this mode is one second.

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<head>
 <script>
  var init = function()
  {
   document.getElementById('cfpbLabel').style.display = 'block';
   ColdFusion.ProgressBar.show('pBar');
   ColdFusion.ProgressBar.start('pBar');
  }
  var hideProgessBar = function(){
   document.getElementById('cfpbLabel').style.display = 'none';
   ColdFusion.ProgressBar.hide('pBar');
  }
 </script>
</head>
<cfform>
 <div id="cfpbLabel" style="display:none">
  Saving File:
 </div>
 <cfprogressbar
  name="pBar"
  duration="10000"
  autodisplay=false
  onComplete="hideProgessBar"
  width="400"
 />
 <cfset ajaxOnLoad('init')>
</cfform>
```

Working with a progress bar at runtime

This section illustrates how to use the JavaScript API to update the progress bar status. The following CFM code loads a progress bar using the JavaScript API ColdFusion.ProgressBar.updatestatus.
On program load, `intit` function displays the progress bar and calls the getProgressData JavaScript function to manually update the progress bar. The getProgressData function assigns the status and message variables passed to the JavaScript API update status.
While working with a progress bar at runtime, ensure that you specify a dummy duration (for instance, `duration=`*`5000`*). Even though the custom JavaScript function decides the actual duration, `duration` is a mandatory attribute.

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cfajaxproxy cfc="pbar" jsclassname="pbar">
<head>
 <script>
  var utils = new pbar();
  var init = function()
  {
   document.getElementById('cfpbLabel').style.display = 'block';
   ColdFusion.ProgressBar.show('pBar');
   getProgessData();
  }
  var hideProgessBar = function()
  {
   document.getElementById('cfpbLabel').style.display = 'none';
   ColdFusion.ProgressBar.hide('pBar');
  }
  var getProgessData = function()
  {
   for(i=1;i <= 10;i++)
   {
    var status = parseFloat(i * 0.10);
    var message = Math.round(status * 100) + "%";
    ColdFusion.ProgressBar.updateStatus('pBar',status,message);
    utils.sleep(1000);
   }
   hideProgessBar();
  }
 </script>
</head>
<cfform>
 <div id="cfpbLabel" style="display:none">
  Saving File:
 </div>
 <cfprogressbar
  name="pBar"
  autodisplay=false
  duration=15000
  onComplete="hideProgessBar"
  width="400"  >
 <cfset ajaxOnLoad('init')>
</cfform>
```

The `sleep` function in the following CFC provides sleep functionality in the JavaScript code:

```
<cfcomponent>
<cffunction name="sleep" access="remote">
 <cfargument name="timetosleep" default="1000">
 <cfset sleep(timetosleep)>
</cffunction>
</cfcomponent>
```

Styling the progress bar

© 2014 Adobe Systems Incorporated. All rights reserved.

The `cfprogressbar` has `style` attribute that lets you decide:

- Background color of the progress bar
- Color of the progress message
- Color of the progress indicator
  The following code illustrates styling: `style="bgcolor:ADD8E6;progresscolor:6183A6;textcolor:191970"`

© 2014 Adobe Systems Incorporated. All rights reserved.

Using the cfmessagebox tag

The `cfmessagebox` tag lets you define controls for displaying pop-up messages. In addition to the standard alert box, this tag lets you include a prompt and entry field in the box.

Using confirm dialog

The following code shows how to build a confirm message box with two buttons YES and NO.

```
<cfmessagebox
 type="confirm"
 name="msgbox1"
 title="Confirm Dialog"
 message="Do you want proceed?"
 buttonType="YesNo"
 icon="info"
 labelYes="Click Yes to continue"
 labelNo="No"
 x=100
 y=200>
<!--- This example illustrates usage of the Confirm dialog in "YesNoCancel" mode
--->
<cfmessagebox
 type="confirm"
 name="msgbox2"
 title="Save File"
 message="Do you want to save the file?"
 buttonType="YesNoCancel"
 icon="question"
 labelYes="Click Yes to save the file"
 labelNo="No"
 labelCancel="Quit"
 width="400"
 x=500
 y=200>
<br><br>
<input
 type="button"
 name="confirm1"
 onClick="javascript:ColdFusion.MessageBox.show('msgbox1');"
 value="YesNo Confirm"
 >
<input
 type="button"
 name="confirm2"
 onClick="javascript:ColdFusion.MessageBox.show('msgbox2');"
 value="YesNoCancel Confirm"
 >
```

Styling a message box

The `bodyStyle` attribute, a CSS style specification for the body of the message box helps you to style the message. As a general rule, use this attribute to set color and font styles.
The following example illustrates the usage of the `bodyStyle` attribute:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cfmessagebox
 type="alert"
 name="msgbox1"
 title="Download Status"
 message="File Download Complete"
 icon="info"
 width="400"
 bodyStyle="background-color:white;color:blue"
 x=300
 y=200>
<br><br>
<input
 type="button"
 name="alert"
 onClick="javascript:ColdFusion.MessageBox.show('msgbox1');"
 value="Alert MessageBox"
 >
```

#back to top

© 2014 Adobe Systems Incorporated. All rights reserved.

Using Ajax Data and Development Features

Adobe ColdFusion supports Ajax features to use data dynamically in web pages.
For information on ColdFusion Ajax user interface capabilities, see Using Ajax User Interface Components and Features.

© 2014 Adobe Systems Incorporated. All rights reserved.

About ColdFusion Ajax data and development features

Ajax (Asynchronous JavaScript and XML) is a set of web technologies for creating interactive web applications. Ajax applications typically combine:

- HTML and CSS for formatting and displaying information.
- JavaScript for client-side dynamic scripting
- Asynchronous communication with a server using the XMLHttpRequest function.
- XML or JSON (JavaScript Object Notation) as a technique for serializing and transferring data between the sever and the client.
  ColdFusion provides many tools that simplify using Ajax technologies for dynamic applications. By using ColdFusion tags and functions, you can easily create complex Ajax applications.

ColdFusion Ajax features

ColdFusion provides data management and development, and user interface Ajax features.

Data and development features

ColdFusion data and development features help you develop effective Ajax applications that use ColdFusion to provide dynamic data. They include many features that you can use with other Ajax frameworks, including Spry.

- ColdFusion supports data binding in many tags. Binding allows an application that uses form and display tags, such as `cfselect` and `cfwindow`, to dynamically display information based on form input. In the simplest application, you display form data directly in other form fields, but usually you pass form field data as parameters to CFC or JavaScript functions or URLs and use the results to control the display. Data binding uses events to automatically update the display, typically when the bound input data changes. You can also use the ColdFusion.Ajax.submitForm JavaScript function to get the current value of any bindable element.
- The `cfajaxproxy` tag creates a JavaScript proxy that represents a CFC on the server. It manages the communication between the client and server, and provides several functions to simplify and manage handling the communication and its results. This tag provides access to all remote functions in a CFC. It also lets applications, including applications that use Ajax frameworks or widget sets such as Dojo or Backbase, easily access data from ColdFusion servers.
- The `cfsprydataset` tag lets you use bind expressions to dynamically create and update Adobe Spry data sets. Applications that use Spry framework elements, such as dynamic regions, use this tag to populate the Spry elements with information based on ColdFusion control input. This feature lets you easily intermix Spry and ColdFusion controls.
- The `cfajaximport` tag specifies the location of the JavaScript and CSS files that a ColdFusion page imports. You also use this tag to selectively import files required by specific Ajax-based tags and functions. The ability to change the file location lets you support a wide range of configurations and use advanced techniques, such as application-specific styles. Although ColdFusion automatically determines and imports the required files, sometimes you must manually specify the information.
- ColdFusion provides several CFML functions that let you create and consume JSON format data on the server and let you prepare data for use in HTML `cfgrid` tags.
- You display a floating logging window that shows client-side logging and debugging information. ColdFusion Ajax features display information and error messages in this window, and several logging tags let you display additional information, including the structure of complex JavaScript variables.

User interface features

- Ajax-based HTML controls including the following:
  - Tree
  - Grid
  - Rich text editor
  - Date field

© 2014 Adobe Systems Incorporated. All rights reserved.

    - Autosuggest text input
- Pop-up menus and menu bars.
- Container tags that provide bordered, box, and tabbed layouts, pop-up windows, and pod regions.
- A `cfdiv` container tag that enables asynchronous form submission and binding in HTML `div` and other regions.
- Tool tips for specific controls and HTML regions.
  For detailed information on using the user interface features, see Using Ajax User Interface Components and Features.

ColdFusion Ajax tags

The following table lists ColdFusion Ajax-related tags and functions, including all tags that support Ajax-based features. It does not include subtags that are used only in the bodies of the listed tags:

Data tags	UI tags	UI tags	Functions
cfajaximport	cfdiv	cfselect	AjaxLink
cfajaxproxy	cfgrid	cftextarea	AjaxOnLoad
cfsprydataset	cfinput	cftree	DeserializeJSON
	cflayout	cftooltip	IsJSON
	cfmenu	cfwindow	QueryConvertForGrid
	cfpod		SerializeJSON

© 2014 Adobe Systems Incorporated. All rights reserved.

Binding data to form fields

Many ColdFusion Ajax features use *binding* to provide dynamic interactivity based on user input or changing data. When you use binding, a *bind expression* gets evaluated, and the display gets updated based on new data each time a specific event (`onChange` by default) occurs on a form control field specified by a *bind parameter*. This way, the value of the tag that specifies the bind expression, and the display, get updated dynamically based on changing information, including user-entered form data. When you use binding the page contents updates, but the entire page is not refreshed.

> **Note**
>
> When a bound window is not visible, or a tab is not selected, its contents is not updated when the controls it is bound to change. When the tab or window is made visible, it is updated only if events have been received from the bound controls while the control was not visible.

Depending on the specific ColdFusion tag, a bind expression uses bind parameter values directly or passes bind parameter values as parameters to a CFC function, a JavaScript function, or an HTTP request and uses the function or request response to update the page. Use the following as the data source for a bind expression:

- ColdFusion form control attributes and values. You can bind to the following controls:
  - cfgrid
  - cfinput with checkbox, datefield, file, hidden, radio, or text types
  - cfselect
  - cftextarea
  - cftree
- Spry data set elements

> **Note**
>
> You cannot use a bind expression to bind to controls in a dynamically loaded region. For example, you cannot bind from a control on one page to a control in a layout area on that page if the `cflayoutarea` tag uses a `source` attribute for its contents. However, a dynamically loaded region binds to controls on the page that loads it, so the file specified by the `source` attribute uses bind expressions that specify controls on the page that contains the `cflayoutarea` tag.

The results of the bind expression determine the value of the tag that uses the expression. For example, if you specify a URL in a bind expression as the `source` attribute of a `cfwindow` control, the page specified by the URL must return the full contents of the window.
For more examples, see Using Ajax User Interface Components and Features and the reference pages for controls that support binding.

Using bind expressions

To specify a `bind` expression, use one of the following formats:

- cfc:*componentPath.functionName(parameters*)

> **Note**
>
> In ColdFusion 9 (and ColdFusion 9 only, this note does not apply to ColdFusion 10!), the component path cannot use a mapping. The `componentPath` value must be a dot-delimited path from the web root or the directory that contains the current page.

© 2014 Adobe Systems Incorporated. All rights reserved.

- javascript:*functionName(parameters*)
- url:*URL?parameters*
- *URL?parameters*
- A string containing one or more instances of {*bind parameter*},
  such as {firstname}.{lastname}@{domain}
  In formats 1-4 the parameters normally include one or more bind parameters. The following table lists the tag attributes that support bind expressions and the formats each use:

Attribute	Tags	Supported formats
`autosuggest`	`cfinput type="text"`	1, 2, 3
`bind`	cfdiv, cfinput, cftextarea	1, 2, 3, 5
`bind`	cfajaxproxy, cfgrid, cfselect, cfspryd ataset, cftreeitem	1, 2, 3
`onChange`	cfgrid	1, 2, 3
`source`	cflayoutarea, cfpod, cfwindow	4

The following examples show some of these uses:

```
bind="cfc:myapp.bookorder.getChoices({book})"
source="/myApp/innerSource/cityWindow.cfm?cityname={inputForm:city}
```

In these examples, {`book`} and {`inputForm:city`} specify
bind parameters that dynamically get data from the book and city
controls, and the city control is in the inputForm form.
If a bind attribute specifies a page that defines JavaScript functions, the function definitions on that page must have the following format:

```
functionName = function(arguments) {function body}
```

Function definitions that use the following format may not work:

```
function functionName (arguments) {function body}
```

However, Adobe recommends that you include all custom JavaScript in external JavaScript files and import them on the application's main page, and not write them inline in code that you get using the source attribute. Imported pages do not have this function definition format restriction.

Specifying bind parameters

A bind parameter specifies a form control value or other attribute, as in the following example:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
bind="cfc:myapplication.bookSearch.getStores({form1:bookTitle})"
```

In this example, the bind parameter is form1:bookTitle and specifies the `value` attribute of the bookTitle field of the `form1` form.
Bind parameters have either of the following formats:

```
{[formName:]controlName[.attributeName][@event]}
{SpryDataSetName.fieldName}
```

The brackets ([]) indicate optional contents and are not part of the parameter.

Note

To include a literal brace character in a bind expression, escape the character with a backslash, as {, }.

The formname value

The *formname* entry identifies the form that contains the control you are binding to. Specify a form name if multiple forms contain bind targets with the same names. To specify the form name, start the bind expression with the form's `id` attribute the `name` attribute if you did not specify an `id` attribute, and follow it with a colon (:). To specify the `book` control that is in a form named `inputForm`, for example, use the following format:

```
bind="cfc:myapp.bookorder.getChoices({inputForm:book})"
```

The controlName value

To bind to a form field, the *controlName* value must be the value of the `id` or `name` attribute of the form control to which you are binding. If a control has both an `id` and a `name` attribute, use either value. You can bind to any ColdFusion form control, including `cfgrid` and `cftree`. You cannot bind to values in other ColdFusion tags, such as `cftable`.To bind to a Spry data set, specify the data set name in this part of the bind parameter.You can bind to multiple radio buttons or check boxes by giving them the same `name` value. If all the radio buttons in a radio button group have the same `name` value, the bind parameter represents the selected button. If multiple check boxes have the same `name` value, the bind parameter represents the values of the selected controls in a comma-delimited list. If you also specify a unique `id` attribute for each check box or radio button, specify an HTML `label` tag for each button or check box and use the `id` value in the `for` attribute; in this case, users select items by clicking the label, not just the button or box.If a `cfselect` control supports multiple selections, the bind expression returns the information about the selected items in a comma-delimited list.You can bind only to controls that are available in the DOM tree when the bind is registered. Binds are registered when the page with the bind expression loads, either in the browser window or in a container tag. As a result, if you have two `cfdiv`, `cflayoutarea`, `cfpod`, or `cfwindow` containers that you load by using a `source` (or for `cfdiv` tag, `bind`) attribute, you cannot bind controls in one container to controls in the other, because one container cannot be assured that the other is loaded when it loads. Similarly, elements on the main page cannot bind to elements on a dynamically loaded container. To prevent this problem, define the bind target in line on the main page, instead of using a `source` or `bind` attribute to retrieve the

© 2014 Adobe Systems Incorporated. All rights reserved.

markup that contains the bind target. In other words, the "master" form with fields that serve as sources of bind expressions is loaded statically (on the main page), and the "child" controls that depend on the data are loaded dynamically, on a page that is specified in a `source` or `bind` attribute.

The attributeName value

When you bind to a form control, by default, the bind expression represents the `value` attribute of the specified control. If the bind target is a `cfselect` tag, the bind expression represents a comma delimited list of the values of the selected items.To bind to a different attribute, follow the control `name` or `id` with a period (.) and the attribute name. To pass the `checked` attribute of a checkbox `cfinput` tag as a CFC parameter, for example, use an expression such as the following:

```
bind="cfc:myapp.bookorder.useStatus({myForm:approved.checked@click})
```

Note

You can bind to the display text of a select box, instead of the value, by specifying an attribute name of `innerHTML`.

Note

When you bind to a check box, use the `@click` event specifier to ensure that the bind expression is triggered in Internet Explorer when the user selects or deselects the check box, not when the box loses focus.

Grids and trees do not have default bind attributes.

- Always specify a grid target attribute
  by using the format {*gridID.columnName*}. The bind expression gets the value of the specified column in the selected row.
- For trees, you must bind to a specific node in the tree. Specify the node by using the node ID or an explicit path to the node.
  To bind to a Spry data set element or attribute, use standard Spry path notation. For example, specify an element name.

The event value

By default, the bind expression function executes each time the control specified in the bind parameter has an `onChange` event. To trigger updates on a different JavaScript event, end the bind expression with an at sign (@) and the event name, without the "on" prefix. The following code, for example, executes the getChoices CFC each time the user presses the mouse button while the pointer is over the `book` control:

```
bind="cfc:myapp.bookorder.getChoices({inputForm:book@mousedown})"
```

© 2014 Adobe Systems Incorporated. All rights reserved.

> **Note**
>
> To bind to a `cfinput` control with `type` attribute of `button`, specify a bind event setting, such as `click`. The `change` event is the default event has no effect.

When you bind to a Spry data set, do not specify an event. The expression is evaluated when the selected row changes in the data set, or when the data set reloads with new data.You can also specify that a specific bind parameter never triggers bind expression reevaluation, by specifying `@none` as the event. This is useful, for example, if a bind expression uses multiple bind parameters binding to different form fields, and you want the bind expression to trigger changes only when one of the fields changes, not when the others change. In this case, you would specify `@none` for the remaining fields, so events from those fields would not trigger the bind. The following code shows this use:

```
bind="cfc:books.getinfo({iForm:book}, {iForm:author@none})"
```

The `@none` event specifier is also useful when used with autosuggest text inputs, trees and grids, as follows:

- When you use an autosuggest text input, the bind expression is evaluated as a user types in text, and picks up data from all bind parameters, including those parameters with `@none` specified. Therefore, for autosuggest, specify `@none` for all bind parameters, because there is no way for it to react to changes in the parameters.
- When you call the `ColdFusion.Grid.refresh` or `ColdFusion.Tree.refresh` function, the function fetches data from all bind parameters when it evaluates the bind expression, including any parameters with `@none` specified. If you specify `@none` for all bind parameters, the tree or grid might not respond to changes in other controls, but gets data from all the bind parameters each time you explicitly refresh it.

Using CFC functions in bind expressions

As with JavaScript functions, you pass arguments to a CFC function specified in a bind expression positionally. When you do this, the argument names in a CFC function definition do not have to be the same as the bind parameter names, but the arguments in the bind expression must be in the same order as the arguments in the CFC function definition.
Alternatively, you pass named CFC function arguments. Then, the bind expression and CFC function must use the same names for the arguments, and the function does not have to define the arguments in the same order as they are passed. To specify argument names in a bind expression, use a format such as the following, which uses two named parameters, arg1 and arg2:

```
bind="cfc:mycfc.myfunction(arg1={myform:myfield1},arg2={myform:myfield2})"
```

Support for CFCs outside webroot

> **Note**
>
> To use this feature, you must install ColdFusion 9 Update 1.

Components outside the webroot can be accessed in bind expressions. This implies that tags such as `cfajaxproxy` or Ajax components such as grid, map, or progress bar can be used in more effective ways.

© 2014 Adobe Systems Incorporated. All rights reserved.

> **Note**
>
> In the previous releases, the CFCs had to be web-accessible for Ajax applications to function.

In addition to accessing CFCs using relative or absolute path, you can also use any of the following methods to access CFCs:

- logical mappings (defined in the ColdFusion Administrator)
- per-app mappings (defined in Application.cfc)
- imports (using `cfimport`/import)

Usage

The following code shows the usage of this enhancement using per-map mappings:
Application.cfc
```
THIS.mappings["/mycfc"] = "C:\www\shared\components";
```
Test.cfm
```
<cfajaxproxy cfc="mycfc.utils" jsclassname='jsobjname' />
```

Example

In this example, a per-app mapping named `mycfcs` has been created in Application.cfc pointing to "c:\components". For the sample code to work, create a folder named components in your system root (in this example, c:) and copy the Employee.cfc to that folder.Application.cfc

```
<cfcomponent>
 <cfset this.name = "cfcoutsidewebroot">
 <cfset this.sessionmanagement = true>
 <Cfset mappingname = "/mycfcs">
 <Cfset mappingpath = "c:\components\">
 <cfset this.mappings[mappingname] = mappingpath>
</cfcomponent>
```

Employee.cfc

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cfcomponent>
 <cfscript>
  remote any function
getEmployees(page,pageSize,gridsortcolumn="EMP_ID",gridsortdirection="ASC"){
       var startRow = (page-1)*pageSize;
       var endRow = page*pageSize;

       if(!isdefined("arguments.gridsortcolumn") or
isdefined("arguments.gridsortcolumn") and trim(arguments.gridsortcolumn) eq "")
        gridsortcolumn = "EMP_ID";
       if(!isdefined("arguments.gridsortdirection") or
isdefined("arguments.gridsortdirection") and arguments.gridsortdirection eq "")
        gridsortdirection = "ASC";
       var mysql = "SELECT Emp_ID, FirstName,  EMail, Department FROM Employees";
       if(isdefined("arguments.gridsortcolumn") and arguments.gridsortcolumn neq "")

        mysql = mysql & " ORDER BY " & gridsortcolumn;
       if(isdefined("arguments.gridsortdirection") and arguments.gridsortdirection
neq "")
        mysql = mysql & " " & gridsortdirection ;
       rs1 = new query(name="team", datasource="cfdocexamples",
sql=mysql).execute();
       return QueryConvertForGrid(rs1.getResult(), page, pageSize);
  }


  remote any function editEmployees(gridaction,gridrow,gridchanged){
    writelog("edit employee info");
  }

 </cfscript>
</cfcomponent>
```

Employee.cfm

```
<cfform>
 <cfgrid
   format="html"
   name="grid01"
   pagesize=10
   title="Employee database"

bind="cfc:mycfcs.employee.getEmployees({cfgridpage},{cfgridpagesize},{cfgridsortcolu
mn},{cfgridsortdirection})"
   onChange="cfc:mycfcs.
employee.editEmployees({cfgridaction},{cfgridrow},{cfgridchanged})">
   <cfgridcolumn name="Emp_ID" display=false header="ID" />
   <cfgridcolumn name="FirstName" display=true header="First Name"/>
   <cfgridcolumn name="Email" display=true header="Email"/>
   <cfgridcolumn name="Department" display=true header="Department" />
 </cfgrid>
</cfform>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

Using binding in control attributes

When you use direct binding you specify a bind expression in a ColdFusion form or display control attribute. In the simplest, form of binding you use form fields, such as a name field, to fill other fields, such as an e-mail field, as the following example. shows. When you enter a name or domain and tab to click in another field, the name is added to the e-mail field.

```
<html>
<head>
</head>
<body>

<cfform name="mycfform">
 First Name: <cfinput type="text" name="firstname" value=""><br>
 Last Name: <cfinput type="text" name="lastname" value=""><br>
 Domain: <cfinput type="text" name="domain" value=""><br>
 E-mail: <cfinput type="text" name="email1" size="30"
  bind="{firstname}.{lastname}@{domain}">
</cfform>
</body>
</html>
```

The following example shows the results of binding to radio buttons and check boxes with the same `name` attribute but different `id` attributes. Notice that because each control has a separate `id` value that is used in the `label` tags, you click the labels to select and deselect the controls.

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<html>
<head>
</head>
<body>

<cfform name="myform">
 Pick one:
 <cfinput id="pickers1" name="pickone" type="radio" value="Apples">
  <label for="pickers1">Apples</label>
 <cfinput id="pickers2" name="pickone" type="radio" value="Oranges">
  <label for="pickers2">Oranges</label>
 <cfinput id="pickers3" name="pickone" type="radio" value="Mangoes">
  <label for="pickers3">Mangoes</label>
 <br>
 <cfinput name="pickone-selected" bind="{pickone}"><br />
 <br />

 Pick as many as you like:
 <cfinput id="pickers4" name="pickmany" type="checkbox" value="Apples">
  <label for="pickers4">Apples</label>
 <cfinput id="pickers5" name="pickmany" type="checkbox" value="Oranges">
  <label for="pickers5">Oranges</label>
 <cfinput id="pickers6" name="pickmany" type="checkbox" value="Mangoes">
  <label for="pickers6">Mangoes</label>
 <br/>
 <cfinput name="pickmany-selected" bind="{pickmany}"><br />

</cfform>
</body>
</html>
```

Most applications call a CFC function, or JavaScript function, or use a URL to make an HTTP request (typically to a CFML page), and pass bind parameters as the function or URL parameters.
The following example uses the same form as the first example in the preceding section, but uses a different bind expression with the following features:

- It uses the `keyup` events of the name and domain fields to trigger binding. So the e-mail field gets updated each time that you enter a letter in any of these fields.
- It calls a CFC, which uses only the first letter of the first name when forming the e-mail address, and forces the domain name to be all lowercase.
  The following example shows the bindapp.cfm page:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<html>
<head>
</head>
<body>
<cfform name="mycfform">
 First Name: <cfinput type="text" name="firstname" value=""><br>
 Last Name: <cfinput type="text" name="lastname" value=""><br>
 Domain: <cfinput type="text" name="domain" value=""><br>
 E-mail: <cfinput type="text" name="email"
  bind="cfc:bindFcns.getEmailId({firstname@keyup},{lastname@keyup},
  {domain@keyup})">
</cfform>
</body>
</html>
```

The following example shows the bindFcns.cfc CFC file:

```
<cfcomponent>
 <cffunction name="getEmailId" access="remote">
  <cfargument name="firstname">
  <cfargument name="lastname">
  <cfargument name="domain">
  <cfreturn
"#left(arguments.firstname,1)#.#arguments.lastname#@#lcase(arguments.domain)#">
 </cffunction>
</cfcomponent>
```

Many of the examples in the documentation for ColdFusion Ajax features use binding, including more complex forms of binding.

Using the cfajaxproxy tag to bind to display controls

The `cfajaxproxy` tag with a `bind` attribute makes any of the following elements dependent on one or more bound ColdFusion Ajax controls:

- A single CFC function
- A single JavaScript function
- An HTTP request; for example, the URL of a CFML page
  The function or request executes whenever a specific event (by default, the `onChange` event) of the bound control occurs.

> **Note**
>
> if you specify a bind attribute with a URL, the HTTP request includes a *CF*`NODEBUG` URL parameter. ColdFusion checks this value, and when it is `true`, does not append to the response any debugging information that it normally would send. This behavior ensures that JSON responses to Ajax requests do not include any non-JSON (that is, debugging information) text.

The `cfajaxproxy` tag includes the following attributes that determine how the proxy handles the data returned by the function or the page:

© 2014 Adobe Systems Incorporated. All rights reserved.

- The `onError` function specifies code to handle an HTTP error return. You use this attribute with a URL or CFC bind.
- The `onSuccess` function handles a valid return from the function or page and updates the display as required with the returned information.
  Binding a function or request by using the `cfajaxproxy` tag enables you to perform a server-side action, such as updating a database by using bind parameter values based on a user action in some control, and then run a specific action or set of actions in one or more controls based on the server response. Because it uses an `onSuccess` function to process the return from the server, this form of binding provides substantially more flexibility than a CFML control `bind` parameter. This format also lets you use a control `bind` parameter for one kind of action, and the `cfajaxproxy` tag for a different activity.
  For example, if you have a form with an editable `cfgrid` control and a delete button that a user clicks to delete a grid row. The application must have the following behaviors:

- When the user clicks the delete button two things must happen:
  - The application must call a mycfc.deleteButton CFC function to delete the row from the database.
  - The grid must update to remove the deleted row.
- When the user edits the grid content, the grid must call a `mycfc.update` function to update the database.
  Implement these behaviors by doing the following:

- In the `cfgrid` tag, specify a `bind` attribute that uses a bind expression to call a mycfc.update function each time the user changes the grid contents.
- In a `cfajaxproxy` tag, specify a `bind` attribute that calls the mycfc.deleterow CFC function, and specify an `onSuccess` attribute that calls the `ColdFusion.Grid.refresh` function to update the displayed grid when the CFC function returns successfully.
  The following code snippets show how you could do this:

```
<cfajaxproxybind="cfc:mycfc.deleteRow({deletebutton@click},
  {mygrid.id@none}"onSuccess="ColdFusion.Grid.refresh(mygrid, true)">

...

<cfinput type="button" name="deletebutton">
<cfgrid name="mygrid" bind="mycfc.update({cfgridpage}, {cfgridpagesize},
  {cfgridsortcolumn}, {cfgridsortdirection})>
```

The following complete example shows a simple use of the `bind` attribute in a `cfajaxproxy` tag. For the sake of brevity, the bind expression calls a JavaScript function; as a result, the `cfajaxproxy` tag cannot use a `onError` at tribute.

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<html>
<head>
<script language="javascript">
 function test(x,y){
  return "Hello, " + x + "!";
 }
 function callbackHandler(result){
  alert("Bind expression evaluated. Result: \n" + result);
 }
</script>

<cfajaxproxy bind="javascript:test({input1@none},{button1@click})"
  onSuccess="callbackHandler">
</head>

<body>
<cfform name="mycfform">
  <cfinput type="text" value="" name="input1" size="30">
  <cfinput type="button" name="button1" value="Submit">
</cfform>
</body>
</html>
```

Getting bindable attribute values in JavaScript

Use the `ColdFusion.Ajax.submitForm` function in your JavaScript code to get the current value of any attribute of a bindable control. This technique is useful for getting values for complex controls such as `cfgrid` and `cftree`. For more information, see the ColdFusion.Ajax.submitForm function in the *CFML Reference*.

© 2014 Adobe Systems Incorporated. All rights reserved.

Managing the client-server interaction

Manage the client-server interaction in several ways:

- Use the `cfajaxproxy` tag to create a client-side JavaScript proxy for a CFC and its functions. You then call the proxy functions in client JavaScript code to access the server-side CFC functions.
- Use the `cfsprydataset` tag to dynamically populate a Spry data set from a URL or a CFC. You then use the data set to populate Spry dynamic regions. You also use Spry data sets in bind expressions.
- Use the `cfajaxproxy` tag to bind fields of ColdFusion Ajax form controls as parameters to a specific CFC function, JavaScript function, or HTTP request, and specify JavaScript functions to handle successful or error results. The function is run each time the event determined by the bind expression occurs.
- Use ColdFusion Ajax-based UI tags, such as `cftree` or `cfgrid` that automatically get data from CFCs or URLs by using data binding.
  For Information on working with Spry, including how to use the `cfsprydataset` tag, see Using Spry with ColdFusion. For detailed information on using binding, including how to use binding with ColdFusion UI tags and the `cfajaxproxy` tag, see Binding data to form fields. For more information on using the ColdFusion Ajax-based UI tags, see Using Ajax User Interface Components and Features.

Using ColdFusion Ajax CFC proxies

Use the `cfajaxproxy` tag to create a client-side JavaScript proxy for a CFC and its functions. The proxy object has the following characteristics:

- It provides a JavaScript function that corresponds to each CFC remote function. Calling these functions in your client-side JavaScript code remotely calls the CFC functions on the server.
- It provides JavaScript support functions for controlling the communication, which specifies asynchronous result and error handler callbacks, and sends form data to the server. For detailed information on these functions, see the cfajaxproxy tag in the *CFML Reference*.
- It manages the interactions between the client and the CFC, including serializing and deserializing JavaScript arrays and structures to and from JSON format for transmission over the web.
- It ensures automatic serialization (into JSON format) and deserialization of CFC return values.
  By using a ColdFusion Ajax proxy, any JavaScript code can call the proxied CFC functions. Thus, any Ajax application, not just one that uses ColdFusion Ajax UI elements, can use dynamic data provided by CFCs. Also, the proxy provides access to all of the functions in a CFC, not just the single function that you specify in a bind expression.

Creating a JavaScript CFC proxy

The `cfajaxproxy` tag with a `cfc` attribute generates a JavaScript proxy that represents a CFC on the web client. Because a ColdFusion page that uses the `cfajaxproxy` tag is used as an Ajax client web page, the page typically starts with the `cfajaxproxy` tag (or tags), and the remainder of the page consists of the HTML and JavaScript required to control the display and perform the page logic on the client.

> **Note**
>
> Because JavaScript is case-sensitive, make sure that you match the case of the keys in any ColdFusion structure or scope that you send to the client. By default, ColdFusion sets variable names and structure element names to all-uppercase. (You create structure element names with lowercase characters by specifying the names in associative array notation, for example, `myStructure["myElement"]="value"`.) The keys for the two arrays in the JSON object that the ColdFusion `SerializeJSON` function generates to represent a query are `COLUMNS` and `DATA`, for example, not `columns` and `data`.

For more information about creating and using CFC proxies, see the cfajaxproxy tag in the *CFML Reference*.

© 2014 Adobe Systems Incorporated. All rights reserved.

Configuring the CFC proxy

The proxy provides several JavaScript functions that you use to control the behavior of the proxy:

- You use the `setAsyncMode` and `setSyncMode` functions to control the call mode. By default, all calls to remote CFC functions are asynchronous, the most common synchronization method for Ajax applications.
- You use the `setCallbackHandler` and `setErrorHandler`functions to specify the functions that handle the results of successful and unsuccessful asynchronous calls.

> **Note**
>
> For error handling to work properly, select the Enable HTTP Status Codes option on the Server Settings > Settings page of the ColdFusion Administrator.

- You use the `setHTTPMethod` function to control whether the call uses a GET HTTP request (the default) or a POST request.
- You use the `setForm` function to prepare the proxy to send full form data to the remote function. This function causes the proxy to pass each form field as a separate parameter to the CFC function.
- You use the `setReturnFormat` function to specify whether to return the result in JSON format (the default), in WDDX format, or as plain text. Use the `setQueryFormat` function to specify whether to return a JSON format query as an object with an array of column names and an array of row arrays, or as an object that corresponds to the WDDX query format. These functions only effect the format of data returned by ColdFusion. Data sent from the proxy to the server is always in JSON format.

Submitting data to a CFC

When you use an Ajax CFC proxy, you send to the CFC function any client-side data that can be serialized to JSON format, not just form data. However, the proxy cannot serialize DOM tree elements because they are wrappers on native code. Therefore, you cannot use DOM tree elements directly as parameters to a CFC function that you call by using an Ajax proxy. To ensure correct serialization to JSON for sending to the CFC, use basic JavaScript types only: array, object, and simple types. Instead of using a DOM element directly, you pass only the specific element attributes that you require to the CFC function, either individually or in an array or object.
When you use the `cfc` attribute, you submit form data to the CFC without refreshing the client page by calling the proxy `setForm` function before you call a CFC proxy function in your JavaScript. The proxy function then passes all field values of the specified form to the CFC function. In the CFC function Arguments scope, the argument names are the form control `ID` attributes (or, by default, the `name` attributes) and the argument values are the control values.

> **Note**
>
> You cannot use the `setForm` function to submit the contents of file fields.

To pass the form parameters to your proxy function, invoke the proxy function immediately after you call the `setFor m` function. Subsequent proxy function invocations do not get the form parameters.
If you also pass arguments explicitly to the CFC, `cfargument` tags in the CFC function that specify the explicitly passed arguments must precede any `cfargument` tags for the form fields. For example, if you have the following submitForm JavaScript function:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
function submitForm() {
 var proxy = new remoteHandler();
 proxy.setCallbackHandler(callbackHandler);
 proxy.setErrorHandler(errorHandler);
 proxy.setForm('myform');
 proxy.setData('loggedIn');
}
```

In this example, the remoteHandler.cfc setData function starts as follows:

```
<cffunction name="setData" access="remote" output="false">
  <cfargument name="loggedIn">
  <cfargument name="userName">
 ...
```

In this example, `userName` is the name of a form field. If the `cfargument` tag for `userName` preceded the `cfargument` tag for the `loggedIn` explicitly passed variable, the CFC function would not get the value of `loggedIn`. Your CFC function can omit `cfargument` tags for the form fields.

Example: Using an asynchronous CFC proxy

The following example uses a remote CFC method to populate a drop-down list of employees. When you select a name from the list, it uses a call to the CFC method to get information about the employee, and displays the results. The main application page has the following lines:

```
<!--- The cfajaxproxy tag creates a client-side proxy for the emp CFC.
  View the generated page source to see the resulting JavaScript.
  The emp CFC must be in the components subdirectory of the directory
  that contains this page. --->
<cfajaxproxy cfc="components.emp" jsclassname="emp">

<html>
 <head>
  <script type="text/javascript">

   // Function to find the index in an array of the first entry
   // with a specific value.
   // It is used to get the index of a column in the column list.
   Array.prototype.findIdx = function(value){
    for (var i=0; i < this.length; i++) {
     if (this[i] == value) {
      return i;
     }
    }
   }

   // Use an asynchronous call to get the employees for the
   // drop-down employee list from the ColdFusion server.
   var getEmployees = function(){
    // Create an instance of the proxy.
```

© 2014 Adobe Systems Incorporated. All rights reserved.

```
 var e = new emp();
 // If you set a callback handler for the proxy, the proxy's calls
 // are asynchronous.
 e.setCallbackHandler(populateEmployees);
 e.setErrorHandler(myErrorHandler);
// The proxy getEmployees function represents the CFC
// getEmployees function.
 e.getEmployees();
}

// Callback function to handle the results returned by the
// getEmployees function and populate the drop-down list.
var populateEmployees = function(res)
{
 with(document.simpleAJAX){
  var option = new Option();
  option.text='Select Employee';
  option.value='0';
  employee.options[0] = option;
  for(i=0;i<res.DATA.length;i++){
   var option = new Option();
   option.text=res.DATA[i][res.COLUMNS.findIdx('FIRSTNAME')]
    + ' ' + res.DATA[i][[res.COLUMNS.findIdx('LASTNAME')]];
   option.value=res.DATA[i][res.COLUMNS.findIdx('EMP_ID')];
   employee.options[i+1] = option;
  }
 }
}

// Use an asynchronous call to get the employee details.
// The function is called when the user selects an employee.
var getEmployeeDetails = function(id){
 var e = new emp();
 e.setCallbackHandler(populateEmployeeDetails);
 e.setErrorHandler(myErrorHandler);
// This time, pass the employee name to the getEmployees CFC
// function.
 e.getEmployees(id);
}
// Callback function to display the results of the getEmployeeDetails
// function.
var populateEmployeeDetails = function(employee)
{
 var eId = employee.DATA[0][0];
 var efname = employee.DATA[0][1];
 var elname = employee.DATA[0][2];
 var eemail = employee.DATA[0][3];
 var ephone = employee.DATA[0][4];
 var edepartment = employee.DATA[0][5];

 with(document.simpleAJAX){
  empData.innerHTML =
  '<span style="width:100px">Employee Id:</span>'
  + '<font color="green"><span align="left">'
  + eId + '</font></span><br>'
  + '<span style="width:100px">First Name:</span>'
  + '<font color="green"><span align="left">'
  + efname + '</font></span><br>'
  + '<span style="width:100px">Last Name:</span>'
```

© 2014 Adobe Systems Incorporated. All rights reserved.

```
      + '<font color="green"><span align="left">'
      + elname + '</font></span><br>'
      + '<span style="width:100px">Email:</span>'
      + '<font color="green"><span align="left">'
      + eemail + '</span></font><br>'
      + '<span style="width:100px">Phone:</span>'
      + '<font color="green"><span align="left">'
      + ephone + '</font></span><br>'
      + '<span style="width:100px">Department:</span>'
      + '<font color="green"><span align="left">'
      + edepartment + '</font></span>';
     }
    }

    // Error handler for the asynchronous functions.
    var myErrorHandler = function(statusCode, statusMsg)
    {
     alert('Status: ' + statusCode + ', ' + statusMsg);
    }

   </script>
  </head>

  <body>
   <!--- The form to display the employee drop-down list and
   employee data. --->
   <form name="simpleAJAX" method="get">
    List of Employees:   
    <select name="employee" onChange="getEmployeeDetails(this.value)">
     <script language="javascript">
      getEmployees();
     </script>
    </select>
    <br><br>
    <span id="empData"></span>
   </form>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

```
 </body>
</html>
```

The following component, which gets the data from the data source, must be in a file named emp.cfc in the components subdirectory of the application directory. The CFC uses the cfdocexamples data source that is installed with ColdFusion if you install the documentation.

```
<cfcomponent>
 <cfset this.dsn = "cfdocexamples">
 <cffunction name="getEmployees" access="remote" returnFormat="json" output="false">

  <cfargument name="empid" required="no" type="string" default="0">
  <Cfquery name="qryEmp" datasource="#this.dsn#">
   select * from Employees
   <cfif empid neq 0>
   where Emp_ID = #empid#
   </cfif>
  </Cfquery>
  <cfreturn qryEmp>
 </cffunction>
</cfcomponent>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

Using Spry with ColdFusion

ColdFusion provides support for mixing native ColdFusion elements and Spry elements in a single application.

- ColdFusion tags use Spry data sets directly in bind expressions. Therefore, a ColdFusion form element, such as `cfinput`, binds to a field in a dynamic Spry data set, and is updated each time the data set updates, including when the user selects an item in a Spry control or dynamic region that the data set populates. To bind to a Spry data set, specify the data set name followed by the path to the specific element that you bind to, by using standard Spry path syntax. For example, if dsFilters is a Spry data set with a name column, the `{dsFilters.name}` bind parameter binds to the value of the current row's name column. The bind parameter *cannot*specify an event; the bind expression is re-evaluated each time the selected row in the data set changes. The following example shows the bind syntax:

```
<cfinput name="Input1" type="text"
bind="CfC:DataManager.getInData(filter={dsFilters.name})
```

- Spry data sets use a CFC function as the data source. To do this, you simply specify the URL of the CFC in the Spry.Data.XMLDataSet function, just as you would invoke any remote CFC method using a URL. Specify the method name with a method URL parameter, and pass data to the function in additional URL parameters, as in the following example:

```
Spry.Data.XMLDataSet("MyAppMgr.cfc?method=getFilter&filter=scores",
"filters/filter");
```

- The cfsprydataset tag dynamically creates and updates Spry XML or JSON data sets based on ColdFusion form data. Spry dynamic regions and other elements then use this data to control their display. The following example shows a `cfsprydataset` tag that creates a Spry XML data set named dsProducts by calling the getData.getProductDetails function and passing it the value of the selected name in a `cfgrid` control. The data set updates each time the `name`value changes.

```
<cfsprydataset
 name="dsProducts"
 type="xml"
 bind="CFC:getData.getProductDetails(prodname={myform:mygrid.name})"
 xpath="products/product"
 options="{method: 'POST'}"
 onBindError="errorHandler">
```

ColdFusion includes the complete Spry 1.5 framework release in *web_root/_CFIDE/scripts/ajax/spry directory. For more information about Spry framework, see www.adobe.com/go/learn_spry_framework_en. For more information, see the cfsprydataset tag in the _CFML Reference*.

Spry data set example

This example has the following behavior:

1. It uses a CFC function directly to populate a Spry XML data set, from an XML file.

© 2014 Adobe Systems Incorporated. All rights reserved.

2. It displays information from the Spry data in a Spry dynamic region list box.
3. It uses the selected item in the Spry data set to control the contents of a `cfgrid` control. The `cfgrid` bind expression calls a CFC and passes it a parameter bound to the selected item in the Spry XML data set.
4. It creates a second Spry XML data set by using a `cfsprydataset` tag that binds to the selected item in the `cfgrid` control and calls a CFC function.
5. It displays information from the second Spry data set in a second Spry dynamic region.
   The example lets a user select the genre of books to display: all books, fiction, or nonfiction from a Spry list box populated from the XML file. The selected genre determines the information displayed by a `cfgrid` contr ol, and a text input control shows the selected genre. The selected item in the `cfgrid` control determines the information that is displayed in a second Spry dynamic region.
   The application consists of the following files:

- A roundtrip.cfm page with the display controls and related logic
- A GridDataManager.cfc file with two functions:
  - A getFilter function that gets the XML for the spry data set
  - A getData function that gets the contents of the cfgrid control
  - A getProduct function that gets detailed information on the selected book
- A Filters.xml file with the XML data for the spry data set
  For this example to display images, create an images subdirectory of your application directory that contains images with the names specified by the BOOKIMAGE column of the cfbookclub database BOOKS table.

The roundtrip.cfm page

```
<html xmlns="http://www.w3.org/1999/xhtml" xmlns:spry="http://ns.adobe.com/spry">

<head>
<!--- The screen.css style sheet is provided in the Spry distribution. --->
<link href="screen.css" rel="stylesheet" type="text/css" media="all"/>
<!--- Include the XPath and Spry JavaScript files. --->
<script type="text/javascript"
 src="/CFIDE/scripts/ajax/spry/includes/xpath.js"></script>
<script type="text/javascript"
 src="/CFIDE/scripts/ajax/spry/includes/SpryData.js"></script>

<!--- Create the dsFilters Spry XML data set used to populate the FiltersList
dynamic region
 that lists the filters. Call the GridDataManager CFC getFilter method directly from
a
 Spry XMLDataSet function because no binding is needed. --->
<script>
 var dsFilters = new
  Spry.Data.XMLDataSet("GridDataManager.cfc?method=getFilter", "filters/filter");
</script>

<!--- Use a cfsprydataset tag with binding to generate a dsProduct Spry data set
with details
 about the book grid selection. --->
<cfsprydataset
 name="dsProduct"
 type="xml"
 bind="CFC:GridDataManager.getProductDetails(prodname={bookform:bookgrid.TITLE})"
 xpath="products/product"
 options="{method: 'POST'}"
 onBindError="errorHandler">

<!--- Function to handle bind errors. --->
```

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<script language="javascript">
 errorHandler = function(code,msg){
  alert("Error w/bind occurred. See details below:\n\n" + "Error Code: "
   + code + "\n" + "Error Message: " + msg);
 }
</script>

<!--- Specify the size of the FiltersList Spry dynamic region.
   By default it would be unnecessarily large. --->
<style type="text/css">
<!--
#FiltersList {
 height:100px;
 width: 150px;
}
-->
</style>
</head>

<body>
<!--- A Spry dynamic region containing repeated ListBoxItem controls.
 Each item specifies a filter to use in filling the book list grid.
 The items are populated by the data from the CFC getFilter method. --->
<div id="FiltersList" spry:region="dsFilters" class="SpryHiddenRegion">
 <div spry:repeat="dsFilters" class="ListBoxItemGroup">
  <div class="ListBoxItem"
   onclick="dsFilters.setCurrentRow('{dsFilters::ds_RowID}');"
   spry:selectgroup="feedsList" spry:select="SelectedListBoxItem"
   spry:hover="ListBoxItemHover">
    {dsFilters::description}
  </div>
 </div>
</div>

<!--- A ColdFusion form with the book list data grid. --->
<cfform name="bookform">
<!--- Create a book list grid.
  Users select the book for which to get details from this grid.
  Populate it with the results of the CFC getData method.
  Pass the method the value of the name field of the selected
  item in the dsfilters Spry dynamic region. --->
 <cfgrid name="bookgrid"
   format="html"
   bind="CfC:GridDataManager.getData(page={cfgridpage},
    pageSize={cfgridpagesize},sortCol={cfgridsortcolumn},
    sortDir={cfgridsortdirection},filter={dsFilters.name})"
   selectMode="browse"
   width=400
   delete="true"
   pageSize=7>
  <cfgridcolumn name="TITLE" header="Name" width=200>
  <cfgridcolumn name="GENRE" header="Type" width=200>
 </cfgrid><br />
 <!--- Show the value of the name field of the selected item in the Spry dynamic
region.
 --->
 <cfinput name="filter" bind="{dsFilters.name}">
</cfform>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<hr>

<!--- A Spry dynamic region that uses the dsProduct data set to display information
on the
 selected product. --->
<div id="RSSResultsList" spry:detailregion="dsProduct" class="SpryHiddenRegion">
 <strong>{name}</strong><br>
 <img src="images/{bookimage}" alt="product box shot" width="238" height="130"/>
 <div>{desc}</div>
</div>

<hr>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

```
</body>
</html>
```

The gridDataManager.cfc file

```
<cfcomponent name="GridDataManager">

 <!--- The getFilter function gets the filter XML to populate the dsFilters Spry
data set.
  It specifies returnFormat=plain to send XML text. --->
 <cffunction name="getFilter" access="remote" output="false" returnFormat="plain">
  <cffile action="read" file="#ExpandPath('.')#\Filters.xml" variable="filtersxml">
  <cfcontent type="text/xml" reset="yes">
  <cfreturn filtersxml>
 </cffunction>

 <!--- The getData function returns books that match the specified genre, or all
books if
  there is no genre. --->
 <cffunction name="getData" access="remote" output="false">
  <cfargument name="page" required="yes">
   <cfargument name="pageSize" required="yes">
  <cfargument name="sortCol" required="yes">
   <cfargument name="sortDir" required="yes">
  <cfargument name="filter" required="no">
  <cfquery name="books" datasource="cfbookclub">
   select TITLE, GENRE from BOOKS
   <cfif isDefined("arguments.filter") AND arguments.filter NEQ "">
    where GENRE = '#arguments.filter#'
   </cfif>
   <cfif arguments.sortCol NEQ "" AND arguments.sortDir NEQ "">
     order by #arguments.sortCol# #arguments.sortDir#
   <cfelse>
    order by TITLE ASC
   </cfif>
  </cfquery>
  <!--- Return the data only for the current page. --->
  <cfreturn QueryConvertForGrid(books, arguments.page,
   arguments.pageSize)>
 </cffunction>

 <!--- The getProductDetails gets data for a single book and converts it to XML for
use
  in the dsProduct Spry data set. --->
 <cffunction name="getProductDetails" access="remote" output="false">
  <cfargument name="prodname" default="The Road">
  <!--- Get the information about the book from the database. --->
  <cfquery name="bookDetails" datasource="cfbookclub">
   select TITLE, GENRE, BOOKIMAGE, BOOKDESCRIPTION from BOOKS
    where TITLE = '#arguments.prodname#'
  </cfquery>
  <!--- Convert the query results to XML. --->
  <cfoutput>
   <cfxml variable="BookDetailsXML" >
    <?xml version="1.0" encoding="iso-8859-1"?>
    <products>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

```
      <product>
       <name>#BookDetails.TITLE#</name>
       <category>#BookDetails.GENRE#</category>
       <bookimage>#BookDetails.BOOKIMAGE#</bookimage>
       <desc>#BookDetails.BOOKDESCRIPTION#</desc>
      </product>
     </products>
    </cfxml>
   </cfoutput>
   <!--- Convert the XML object to an XML string. --->
   <cfset xmldata = xmlparse(BookDetailsXML)>
   <cfcontent type="text/xml" reset="yes">
   <cfreturn xmldata>
  </cffunction>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

```
</cfcomponent>
```

The Filters.xml file

```
<?xml version="1.0" encoding="iso-8859-1"?>
<filters>

 <filter>
  <filterid>1</filterid>
  <name></name>
  <description>No Filter</description>
 </filter>

 <filter>
  <filterid>2</filterid>
  <name>Fiction</name>
  <description>Look for Fiction</description>
 </filter>

 <filter>
  <filterid>3</filterid>
  <name>Non-fiction</name>
  <description>Look for Nonfiction</description>
 </filter>

</filters>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

Specifying client-side support files

By default, ColdFusion does the following:

- Gets all the client-side JavaScript, CSS, and other files required for Ajax-based features from the *web_root*/CFIDE/scripts/ajax directory.
- For each application page, imports only the JavaScript files required for the tags that are explicitly included on the page.
  In some cases, override these default behaviors.

Specifying a custom script or CSS location

In some situations, you cannot use the default location for the CFIDE directory, because a hosting site blocks access to it to prevent access to the ColdFusion Administrator. Then, move the CFIDE/scripts directory, or the subdirectories that you use in your applications, to a different location.
In other situations, you have custom versions of some of the client-side files, such as the CSS files that specify form control appearance, that apply only to certain applications.
In both situations, inform ColdFusion of the new location. Specify the location of either or both directories containing the following files:

- All client-side resources required by the ColdFusion Ajax features
- Only the CSS files required by the ColdFusion Ajax features

Specify the client-side resource location

Use any of the following techniques to control the location of the directory that contains the client-side resources required by the ColdFusion Ajax features:

- If the ColdFusion client-side files required by all applications, including the files used by `cfform` tags are in a single location, you specify the directory in the ColdFusion Administrator > Server Settings > Settings page, Default CFFORM ScriptSrc Directory field. The directory you specify and its subdirectories must have the same structure and contents as the CFIDE/scripts directory tree.
- If the client-side files required for Ajax features on a specific page are in one location, you use the `cfajaximport` tag `scriptsrc` attribute to specify the source directory. This tag overrides the setting in the administrator, and does not affect the files used for standard `cfform` features. The directory you specify must have an ajax subdirectory with the same structure and contents as the CFIDE/scripts/ajax directory tree.
- You specify the client-side source directory for a specific form in the `cfform` tag `scriptsrc` attribute. This setting overrides any `cfajaximport` tag setting for the form and its child controls. The directory you specify and its subdirectories must have the same structure and contents as the CFIDE/scripts directory tree.
  If you require multiple resource locations for a single page, make sure that each JavaScript file is imported only once on a page, the first time it is required. Therefore, you cannot use different copies of one JavaScript file on the same page.
  To prevent problems, ColdFusion generates an error if you specify more than one `scriptsrc` attribute on a page. Therefore, if multiple forms require custom client-side resource files, specify their location in a single `cfajaximport` tag, not in `scriptsrc` attributes in the `cfform` tags.

Specify the CSS file location

Use the `cfajaximport` tag `cssSrc` attribute to specify the location of a directory that contains only the CSS files that control the style of ColdFusion Ajax-based controls. This attribute overrides any `scriptsrc` value in determining the CSS file location. Therefore, you could use the CSS files in the `scriptsrc` directory tree for most pages, and specify a `cssSrc` attribute on selected application pages that require a custom look.
For detailed information on how to use the `scriptsrc` and `cssSrc` attributes, and requirements for the contents of the specified directory, see the cfajaximport tag in the *CFML Reference*.

© 2014 Adobe Systems Incorporated. All rights reserved.

Importing tag-specific JavaScript files

In the following situations, ColdFusion does *not* automatically import the JavaScript files that are required for Ajax-based tags:

- If you use a ColdFusion Ajax-based tag on a page that you specify by using a `source` or `bind` attribute in a container tag, such as `cfdiv`, `cflayoutarea`, `cfpod`, or `cfwindow`. Place a `cfajaximport` tag on the page that has the container tag and use the `tags` attribute to specify the Ajax feature tags that are on the other pages. (You do not have to do this for any tags that are also used on the page with the `source` attribut e.)
- If you use a ColdFusion Ajax JavaScript function, such as ColdFusion.Window.create or ColdFusion.navigate , on a page that does not otherwise import the required ColdFusion Ajax JavaScript functions, use the `cfajaximport` tag to import the required JavaScript functions. If you are using a function, such as `coldFuson.navigate`, that is not used for a specific control, you can omit any attributes; the default behavior is to import the base functions that are not control-specific. If you are using a function such as `ColdFusion.Window.create`, use the `tags` attribute and identify the associated control, for example, `cfwindow`in the following line:

```
<cfajaximport tags="cfwindow">
```

For detailed information on importing tag-specific JavaScript files, see the cfajaximport tag in the *CFML Reference*.

© 2014 Adobe Systems Incorporated. All rights reserved.

Using data interchange formats

All complex data that is communicated over an HTTP connection must be serialized into a string representation that can be transmitted over the web. Most commonly, web client applications use XML or JSON.
As a general rule, ColdFusion automatically handles all necessary serialization and deserialization when you use ColdFusion Ajax features. The proxies that you create with the `cfajaxproxy` tag, and the bind expressions that call CFC functions automatically request data in JSON format, and automatically deserialize JSON data to JavaScript variables.
ColdFusion also provides the capability to create, convert, and manage data in web interchange formats. This is helpful, for example, if you use custom Ajax elements to get data from ColdFusion servers.
Also, you use ColdFusion data serialization capability for any applications that create or consume complex data transmitted over an HTTP connection. For example, if you want to make a web service or feed available in JSON format, many Yahoo! web services currently are accessible by using simple URLS that return data as JSON.

> **Note**
>
> For information on ColdFusion tags and functions for handling XML or WDDX data, see Using XML and WDDX.

Controlling CFC remote return value data format

By default, CFC functions convert data that they return to remote callers to WDDX format. However, they can also return the data in JSON format, or as plain string data. (XML objects are automatically converted to string representation when returning plain data.)
ColdFusion Ajax elements that request data from CFC functions, including bind expressions and the function proxies generated by the `cfajaxproxy` tag, automatically generate a `returnFormat` parameter in the HTTP URL to request JSON data from the CFC function.
Control the CFC function return format in the following ways:

- Use the `returnFormat` attribute on the `cffunction` tag.
- Set a `returnFormat` parameter in the HTTP request that calls the CFC function.
- Use the CFC proxy `setReturnFormat` function. (You do this only if your client-side code requires non-JSON format data, for example, XML or WDDX.)
  If the requested return format is JSON and the function returns a query, ColdFusion serializes the query into a JSON object in either of the following formats:
- As a JSON object with two entries: an array of column names, and an array of column data arrays.These entries are returned in the following situations:
  - By default
  - If you specify an HTTP URL parameter of `queryFormat="row"`
  - If you use the `cfajaxproxy` tag and call the proxy object's `setReturnFormat` function with a parameter value of `row`
    ColdFusion client-side binding and proxy code automatically converts this data into JavaScript that is consumed directly by HTML grids.
- As a JSON object with three entries: the number of rows, an array of column names, and an object where each key is a column name and each value is an array with the column dataThese entries are returned in the following situations:
  - If you specify an HTTP URL parameter of `queryFormat="column"`
- If you use the `cfajaxproxy` tag and call the proxy object's `setQueryFormat` function with a parameter value of `column` ColdFusion client-side binding and proxy code *does not* convert column format data into JavaScript that is consumed directly by HTML grids. However, use this format with the `cfajaxproxy` tag, because you can refer to the returned data by using the column names directly. For example, if a CFC function returns a query with user data, you get the user names in your JavaScript by specifying values such

© 2014 Adobe Systems Incorporated. All rights reserved.

as userData.firstName0 and userData.lastName0.
For more information, see the SerializeJSON function in the *CFML Reference*.

Using JSON

JSON (JavaScript Object Notation) is a lightweight JavaScript-based data interchange format for transmission between computer systems. It is a much simpler format than XML or WDDX, and is an efficient, compact format for transmitting data required for Ajax applications. ColdFusion Ajax bind expressions that use CFCs tell the CFC function to send the data in JSON format by including a `returnformat="json"` parameter in the HTTP request, and automatically handle the JSON-formatted result.
JSON represents objects by using { *key* : *value* , *key* : *value...* } notation, and represents arrays in standard [_ value_ , *value*...] notation. Values can be strings, numbers, objects, arrays, true, false, or null. Therefore, you can nest arrays and objects inside each other. For a detailed specification of the JSON format, see www.JSON.org.
Although ColdFusion Ajax-based controls and the `cffunction` tag interoperate transparently, without you converting anything to JSON format, other applications can take advantage of JSON format data. Many public feeds are now available in JSON format. For example, the Yahoo! search interface returns a JSON data set, del.icio.us provides JSON feeds showing your posts and tags, and Blogger feeds are available in JSON format. You don't have to use Ajax to display these feeds; use standard ColdFusion tags and functions to display the results.
The following CFML functions support using JSON format in server-side code:

- `DeserializeJSON`
- `SerializeJSON`
- `IsJSON`

  For more information about these functions and examples, see the *CFML Reference*.
  The following example shows how to use ColdFusion JSON functions in a non-Ajax application. It does a Yahoo search for references to "ColdFusion Ajax" and displays these results:

- The total number of web pages found
- The titles and summaries of the (by default 10) returned results. The title is a link to the web pageURL.

```
<!--- Send an http request to the Yahoo Web Search Service. --->
<cfhttp
 url='http://api.search.yahoo.com/WebSearchService/V1/webSearch?appid=YahooDemo&quer
y=
 "ColdFusion Ajax"&output=json'>

<!--- The result is a JSON-formatted string that represents a structure.
  Convert it to a ColdFusion structure. --->
<cfset myJSON=DeserializeJSON(#cfhttp.FileContent#)>

<!--- Display the results. --->
<cfoutput>
 <h1>Results of search for "ColdFusion 9"</h1>
 <p>There were #myJSON.ResultSet.totalResultsAvailable# Entries.<br>
 Here are the first #myJSON.ResultSet.totalResultsReturned#.</p>
 <cfloop index="i" from="1" to="#myJSON.ResultSet.totalResultsReturned#">
  <h3><a href="#myJSON.ResultSet.Result[i].URL#">
   #myJSON.ResultSet.Result[i].Title#</a></h3>
  #myJSON.ResultSet.Result[i].Summary#
 </cfloop>
</cfoutput>
```

Serialization of numeric values

© 2014 Adobe Systems Incorporated. All rights reserved.

> **Note**
>
> To use this feature, you must install ColdFusion 9 Update 1.

In the previous releases (including ColdFusion 9), serializing an integer using serializeJSON, converts the number to a double. For example, `SerializeJSON (123)` returns `123.0`.
In ColdFusion 9.0.1, the integer is retained in its original format but enclosed within quotes. That is, `SerializeJSON (123)` returns `"123"`. This is applicable only to positive integers.
The following table gives more examples:

Input	Serialized JSON
10	"10"
012	"012"
10.25	"10.25"
10.25E5	"1025000.0"
10.25E-5	"1.025E-4"
-10	"-10.0"
-10.25	"-10.25"

> **Note**
>
> To remove the quotes in the returned value, for positive integers and integers within quotes, set the jvm argument `json.numberasdouble to true` . However, negative integers such as -10 will still be serialized to "-10.0" even if `json.numberasdouble=true`.

Ajax plumbing enhancement in ColdFusion 9.0.1

ORM CFCs support the attribute `remotingfetch` for a property.
By default, `remotingfetch` is set to `false`.
While serializing ORM CFCs, ColdFusion introspects the `remotingfetch` property and if it is `false`, does not return any relationship information.
If it is set to `true`, relationship information is shown. If circular reference is detected, only one level of relationship is shown.

© 2014 Adobe Systems Incorporated. All rights reserved.

Debugging Ajax applications

ColdFusion provides a set of JavaScript functions that log information to a pop-up display window. ColdFusion also logs many standard client-side activities to the window.

Display logging information

To display the logging window you must do the following:

1. Enable ColdFusion to send information to the logging window
2. Request logging window information in the main CFML page request.

Enable logging output

To enable ColdFusion to send information to the logging window, do the following:

- Select the Enable Ajax Debug Log Window option on the ColdFusion Administrator > Debugging & Logging > Debug Output Settings page. To view exception messages in the logging window, select the Enable Robust Exception Information option on the Debug Output Settings page.
- Make sure that the IP address of the system where you do the debugging is included on the ColdFusion Administrator > Debugging & Logging > Debugging IP List page of the ColdFusion Administrator. By default this list includes only 127.0.0.1.

Display logging information for a page

To display the logging window when you request a CFML page in the browser, specify an HTTP parameter of `cfdebug` in the URL when you request a page, as in the following URL:

```
http://localhost:8500/myStore/products.cfm?cfdebug
```

After the debug log window appears, it continues running until you navigate to a new page in the browser. The logging window includes options that let you filter the messages by either or both of the following criteria:

- Severity
- Category
  You can select to display logging information at any combination of four levels of severity: debug, info, error, and window. The specific logging function that you call determines the severity level.
  The logging window always displays options to filter the output by using standard categories: bind, global, http, LogReader, and widget. (For information on these categories, see *Standard ColdFusion logging messages* below.) It also displays a filter option for each custom category that you specify in a ColdFusion logging call. ColdFusion does not limit the number of categories you create, but create only as many categories as you require to debug your application effectively.

Logging information

You call the following JavaScript functions to send information to the logger. In most cases, the function corresponds to a severity level, as follows:

Function	Severity	Purpose
ColdFusion.Log.debug	debug	A message that aids in debugging problems.

© 2014 Adobe Systems Incorporated. All rights reserved.

ColdFusion.Log.dump	debug	A representation of a single variable in a format similar to `cfdump`. This function displays the structure and contents of JavaScript Array and Object variables.
ColdFusion.Log.error	error	Information about an error. Use this function only in error-handling code.
ColdFusion.Log.info	info	Information about properly operating code that is useful in tracing and analyzing the client-side code's execution.

You cannot generate a window-level message. This level is reserved for messages generated by the log reader window, including information about JavaScript errors in the log function calls.
When you call a logging function, you specify a message and a category.

- The message can include JavaScript variables and HTML markup, such as bold text and line breaks.
- The category is a short descriptive name. ColdFusion generates a check box option for each category to filter the logging window output. This parameter is optional; the default value is `global`. You can specify a standard ColdFusion category or a custom category.
  To log information for a page, you must have a ColdFusion Ajax tag on the page, or use the `cfajaximport` tag. The `cfajaximport` tag does not require any attributes to enable logging.
  The following logging function generates an error level, Pod A category log message:

```
ColdFusion.Log.error("<b>Invalid value:</b><br>" + arg.A, "Pod A");
```

Standard ColdFusion logging messages

ColdFusion automatically logs messages in the following categories:

Category	Description
global	(the default) Messages that are not logged from within the ColdFusion Ajax libraries, for example, initialization of the logging infrastructure.
http	Information about HTTP calls and their responses, including the contents of HTTP requests and information on CFC invocations and responses.
LogReader	Messages about the log display window.
bind	Bind-related actions such as evaluating a bind expression.
widget	Control-specific actions such as tree and grid creation.

© 2014 Adobe Systems Incorporated. All rights reserved.

© 2014 Adobe Systems Incorporated. All rights reserved.

Ajax programming rules and techniques

The following techniques help you to prevent Ajax application errors, improve application security, and develop more effective applications.

Preventing errors

The following rules and techniques help you prevent errors in your applications:

- To ensure that your code works properly, make sure that all your pages, including dynamically loaded content and pages that contain dynamic regions, have valid `html`, `head`, and `body` tags, and that all `script` tags are located in the page head. This is important for any page with ColdFusion Ajax tags and `script` tags, where it ensures that the script code is processed and that code is generated in the correct order. It also prevents problems in some browsers, such as Internet Explorer.
- All JavaScript function definitions on pages that you include dynamically, for example by using a bind expression, the `ColdFusion.navigate`function, or a form submission within a ColdFusion Ajax container tag, must have the following syntax format:

  ```
  functionName = function(arguments) {function body}
  ```

  Function definitions that use the following format might not work:

  ```
  function functionName (arguments) {function body}
  ```

  However, Adobe recommends that you include all custom JavaScript in external JavaScript files and import them on the application's main page, and not write them in-line in code that you get dynamically. Imported pages do not have this restriction on the function definition format.

- In a CFM page, if you call a JavaScript function present in a file that is bound to the page, ensure that you do not use the keyword `var` while declaring the function. `var` declares a function-local scope variable. Therefore, you cannot invoke the JavaScript function from the parent page.
- As a general rule, the `id` attributes or `name` attributes, when you do not specify `id` attributes, of controls should be unique on the page, including on any pages that you specify in `source`attributes. Exceptions to this rule include the following:
  - You can use the same `name` attribute for all options in a radio button group. Bind expressions get information about the selected button.
  - You can use the same `name` attribute for check boxes in a group if you want a single bind expression to get information about all selected controls in the group.
  - If you have multiple similar forms on a page, you might have controls in each form with the same name or ID. You specify the individual controls in bind expressions by including the form name in the bind parameter.
- Do not use an Application.cfc `onRequestEnd` function or onRequestEnd.cfm page that creates output in applications that use the `cfajaxproxy` tag or bind expressions that call CFC functions to get data. ColdFusion Ajax features normally require that all returned data from the server must be in JSON format; the `onRequestEnd` method onRequestEnd.cfm page appends any output as non-JSON information to the end of the returned data.
- By default, all ColdFusion structure element names are in all uppercase characters. Therefore, your client-side Ajax code, such as an `onSuccess` function specified by a `cfajaxproxy` tag, must use uppercase letters for the returned object's element names if you do not explicitly ensure that the element

© 2014 Adobe Systems Incorporated. All rights reserved.

names are not all uppercase. (You can create structure element names with lowercase characters by specifying the names in associative array notation, for example, `myStruct["myElement"]="value"`.)

- ColdFusion Ajax controls throw JavaScript errors if badly formed HTML causes errors in the browser DOM hierarchy order. One example of such badly formed HTML is a table that contains a `cfform` tag, which in turn contains table rows. In this situation, you place the table tag inside the `cfform` tag.
  For browser-specific issues and other issues that could affect application appearance and behavior, see the ColdFusion Release Notes on the Adobe website at www.adobe.com/go/learn_cfu_docs_en, and the ColdFusion Developer Center on the Adobe website at www.adobe.com/go/prod_techarticles.

Improving security

ColdFusion includes several capabilities that help to ensure the security of Ajax application. Also, the ColdFusion Administrator disables output to the client-side logging window by default (see *Enable logging output* in Debugging Ajax applications).

- To prevent cross-site scripting, you cannot use remote URLs in code that executes on the client. For example, if you use a URL such as http://www.myco.com/mypage.cfm in a `cfwindow` tag `source` attribute, the remote page does not load in the window and the window shows an error message. If you must access remote URLs, do so in CFML code that executes on the server, for example, by using a `cfhttp` tag on the page specified by a `source` attribute.
- When a CFC function returns remote data in JSON format, by default, the data is sent without any prefix or wrapper. To help prevent cross-site scripting attacks where the attacker accesses the JSON data, you can tell ColdFusion to prefix the returned data with one or more characters. You can specify this behavior in several ways. The value of an item in the following list is determined by the preceding item in this list:
  1. In the Administrator, enable the Prefix Serialized JSON option on Server Settings > Settings page (the default value is `false`). You can also use this setting to specify the prefix characters. The default prefix is `//`, which is the JavaScript comment marker that turns the returned JSON code into a comment from the browser's perspective. The // prefix helps prevent security breaches because it prevents the browser from converting the returned value to the equivalent JavaScript objects.
  2. Set the `Application.cfc` file `This.secureJSON` and `This.secureJSONPrefix` variable values, or set the `cfapplication` tag `secureJSON` and `secureJSONPrefix` attributes.
  3. Set the `cffunction` tag `secureJSON` attribute. (You cannot use the `cffunction` tag to set the prefix.)As a general rule, use one of these techniques for any CFC or CFML page that returns sensitive data, such as credit card numbers.When you use any of these techniques, the ColdFusion Ajax elements that call CFC functions, including bind expressions and the CFC proxies created by the `cfajaxproxy` tag, automatically remove the security prefix when appropriate. You do not have to modify your client-side code.
- ColdFusion provides capabilities that help prevent security attacks where an unauthorized party attempts to perform an action on the server, such as changing a password. Use the following techniques to ensure that a request to a CFML page or remote CFC function comes from a ColdFusion Ajax feature, such as a bind expression or CFC proxy, that is a valid part of your application:
  - In the `cffunction` tag in a CFC function that returns data to an Ajax client, specify a `verifyClient` attribute with a value of `yes`.
  - At the top of a CFML page or function that is requested by a ColdFusion Ajax client, call the `VerifyClient` ColdFusion function. This function takes no parameters.
    The `VerifyClient` function and attribute tell ColdFusion to require an encrypted security token in each request. To use this function, enable client management or session management in your application; otherwise, you do not get an error, but ColdFusion does not verify clients.Enable client verification only for code that responds to ColdFusion Ajax client code, because only the ColdFusion Ajax library contains the client-side support code. Enabling client verification for clients other than ColdFusion Ajax applications can result in the client application not running.As a general rule, use this function for Ajax requests to the server to perform sensitive actions, such as updating passwords. Typically, do *not* enable client verification for public APIs that do not need protected, search engine

© 2014 Adobe Systems Incorporated. All rights reserved.

web services. Also, do not enable client verification for the top-level page of an application, because the security token is not available when the user enters a URL in the browser address bar.

Programming effectively

The following recommendations help improve or customize your ColdFusion Ajax application.

- Use the AjaxOnLoad function, which specifies a JavaScript function to run when the page loads, to perform any initialization actions that are required for a page to function properly. Use the `AjaxOnLoad` function to call functions when a page is loaded in a container tag. One use for this function could be on a page that pops up a login window if the user is not already logged in when it displays. You can use the `AjaxOnLoad` fu nction to specify a JavaScript function that determines the login status and pops up the window only if necessary.
- Use the following ColdFusion JavaScript functions to access the Ext JS or Yahoo YUI JavaScript library objects that underlie border and tab style `cflayout` controls, `cfwindow` controls, and HTML format `cfgrid` and `cftree`controls. Then use the raw object to modify the displayed control.
  - ColdFusion.Layout.getBorderLayout
  - ColdFusion.Grid.getGridObject
  - ColdFusion.Layout.getTabLayout
  - ColdFusion.Tree.getTreeObject
  - ColdFusion.Window.getWindowObject
    For documentation on the objects and how to manage them, see the Ext documentation at extjs.com/deploy/ext/docs/ and the Yahoo toolkit documentation at developer.yahoo.com/yui/.
    - Preventing errors
    - Improving security
    - Programming effectively

© 2014 Adobe Systems Incorporated. All rights reserved.

Office file interoperability

Adobe ColdFusion provides interfaces to work with PDF, Adobe Flash, and Adobe Connect. ColdFusion 9 extended the integration support to OpenOffice and Microsoft Office application formats such as Excel, PowerPoint, and SharePoint.

Office interoperability supports both OpenOffice and Apache POI libraries (see http://poi.apache.org/ for information on Apache POI). OpenOffice libraries support conversion of all Office file formats, including Word documents to PDF. When you use the `cfdocument`, `cfpresentation`, or `cfspreadsheet` tags to convert Office files, the tags first search for an OpenOffice installation.

If an OpenOffice installation is not found, POI libraries are used. POI libraries support conversion of all office files except Word documents.

See Supported Office conversion formats for the complete list of supported Office conversion formats.

- Using cfdocument
- Using cfpresentation
- Using cfspreadsheet
- Supported Office conversion formats
- SharePoint integration

#back to top

© 2014 Adobe Systems Incorporated. All rights reserved.

Using cfdocument

In addition to the existing functionality, the {{cfdocument}}tag lets you convert Word documents and PowerPoint presentations to PDF. All versions of Microsoft Word and Microsoft PowerPoint from 97 to 2003 are supported.

© 2014 Adobe Systems Incorporated. All rights reserved.

Working with documents using OpenOffice

OpenOffice is an open-source office software that supports word processing, spreadsheets, presentations, and more. OpenOffice stores data in an international open standard format. See http://www.openoffice.org/ for details.

ColdFusion supports OpenOffice, which uses the `cfdocument` tag to convert a Word document (.doc format) to PDF.

When you use `cfdocument` to convert a document file, the tag first checks for an OpenOffice installation. When the OpenOffice installation is found, the tag processes the rich text conversion through the OpenOffice libraries.

The `cfdocument` attributes, `userPassword` and `permissions` are used to open the converted PDF documents. For complete information about the `cfdocument` attributes that support OpenOffice document conversion, see the *CFML Reference*.

To install OpenOffice, see http://download.openoffice.org/index.html . See the ColdFusion Installation Guide for information about installing and configuring OpenOffice.

Example

The following example converts a document, MyDocument.doc, to a PDF file. The PDF conversion occurs only when you specify the `format` attribute as `"pdf"`.

Note

When you convert, specify only the absolute path, for example, "c:\documents\MyDocument.doc"

```
<cfdocument
 format="pdf"
 srcfile="C:\documents\MyDocument.doc"
 filename="C:\documents\MyDocument.pdf">
</cfdocument>
```

Note

If you do not specify the filename attribute, the converted PDF opens in a browser.

© 2014 Adobe Systems Incorporated. All rights reserved.

Working with PowerPoint presentation files

You use the `cfdocument` tag to convert a PowerPoint presentation (PPT file) to a PDF document.

Example

The following example converts a PowerPoint presentation to a PDF file.

```
<cfdocument
 format="pdf"
 srcfile="C:\presentations\MyPresentation.ppt"
 filename="C:\presentations\MyPresentation.pdf">
</cfdocument>
```

#back to top

© 2014 Adobe Systems Incorporated. All rights reserved.

Using cfpresentation

The `cfpresentation` tag is the parent tag for one or more `cfpresentationslide` tags, where you define the content for the presentation, and the `cfpresenter` tags, which provide information about the people presenting the slides.

You use the `cfpresentation` tag to convert a PowerPoint presentation to an Adobe Connect presentation or HTML. Browsers like Internet Explorer, Mozilla Firefox, and Safari are all compatible with the conversion from PPT to a Connect presentation or HTML.

For complete information about `cfpresentation`, and `cfpresentationslide`, see *CFML Reference*.

© 2014 Adobe Systems Incorporated. All rights reserved.

Example for converting from PowerPoint to Connect

The following example converts a PowerPoint presentation to an Adobe Connect presentation.

```
<cfpresentation
 title="my presentation"
 directory="C:\presentations\"
 overwrite=true>
 <cfpresentationslide
  src="#ppttemplate#backgrounds.ppt"
  slides="1">
 </cfpresentationslide>
 <cfpresentationslide
  duration="4"
  video="video1.flv">
   Sample slide
 </cfpresentationslide>
</cfpresentation>
```

The following example converts an HTML file to a PowerPoint presentation.

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cfpresentation
 title = "text string"
 format= "ppt"
 destination="#generated#html_to_ppt_01.ppt"
 backgroundColor = "YELLOW"
 overwrite = "yes">
 <cfpresentationslide
  Title="Q1 Sales Figures"
  duration="14">


<h3>Q1 Sales Figures</h3>
<cfchart
 format="png"
 showborder="yes"
 chartheight="250"
 chartwidth="300"
 pieslicestyle="sliced">
 <cfchartseries type="pie">

 <cfchartdata
 item="Europe"
 value="9">

<cfchartdata
 item="Asia"
 value="20">

<cfchartdata
 item="North America"
 value="50">

<cfchartdata
 item="South America"
 value="21">
</cfchartseries>
</cfchart>
</cfpresentationslide>

<cfpresentationslide
 src="cfdocument_pos24.html"
 duration="15" />
</cfpresentation>
```

The following example converts a PowerPoint presentation to a Connect presentation.

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cfpresentation
 title="my presentation"
 directory="C:\presentations"
 overwrite=true>
 <cfpresentationslide
  src="#ppttemplate#backgrounds.ppt"
  slides="1-3,5">
 </cfpresentationslide>
 <cfpresentationslide
  duration="4"
  video="video1.flv">
   Sample slide
 </cfpresentationslide>
</cfpresentation>
```

#back to top

© 2014 Adobe Systems Incorporated. All rights reserved.

Using cfspreadsheet

The `cfspreadsheet` tag lets you manage Excel spreadsheets. The tag lets you do the following:

- Read a spreadsheet file (XLS file) and store it in a ColdFusion spreadsheet object, query, CSV string, or HTML string.
- Write a single sheet to a new XLS from a query, ColdFusion spreadsheet object, or CSV string variable.
- Add a sheet to an existing XLS file.
  Use the spreadsheet functions to manipulate rows and columns in a spreadsheet and the data in the rows and columns. You can also specify and get comments, values, and formulas for a cell in the spreadsheet.
  Microsoft Office Excel 2007 is supported by `cfspreadsheet` and all the spreadsheet functions except the following:
- `SpreadSheetAddSplitPane`
- `SpreadSheetAddFreezePane`
  For detailed information about `cfspreadsheet` and all the spreadsheet functions, see *CFML Reference*.

© 2014 Adobe Systems Incorporated. All rights reserved.

Example- storing the spreadsheet data in a CSV string

The following example reads the spreadsheet file - SingleSheet.xls and stores the spreadsheet data in a CSV string.

```
<cfspreadsheet action = "read"
 format="csv"
 src="C:\documents\SingleSheet.xls"
 name="csvvar"
 rows="1-4,5,6,7-8">
 <cfoutput>#csvvar#</cfoutput>
```

The following example reads the spreadsheet file - template_02.xls and stores the spreadsheet data in a query.

```
<cfspreadsheet
  action = "read"
  src="C:\dcuments\template_02.xls"
  query="excelquery"
  sheet="1"
  rows="1-3,4-5"
  columns="1,4">
<cfoutput
  query="excelquery"
  startrow="1"
  maxrows="#excelquery.recordcount#">
  #excelquery.col_1#
  #excelquery.col_2#
</cfoutput>
```

The following example reads a spreadsheet file - template_08_Charts_Graph.xls and stores the spreadsheet data in an HTML string.

```
<cfspreadsheet
  action = "read"
  format="html"
  src="C:\documents\template_08_Charts_Graph.xls"
  name="report1"
  rows="5-11"
  columns="1-6">
<cfoutput>
  #report1#
</cfoutput>
```

The following example uses data from a query and writes it to a single sheet in the spreadsheet file - SingleSheet1.xls

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cfquery
  name="excelquery"
  datasource="cfdocexamples">
  SELECT PARKNAME, REGION, STATE FROM Parks WHERE STATE='WI'
  ORDER BY ParkName, State
</cfquery>
<cfspreadsheet
  action = "write"
  filename="C:\SingleSheet1.xls"
  query="excelquery"
  overwrite="true">
```

#back to top

© 2014 Adobe Systems Incorporated. All rights reserved.

Supported Office conversion formats

The following table lists the conversion formats supported by Office applications, and the CFML tags that support the conversion. It also shows whether OpenOffice installation is required for the conversion.

All versions of Microsoft Word and Microsoft PowerPoint from 97 to 2003 are supported. Also, all versions of Microsoft Excel from Versions 97 to 2007 are supported.

Format		**CFML Tag**	**OpenOffice installation**
From	**To**		
PPT	Connect Presentation	`cfpresentation`	optional
PPT	HTML	`cfpresentation`	optional
PPT	PDF	`cfdocument`	optional
HTML	PPT	`cfpresentation`	not required
Excel	HTML	`cfspreadsheet`	not required
Excel	Query	`cfspreadsheet`	not required
Excel	In-memory Variable	`cfspreadsheet`	not required
Query	Excel	`cfspreadsheet`	not required
In-memory variable	Excel	`cfspreadsheet`	not required
Word	PDF	`cfdocument`	required

#back to top

© 2014 Adobe Systems Incorporated. All rights reserved.

SharePoint integration

You can use ColdFusion with Microsoft Windows SharePoint Services 2.0 or 3.0, and Microsoft Office SharePoint Portal Server 2003 or Microsoft Office SharePoint Server 2007. You can integrate ColdFusion applications with SharePoint features that are exposed as web service actions.

© 2014 Adobe Systems Incorporated. All rights reserved.

Load SharePoint actions from ColdFusion

Use the `cfsharepoint` tag to access exposed SharePoint features, without loading the web services directly. The `cfsharepoint` tag works only if the SharePoint server uses basic authentication. By default, ColdFusion supports a subset of features that are exposed as web services in SharePoint. If a web service is not supported in ColdFusion, specify the URL to the WSDL (Web Services Description Language) of the web service that is loaded.

For information about the supported SharePoint features, see the *CFML Reference*.

© 2014 Adobe Systems Incorporated. All rights reserved.

Using cfsharepoint

Sharepoint integration with ColdFusion helps you dynamically manage user lists, views, and groups; work with images and document workspaces; and use search effectively. The `cfsharepoint` tag lets you create new lists, retrieve list items, and update list items on the SharePoint server.

The following example shows how to create a picture library list called "getpics".

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cfsharepoint
 action="addlist"
 login="#login#"
 params="#{ listname ="getpics",
 description="This a picture library list",
 templateId= "109 " }#"/>
 <! --- Creates a folder within the picture library list>
< cfsharepoint
 action  ="create new folder"
 login= "#login#"
 name="collection1"
 params="#{strListName="getpics",
 strParentFolder=""}#"/>
<!--- Uploads pictures to the folder that you created --->
<cfscript>
 myimage = filereadbinary(expandpath("Bird.jpg"));
 //convert the image into byte array to pass as input for "upload" action.
</cfscript>
<cfsharepoint
 action="upload" login="#login#"
 params="#{strListName="testpics",
 strfolder="Collection1",
 bytes="#myimage#",
 filename="bird.jpg",
 fOverwriteifexist=true}#"/>

<!--- Rotates the picture downloaded from the SharePoint server.--->
<cfsharepoint
 name ="result1"
 action="download"
 login="#login#"
 params="#{strListName="getpics",
 strfolder="New Folder",
 itemFileNames=["bird.jpg"],type=1,
 fFetchoriginalIfNotAvailable=true}#"/>
<cfimage
 action="rotate"
 source="#result1.file#"
 isbase64="yes"
 angle="45"
 name="temp"
 destination="bird.jpg"
 overwrite="yes"/>
<cfscript>
baseimage = filereadbinary(expandpath("bird.jpg"));
//convert the image into byte array to pass as input for "upload" action.
</cfscript>
<!--- Uploads the rotated image back to the SharePoint server --->
<cfsharepoint
action="upload"
login="#login#"
params="#{strListName="getpics",strfolder="Collection1",
bytes="#baseimage#",filename="bird.jpg",fOverwriteifexist=true}#"/>
```

To check and ensure that all the updates are made, you can retrieve the list items using code like the following:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cfsharepoint
 action="getimaginglistitems"
 login="#login#" name="result"
 params="#{strListName="getpics",strFolder="#result3.title#"}#"/>
<cfloop array="#result.Library#" index="n">
<cfif n.ows_FileLeafRef  contains "temppicrotate.jpg">
SUCCESS
<cfbreak>
</cfif>
</cfloop>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

Access ColdFusion from SharePoint using custom Web Parts

You can access ColdFusion applications from within SharePoint using custom Web Parts. You can create a custom Web Part using the Page Viewer Web Part template that is shipped, by default, with SharePoint services 2.0 and 3.0, and Microsoft Office SharePoint Portal Server 2003 or Microsoft Office SharePoint Server 2007.

1. From the SharePoint Server page, click Modify Shared Page.
2. Select Add Web Part.
3. Click Browse from the pop-up menu. The Web Parts list appears.
4. Select Page Viewer Web Part.
5. Click Add. The Page Viewer Web Part loads.
6. Click the Open the Tools Pane link.
7. Specify the URL of the ColdFusion application in the URL text field. The ColdFusion application loads within the Web Part.

© 2014 Adobe Systems Incorporated. All rights reserved.

Use Single Sign-On to access ColdFusion applications via SharePoint

SharePoint custom Web Parts let you access multiple ColdFusion applications from the SharePoint server using Single Sign-On (SSO). After signing in, users can access multiple secure ColdFusion applications by accessing ColdFusion services from multiple Web Parts.

> **Note**
>
> For single sign-on capability, you need Microsoft Office SharePoint Portal Server 2003 or Microsoft Office SharePoint Server 2007.

To make a ColdFusion application available from SharePoint, use the CFSharepoint SSO WebPart template. This template is a customized version of PageViewer WebPart. It enables you to pass SSO credentials to the ColdFusion application. Download this template from the Adobe website or copy it from the ColdFusion 9 DVD.

Remember these points:

- Web Parts support only the native single sign-on solution; other pluggable single sign-on services are not supported.
- Only single sign-on credentials are passed to the ColdFusion application. The ColdFusion application must have the necessary logic to retrieve the credentials and login to the application.

Deploy the CF9SSOWebPart.wsp Web Part for Microsoft Office SharePoint Server 2007

To configure single sign-on for Microsoft Office SharePoint Server 2007, deploy the CF9SSOWebPart.wsp file to the SharePoint server.

1. Copy the CF9SSOWebPart.wsp file to the BIN folder within the Web Server extensions. It is normally located at Program Files\Common Files\Microsoft Shared\Web Server Extensions\12\BIN in the SharePoint server.
2. To deploy the solution to SharePoint, use the command prompt to navigate to Program Files\Common Files\Microsoft Shared\Web Server Extensions\12\BIN and enter the following commands, as required.To delete the solution if it is already present:

   ```
   STSADM.EXE -o deletesolution -name CF9SSOWebPart.wsp -override
   ```

   To add the solution to SharePoint:

   ```
   STSADM.EXE -o addsolution -f CF9SSOWebPart.wsp
   ```

   To deploy the solution to the configured website by specifying the URL:

   ```
   STSADM.EXE -o deploysolution -name CF9SSOWebPart.wsp
   -url <virtual server url> -local -allowGacDeployment
   ```

   To deploy the solution to all the configured websites:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
STSADM.EXE -o deploysolution -name CF9SSOWebPart.wsp -allcontenturls -local
-allowGacDeployment
```

Import the CF9SSOWebPart.wsp Web Part into a Web Part Page

1. Navigate to the web page on the SharePoint server where you want the Web Part to be accessible.
2. In the Web Part page, click Site Actions > Site Settings.
3. In the Site Settings page, click Galleries > Web Parts.
4. In the Web Part gallery, click Upload in the toolbar pane.
5. Select the CF9SSOWebPart.wsp Web Part.
6. Enter the following details in the toolbar pane.
   - URL of the ColdFusion application to access
   - The form field name as the User ID
   - The form field name as the password
   - Name of the SSO application where the credentials are configured
     Once the Web Part is deployed, it takes the credentials from the SharePoint Single Sign-On database (based on the application name entered in the Tools Pane). These credentials are transferred to the ColdFusion application through the URL (provided in the Tools Pane) in a FORM containing the specified form fields.

Deploy the CF9SharepointSSOCab.CAB Web Part for Microsoft Office SharePoint Portal Server 2003

To configure single sign-on for Microsoft Office SharePoint Portal Server 2003, deploy the CAB file - CF9SharepointSSOCab.CAB

1. Copy the CF9SharepointSSOCab.CAB to the BIN folder within the Web Server extensions. It is normally located at Program Files\Common Files\Microsoft Shared\Web Server Extensions\60\BIN in the SharePoint server.
2. Create a folder named CFSharePointSSO within the Layouts folder. The Layouts folder is normally located at Program Files\Common Files\Microsoft Shared\Web Server Extensions\60\Template\Layouts.
3. From the CF9SharepointSSOCab.CAB file, copy the template file CFSSO.aspx to the CFSharePointSSO folder that you created in the previous step.
4. In the command prompt, navigate to Program Files\Common Files\Microsoft Shared\Web Server Extensions\60\bin and enter the following command to add the CAB file:

   ```
   stsadm.exe -o addwppack -filename CF9SharepointSSOCab.CAB -globalinstall
   ```

   If the CAB file exists, delete the CAB file and then enter the command to add the CAB file as follows:

   ```
   stsadm.EXE -o deletewppack -name CF9SharepointSSOCab.CAB
   ```

```
stsadm.exe -o addwppack -filename CF9SharepointSSOCab.CAB -globalinstall
```

Configure the CFSharepointSSO Web Part

© 2014 Adobe Systems Incorporated. All rights reserved.

1. In the Site Settings page, go to Manage Security and Additional Settings > Manage Web Part Gallery.
2. Click New in the toolbar in the Web Part Gallery. The New Web Parts List appears.
3. Select the CFSSOwebpart.dwp Web Part and click Populate Gallery.
4. Add the CFSharepointSSO Web Part to the Virtual Server Gallery. Clicking Edit Page > Modify Shared Page > Add Web Parts > Browse > Virtual Server Gallery. Then add the Web Part to the Web Parts list.
5. After adding the CFSharepointSSO Web Part, click the Tools pane and enter the following details.
    - URL of the ColdFusion application to access
    - The form field name as the User ID
    - The form field name as the password
    - Name of the SSO application where the credentials are configured

Once the Web Part is deployed, it takes the credentials from the SharePoint Single Sign-On database (based on the application name entered in the Tools Pane). These credentials are transferred to the ColdFusion application through the URL (provided in the Tools Pane) in a FORM containing the specified form fields.

- Deploy the CF9SSOWebPart.wsp Web Part for Microsoft Office SharePoint Server 2007
- Import the CF9SSOWebPart.wsp Web Part into a Web Part Page
- Deploy the CF9SharepointSSOCab.CAB Web Part for Microsoft Office SharePoint Portal Server 2003
- Configure the CFSharepointSSO Web Part

© 2014 Adobe Systems Incorporated. All rights reserved.

ColdFusion Portlets

You can now build your own portlets by leveraging Adobe ColdFusion components (CFCs). You can create your own portlet using ColdFusion and run it on:

- JBoss portal server
- WebSphere portal server 6.1

- Run a ColdFusion portlet on JBoss Portal Server
- Run a ColdFusion portlet on WebSphere Portal Server
- Common methods used in portlet.cfc
- ColdFusion portlet components
- JSR-286 Support

#back to top

© 2014 Adobe Systems Incorporated. All rights reserved.

Run a ColdFusion portlet on JBoss Portal Server

You can run and access ColdFusion portlets on a JBoss portal server, which can be either local or remote.

- A local host: A portal can access portlets on the same computer where JBoss Portal server exists.
- A remote host: A portal can access portlets deployed on a remote ColdFusion server instance.

© 2014 Adobe Systems Incorporated. All rights reserved.

Prerequisites

Before you start developing ColdFusion portlets, you must:

- Install JDK 1.5.x.
- For JSR-168, install bundled version of JBoss 2.6.7 or 2.6.8 portal server and JBoss 4.2.3 Application server. For JSR-286, install bundled version of JBoss 2.7.2 portal server and JBoss 4.2.3 Application server.
- Deploy ColdFusion on JBoss application server.
- Enable the J2EE sessions in ColdFusion Administrator.

The JBoss Server does not always add Expires directive in the set cookie header. To fix this, you need to add the following JVM argument:

```
-Dorg.apache.tomcat.util.http.ServerCookie.ALWAYS_ADD_EXPIRES=true
```

© 2014 Adobe Systems Incorporated. All rights reserved.

Build a portlet for a local server

To write a ColdFusion portlet:

1. Create a CFC that extends the `CFIDE.portlets.ColdFusionPortlet` package. For example, the following `HelloPortlet.cfc` extends this package and defines the `doView()` and `doHelp()`methods:

```
<cfcomponent extends="CFIDE.portlets.ColdFusionPortlet">

 <cffunction name="doView" returntype="void" output="true">
  <cfargument name="renderRequest" type="any" required="true" hint="A
  javax.portlet.RenderRequest java object">
  <cfargument name="renderResponse" type="any" required="true" hint="A
  javax.portlet.RenderResponse java object">
  <cfoutput>
   Hello World ColdFusion Portlet
  </cfoutput>
 </cffunction>
 <cffunction name="doHelp" returntype="void" output="true">
  <cfargument name="renderRequest" type="any" required="true" hint="A
  javax.portlet.RenderRequest java object">
  <cfargument name="renderResponse" type="any" required="true" hint="A
  javax.portlet.RenderResponse java object">
  <h1>ColdFusion Help</h1>
  <p>This is a Help message for the Hello Portlet.</p>
 </cffunction>
</cfcomponent>
```

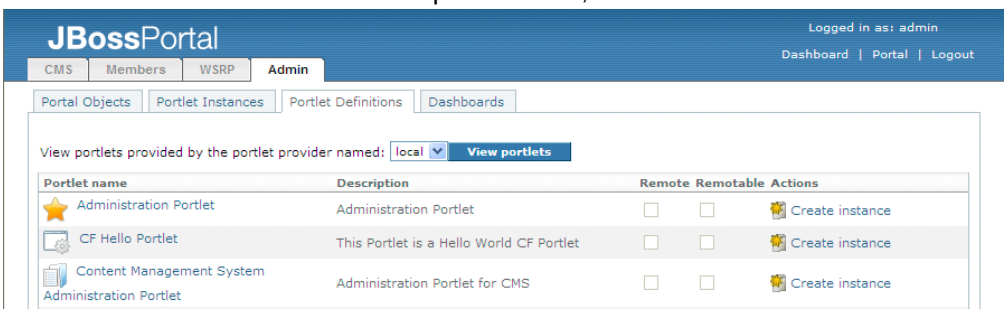
2. Save `HelloPortlet.cfc }}in {{<jboss_server_home>\server\default\deploy\cfusion.ear\cfusion.war\portlets\hello`.
3. Define `HelloPortlet.cfc` in `portlet.xml`, which is located in the WEB-INF folder of ColdFusion web root. The `portlet.xml`file looks similar to the following:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<?xml version="1.0" encoding="UTF-8"?>
<portlet-app xmlns="http://java.sun.com/xml/ns/portlet/portlet-app_1_0.xsd"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/portlet/portlet-app_1_0.xsd
http://java.sun.com/xml/ns/portlet/portlet-app_1_0.xsd" version="1.0">
 <portlet>
  <description xml:lang="en">This Portlet is a Hello World CF
Portlet</description>
  <portlet-name>ColdFusionPortlet</portlet-name>
  <display-name xml:lang="en">CF Hello Portlet</display-name>
  <portlet-class>coldfusion.portlet.ColdFusionPortlet</portlet-class>
  <init-param>
    <name>cfcName</name>
      <value>portlets.hello.HelloPortlet</value>
  </init-param>
  <supports>
   <mime-type>text/html</mime-type>
   <portlet-mode>VIEW</portlet-mode>
   <portlet-mode>HELP</portlet-mode>
  </supports>
  <portlet-info>
   <title>ColdFusion Hello World Portlet</title>
  </portlet-info>
 </portlet>
</portlet-app>
```

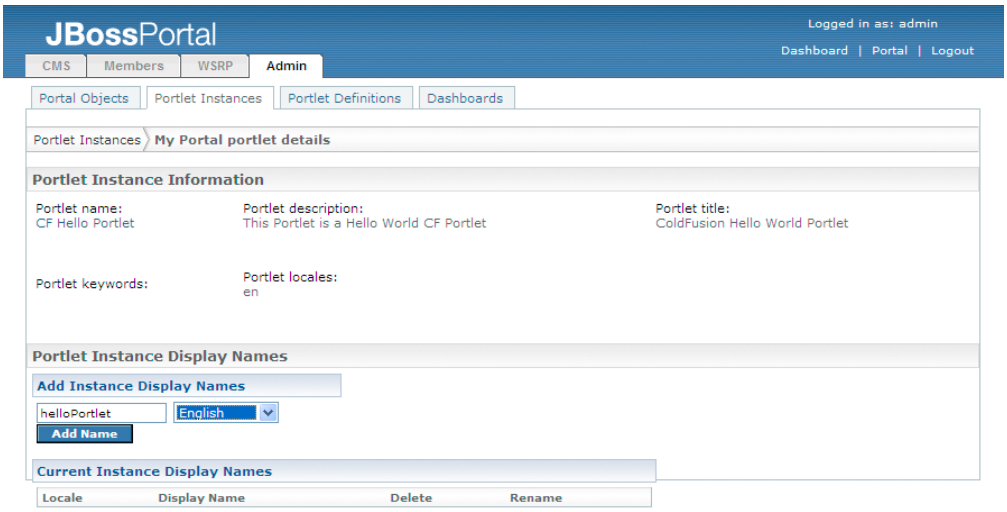
The portlet is defined and registered for portlet definition, with targeted cfcName defined as the INIT parameter. The INIT parameter value must be from the web root of ColdFusion.

4. Run the JBoss server by running one of the following commands: On UNIX{{<JBoss_home>/bin/run.sh}} On Windows<`JBoss_home>\bin\run.bat}}By default JBoss only binds to localhost. You can have it bind to any IP address by running {{bin/run.sh -b 0.0.0.0 }}on UNIX or {{bin/run.bat -b 0.0.0.0` on Windows.
5. Launch the JBoss portal server. By default, JBoss binds to port 8080, so launch the server using the URL: http://<matching_IP>:<port>/portal/. For example: http://127.0.0.1:8080/portal.
6. Log in to the portal by clicking the login link in the upper-right corner. The default credentials are admin/admin.
7. Click the Admin option in the upper-right corner.
8. Click the Portlet Definitions tab. The portlet name, CF HelloPortlet is listed here.

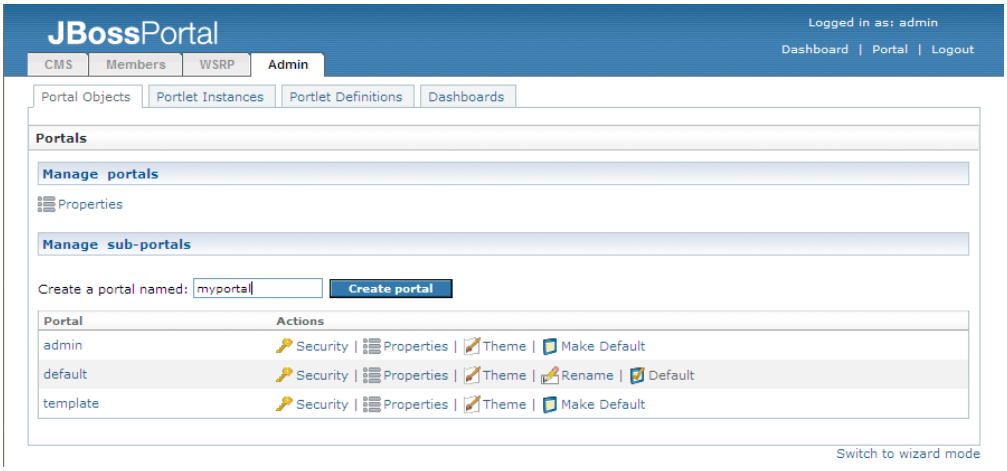


9. Create an instance of this portlet by clicking Create Instance under Actions.
10. Specify the instance name.
11. Add the instance display name.

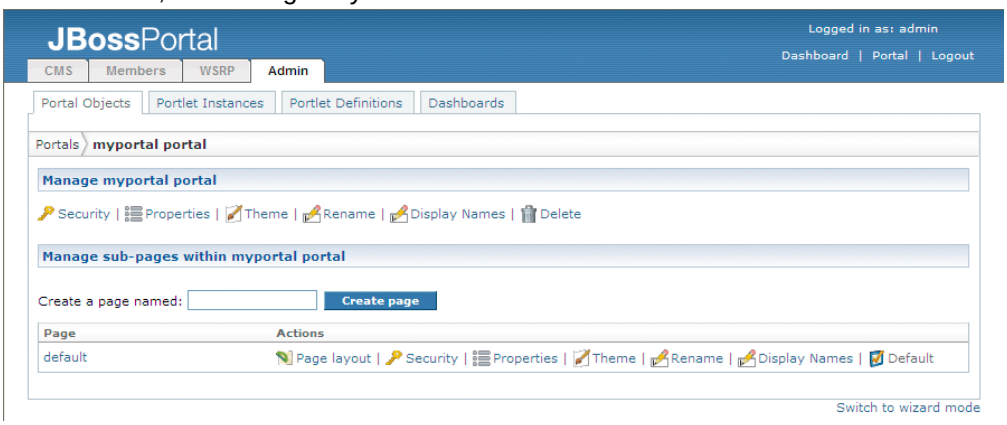
© 2014 Adobe Systems Incorporated. All rights reserved.



12. Click the Portal Objects tab.
13. Create a new portal page by specifying the portal name in the Create a portal named box.

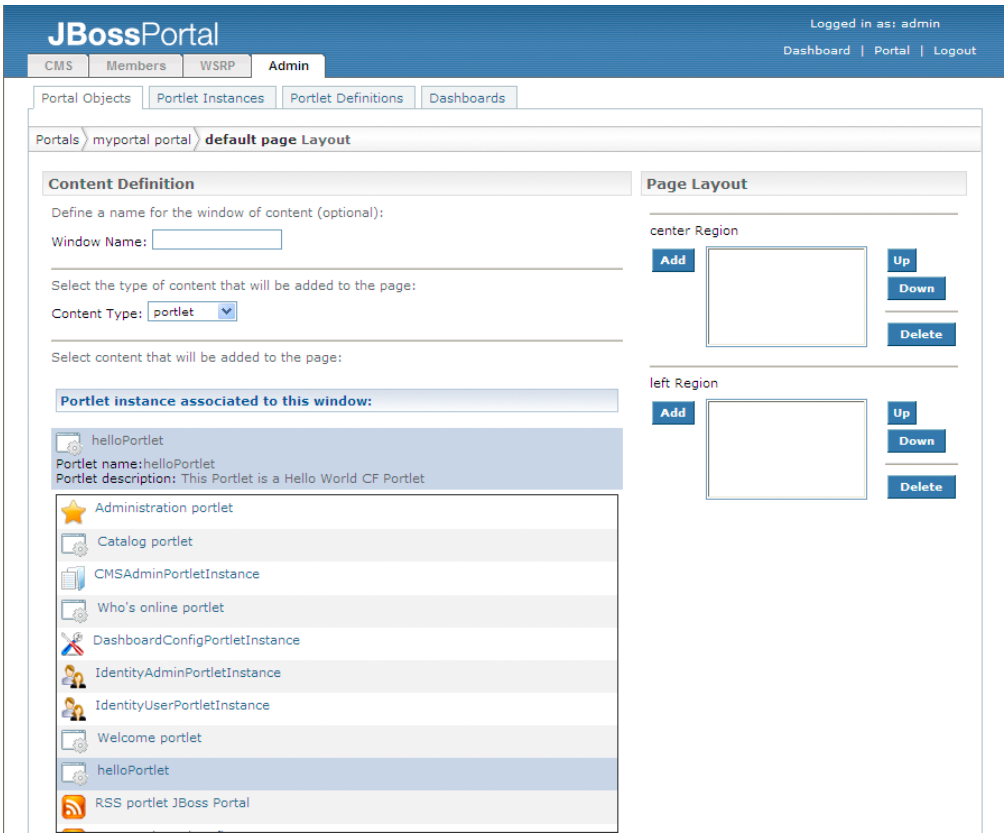


14. Select the created portal page from the list of portals.
15. From Actions, select Page Layout.

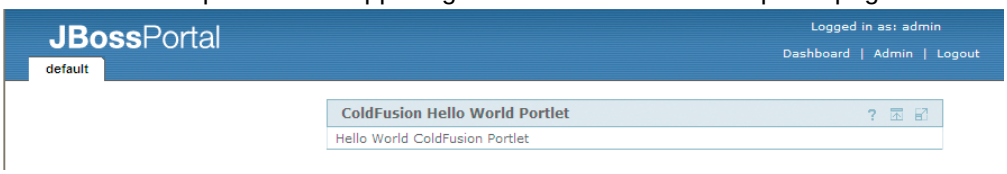


16. From the Portlet instance associated with this Window list, select the portlet instance you created in step 10.

© 2014 Adobe Systems Incorporated. All rights reserved.



17. In the Page Layout section, click Add for the center region or left region to add the portlet container in the required area.
18. Go back to the Portal Objects > Portals page and select Make Default option to set the new portal as default.
19. Click the Portal option on the upper-right corner to view the new portal page containing the portlet.



© 2014 Adobe Systems Incorporated. All rights reserved.

Access remote ColdFusion portlets (WSRP)

To access remote ColdFusion portlets and expose them as web services using the Web Service Response Protocol (WSRP), configure the following:

- WSRP Producer: The Web Service Response Protocol (WSRP) is used to create the producer of the WSDL. A WSRP producer can be any ColdFusion instance (Server or JEE).
- WSRP Consumer: Portlets are exposed as web services, which are then consumed by the portal server (JBoss).

Configure the WSRP producer

1. Create the ColdFusion HelloPortlet.cfc as described in step 1 of Build a portlet for a local server.
2. Save the `HelloPortlet.cfc`file in the following directory:

```
<ColdFusion_webroot>/portlets/hello/
```

3. Define `HelloPortlet.cfc` in the `cf-wsrp-portlet.xml`file.

```
<?xml version="1.0" encoding="UTF-8"?>
<portlet-app xmlns="http://java.sun.com/xml/ns/portlet/portlet-app_1_0.xsd"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/portlet/portlet-app_1_0.xsd
http://java.sun.com/xml/ns/portlet/portlet-app_1_0.xsd"
version="1.0">
 <portlet>
  <description xml:lang="en">This Portlet is a Hello World CF
Portlet</description>
  <portlet-name>HelloPortlet</portlet-name>
  <display-name xml:lang="en">Hello Portlet</display-name>
  <portlet-class>portlets.hello.HelloPortlet</portlet-class>
  <supports>
   <mime-type>text/html</mime-type>
   <portlet-mode>VIEW</portlet-mode>
   <portlet-mode>HELP</portlet-mode>
  </supports>
  <supported-locale>en</supported-locale>
  <portlet-info>
   <title>Hello ColdFusion Portlet</title>
   <short-title>CF Hello</short-title>
   <keywords>hello, portlet</keywords>
  </portlet-info>
 </portlet>
</portlet-app>
```

After updating cf-wsrp-portlet.xml, restart your ColdFusion instance.

Configure the WSRP consumer

1. Run the JBoss server by running the command: On UNIX{{bin/run.sh}} On Windows{{bin\run.bat}}By default JBoss only binds to localhost. You can have it bind to any IP address by running `bin/run.sh -b 0.0.0.0 }}on UNIX or {{bin\run.bat -b 0.0.0.0` on Windows.
2. Launch the JBoss portal server. By default, JBoss binds to port 8080, so launch the server using the URL: http://localhost:8080/portal/.

© 2014 Adobe Systems Incorporated. All rights reserved.

3. Log in to the portal by clicking the login link in the upper-right corner. The default credentials are admin/admin.
4. Click the Admin option in the upper-right corner.
5. Click the WSRP tab.
6. Specify the WSRP consumer name, such as *wsrp-test* in the Create a Consumer Named box.
7. Click Create Consumer.
8. On the next page, specify the WSDL URL of the WSRP producer:*http://<WSRPProducer_IP>:<port>/<context_root>/WSRPProducer?wsdl*
9. Click Refresh & Save. If successful, the following message appears"Refresh was successful."

> **Note**
>
> You can try a demo WSRP consumer hosted by BEA, by entering the WSDL URL: http://wsrp.bea.com:7001/producer/producer?WSDL. After clicking Refresh & Save, the system prompts you to provide the consumerRole registration property. Use the string "public" and click Update Properties. If the BEA WSRP Producer does not get configured, the problem could be with the JBoss Portal Server, rather than the WSRP Producer. Make sure that you are using Java 1.5 JVM.

1. Click the Portlet Definitions tab.
2. Select *wsrp-test* from the View portlets provided by the portlet provider named drop-down list.
3. From the Select the WSRP Portlet You Created box, select the WSRP portlet.
4. Click View Portlets.
5. From Actions column, select Create Instance.
6. Specify the instance name and click Create instance.
7. Optionally, specify the instance display name and other details.
8. Go back to Portal Objects and select default. Alternatively, you can create your own portal object.
9. From the Actions column, select Page Layout.
10. From the Portal Instance Associated to this Window list, select the WSRP portlet instance you created earlier.
11. From the Page Layout section, add the instance to the required region by clicking Add.
12. Go back to the Portal Objects > Portal page.
13. Click Make Default to make the portal which contains the WSRP portlet as default.
14. Click the Portal link on the upper-right corner to view the portal page.

#back to top

© 2014 Adobe Systems Incorporated. All rights reserved.

Run a ColdFusion portlet on WebSphere Portal Server

To access and run ColdFusion portlets on WebSphere Portal Server 6.1:

1. Create cfusion.war file.
2. Extract it using `jar -xvf cfusion.war`
3. Create a `/portlets` directory under cfusion.war.
4. Add portlets to the `/portlets` directory. Add portlet entries to portlet.xml, present at: {{cfusion.war/WEB-INF/portlet.xm}}l
5. Go to repackage the WAR file using `jar cvf cfusion.war`.
6. Deploy it through WebSphere Portal Server administrator console. The portlets would be visible present in portlet.xml
7. Create a portal page and add the portlets.

© 2014 Adobe Systems Incorporated. All rights reserved.

Common methods used in portlet.cfc

Some common methods that are used frequently while creating the ColdFusion component portlet, such as HelloPortlet.cfc, are:

Method	Description	Syntax
doView()	This method renders the portlet content. It is called by the portlet container to allow the portlet to generate the content of the response based on its current state.	{{<cffunction name="doView" returntype="void" output="true">\<cfargument name="renderRequest" type="any" required="true" hint="A javax.portlet.RenderRequest java object">\<cfargument name="renderResponse" type="any" required="true" hint="A javax.portlet.RenderResponse java object">\<!--- User code goes here ---> </cffunction>}}
doHelp()	Helper method to serve up the HELP mode.	{{<cffunction name="doHelp" returntype="void" output="true">\<cfargument name="renderRequest" type="any" required="true" hint="A javax.portlet.RenderRequest java object">\<cfargument name="renderResponse" type="any" required="true" hint="A javax.portlet.RenderResponse java object">\<!--- User code goes here ---> </cffunction>}}
doEdit()	Helper method to serve up the EDIT mode.	{{<cffunction name="doEdit" returntype="void" output="true">\<cfargument name="renderRequest" type="any" required="true" hint="A javax.portlet.RenderRequest java object">\<cfargument name="renderResponse" type="any" required="true" hint="A javax.portlet.RenderResponse java object">\<!--- User code goes here ---> </cffunction>}}

© 2014 Adobe Systems Incorporated. All rights reserved.

ProcessAction()	Called by the portlet container to allow the portlet to process an action request	{{<cffunction name="processAction" returntype="void" access="public" output="false" hint="Called by the portlet container to allow the portlet to process an action request.">
		<cfargument name="actionRequest" type="any" required="true" hint="A javax.portlet.ActionRequest java object">{{
		<cfargument name="actionResponse" type="any" required="true" hint="A javax.portlet.ActionResponse java object">{{
		<!--- User code goes here -->
		</cffunction>}}

Init()	Called by the portlet container to indicate to a portlet that the portlet is being placed into service	{{<cffunction name="init" returntype="void" access="public" output="false" hint="Called by the portlet container to indicate to a portlet that the portlet is being placed into service."> <cfargument name="portletConfig" type="any" required="true" hint="A javax.portlet.PortletConfig java object">{{ <!--- User code goes here --> </cffunction>}}
processEvent	This is used to consume the event once it is published.	`<cffunction name="processEvent" returntype="void" access="public" output="false" hint="Called by the portlet container requesting the portlet to process a specific event."> <!--user code--> </cffunction>`

#back to top

© 2014 Adobe Systems Incorporated. All rights reserved.

ColdFusion portlet components

You can configure your ColdFusion portlet components to define its modes, window states, title, scope, and parameters.

For references of ColdFusion Portlet API, see the JSR-168 specification for all{{ javax.portlet.*}} classes.

Currently, WSRP 1.0 is the supported standard for portlets.

© 2014 Adobe Systems Incorporated. All rights reserved.

Portlet modes

Portal servers typically allow three portlet modes: View, Edit, and Help.

The View mode is the default state when rendering a portlet. The portlet window has links in the title bar that enable you to change the mode to Help or Edit.

To add a Help mode view, add the `doHelp()` with the same signature as the `doView()` function.

To support the edit mode create the `doEdit()`.

© 2014 Adobe Systems Incorporated. All rights reserved.

Portlet window states

Most portal servers support three window states (normal, minimized, and maximized). You can obtain the current window state by calling the `getWindowState()` method of the `ColdFusionPortlet` base component.

© 2014 Adobe Systems Incorporated. All rights reserved.

Portlet title

To set the portlet title, add a method to the CFC called `getTitle` as follows:

```
<cffunction name="getTitle" returntype="string" output="false" access="public">
  <cfargument name="renderRequest" type="any" required="true" hint="A
  javax.portlet.RenderRequest java object">
  <cfreturn "My ColdFusion Portlet">
</cffunction>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

Portlet scope

The ColdFusion portlet toolkit defines the variable `request.portlet`. It contains the following structures:

```
request.portlet.parameters - Parameters of the Portlet Request
request.portlet.attributes - attributes of the Portlet Request
request.portlet.properties - properties of the Portlet Request
```

These variables are defined for convenience and convention.

© 2014 Adobe Systems Incorporated. All rights reserved.

Create portlet parameters

To create different page views within your portlet you can configure the render parameters such as `renderURL`. For example, to set the `renderURL` parameter:

1. Create a `renderURL`parameter

```
<cfset params = StructNew()>
<cfset params.page = "somepage">
<cfoutput><a href="#createRenderURL(params)#">Link to somepage</a>
```

1. Check for parameter in the page and render conditionally:

```
<cfparam name="request.portlet.parameters.page" default="">
<cfif request.portlet.parameters.page IS "somepage">
 <cfinclude template="somepage.cfm">
<cfelse>
 <!--- put step 1 here --->
</cfif>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

Processing actions using form posts

To process a form post, use the `createActionURL()` function, which generates the form action URL. For example:

```
<cfoutput>
 <form action="#createActionURL()#" method="post">
  Value:   <input type="text" name="action_value" >
    <input type="submit" value="Process Action" />
 </form>
</cfoutput>
```

When the form is submitted, the portal container calls the `processAction()` method in your CFC. So, add this method as follows:

```
<cffunction name="processAction" returntype="void" access="public" output="false"
hint="Called by the portlet container to allow the portlet to process an action
request.">
 <cfargument name="actionRequest" type="any" required="true" hint="A
 javax.portlet.ActionRequest java object">
 <cfargument name="actionResponse" type="any" required="true" hint="A
 javax.portlet.ActionResponse java object">
 <cfif IsDefined("request.portlet.parameters.action_value")>
 <!--- do something with this value, such as update your database --->
 </cfif>
</cffunction>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

Examples

The following examples show how you can configure portlets. You can add the following code to the `doView()` met hod depending on whether you are configuring the portlet on a local or remote server.

- To get portal user information:JSR:

```
<cfoutput>#renderRequest.getRemoteUser()#</cfoutput>
```

  WSRP:

```
<cfdump var = #renderRequest.getAttribute("javax.portlet.userinfo")#>
```

- To display PDF:

```
<cfdocument format="pdf" src="http://www.google.com" filename="cfdoc1.pd f "
overwrite="true">
</cfdocument>
<cfset pdfURL =
getPortletResponse().encodeURL(getPortletRequest().getContextPath() & "/<path
of pdf>/cfdoc1.pdf")>
<cfoutput>
 <object data="#pdfURL#" type="application/pdf" width="600" height="400">
 </object>
</cfoutput>
```

- To display Ajax components, all the URLs used in portlets must be encoded.

```
CFPOD:
<cfset sourceURL =
getPortletResponse().encodeURL(getPortletRequest().getContextPath() & "/<path
to cfm>/expandpath.cfm")>
<cfpod name="pod01" source="#sourceURL#" height="500" width="300"
title="Example CFPod"/>
expandpath:
<cfoutput>#ExpandPath("./")#</cfoutput>
CFWINDOW:
<cfset sourceURL =
getPortletResponse().encodeURL(getPortletRequest().getContextPath() & "/<path
to cfm>/expandpath.cfm")>
<cfwindow title="Test Window" name="myWindow" width="200" height="200"
initShow="true" source="#sourceURL#">
</cfwindow>
```

#back to top

© 2014 Adobe Systems Incorporated. All rights reserved.

JSR-286 Support

ColdFusion 9 also supports JSR-286 specifications. In portlets there are three types of requests: action, event, and render. A portlet first handles an action request, and then an event request, and only after that, it would render any request.
Some of the capabilities of JSR-286 include the following:

© 2014 Adobe Systems Incorporated. All rights reserved.

Publishing and Processing Events

To define an event, you would have to declare it in portlet.xml

```
<event-definition>
<qname xmlns:cf="http://adobe.com/coldfusion/portlet/example">cf:HelloEvent
</qname>
 <value-type>java.lang.String</value-type>
</event-definition>
```

This code defines an event named cf:HelloEvent, where cf refers to the namespace and HelloEvent is the local name. Its type is defined by the <value-type> tag. These event definitions require you to use qname to uniquely identify the event.

Now add the events to specific portlets, which are either going to publish (generate) an event or process (consume) an event. You add this information to portlet.xml as well.
`<supported-publishing-event>` tag is used to publish an event.

© 2014 Adobe Systems Incorporated. All rights reserved.

Publishing an event (Event producer)

```
<portlet>
…
<supported-publishing-event>
<qname xmlns:cf="http://adobe.com/coldfusion/portlet/example">cf:HelloEvent</qname>
</supported-publishing-event>
…
</portlet>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

Processing an event (event consumer)

```
<portlet>
….
<supported-processing-event>
<qname xmlns:cf="http://adobe.com/coldfusion/portlet/example">cf:HelloEvent</qname>
</supported-processing-event>
…..
</portlet>
```

Portlet definition can have both publishing and processing tags. `portlet.xml` file has event definition, publishing event, and processing event tags, as its portlets create and consume events.

© 2014 Adobe Systems Incorporated. All rights reserved.

Initiate events in portlet (CFC)

You publish events in the `processAction()` method of the portlet code by calling `setEvent()` on the `ActionResponse` object. This `setEvent()` method takes two parameters: the QName of the event object and the type of object defined in portlet.xml.

Following is an example of processAction() method.

```
<cffunction name="processAction" returntype="void" access="public" output="false"
hint="Called by the portlet container to allow the portlet to process an action
request.">
  <cfargument name="actionRequest" type="any" required="true" hint="A
javax.portlet.ActionRequest java object">
  <cfargument name="actionResponse" type="any" required="true" hint="A
javax.portlet.ActionResponse java object">
  <cfset super.processAction(arguments.actionRequest, arguments.actionResponse)>
  <!--- send event notification --->
  <cftry>
   <cfset arguments.actionResponse.setEvent("HelloEvent",
request.portlet.parameters.event_value)>
   <cfcatch type="any">
    <cflog file="simple-event-portlet" type="error" text="processAction() threw
exception: #cfcatch.message#">
   </cfcatch>
  </cftry>
</cffunction>
```

Capture events in `processEvent()` method:

```
<cffunction name="processEvent" returntype="void" access="public" output="false"
hint="Called by the portlet container requesting the portlet to process a specific
event.">
  <cfargument name="eventRequest" type="any" required="true" hint="A
javax.portlet.EventRequest java object">
  <cfargument name="eventResponse" type="any" required="true" hint="A
javax.portlet.EventResponse java object">
 <cfset var e = StructNew()>
 <cftry>
  <cfset e.name = arguments.eventRequest.getEvent().getName()>
  <cfset e.value = arguments.eventRequest.getEvent().getValue()>
 <cfif NOT IsDefined("application.EventReceivingPortletEvents")>
  <cfset application.EventReceivingPortletEvents = ArrayNew(1)>
 </cfif>
  <cfset ArrayAppend(application.EventReceivingPortletEvents,e)>
   <cfcatch type="any">

   </cfcatch>
 </cftry>
</cffunction>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

Using filters

Filter definition and mapping in portlet.xml

```
<filter>
 <filter-name>Example ColdFusion Filter</filter-name>
 <filter-class>coldfusion.portlet.ColdFusionPortletFilter</filter-class>
 <lifecycle>RENDER_PHASE</lifecycle>
 <lifecycle>EVENT_PHASE</lifecycle>
 <lifecycle>RESOURCE_PHASE</lifecycle>
 <lifecycle>ACTION_PHASE</lifecycle>
 <init-param>
  <name>cfcName</name>
  <value>portlets.filter.ExampleFilter</value>
 </init-param>
</filter>
```

Add filter mapping, that filter applied to particular portlet.

```
<!-- Applies Example Filter to All Portlets -->
 <filter-mapping>
  <filter-name>Example ColdFusion Filter</filter-name>
  <portlet-name>*</portlet-name>
 </filter-mapping>
```

ExampleFilter.cfc:

The following is the ExampleFilter.cfc mentioned in the portlet.xml.

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cfcomponent extends="CFIDE.portlets.filter.ColdFusionPortletFilter">
  <cffunction name="doRenderFilter" returntype="void">
  <cfargument name="renderRequest">
  <cfargument name="renderResponse">
  <cfargument name="filterChain">
  <cflog file="portlet-filter" type="information" text="doRenderFilter() invoked">
  <!--- call the next filter in the chain --->
  <cfset arguments.filterChain.doFilter(arguments.renderRequest,
arguments.renderResponse)>
 </cffunction>
 <cffunction name="doActionFilter" returntype="void">
  <cfargument name="actionRequest">
  <cfargument name="actionResponse">
  <cfargument name="filterChain">
  <cflog file="portlet-filter" type="information" text="doActionFilter() invoked">
  <!--- call the next filter in the chain --->
  <cfset arguments.filterChain.doFilter(arguments.actionRequest,
arguments.actionResponse)>
 </cffunction>
 <cffunction name="doResourceFilter" returntype="void">
  <cfargument name="resourceRequest">
  <cfargument name="resourceResponse">
  <cfargument name="filterChain">
  <cflog file="portlet-filter" type="information" text="doResourceFilter() invoked">

  <!--- call the next filter in the chain --->
  <cfset arguments.filterChain.doFilter(arguments.resourceRequest,
arguments.resourceResponse)>
 </cffunction>
 <cffunction name="doEventFilter" returntype="void">
  <cfargument name="eventRequest">
  <cfargument name="eventResponse">
  <cfargument name="filterChain">
  <cflog file="portlet-filter" type="information" text="doEventFilter() invoked">
  <!--- call the next filter in the chain --->
  <cfset arguments.filterChain.doFilter(arguments.eventRequest,
arguments.eventResponse)>
 </cffunction>
</cfcomponent>
```

#back to top

© 2014 Adobe Systems Incorporated. All rights reserved.

Working with Documents, Charts, and Reports

To create powerful documents, charts, and reports, Adobe ColdFusion provides interfaces to work with PDF, Adobe Flash, Adobe Connect and extends the integration support to OpenOffice and Microsoft Office application formats such as Excel, PowerPoint, and SharePoint.

- Manipulating PDF Forms in ColdFusion
- Assembling PDF Documents
- Creating and Manipulating ColdFusion Images
- Creating Charts and Graphs
- Creating Reports and Documents for Printing
- Creating Reports with Report Builder
- Creating Slide Presentations
- PDF Generation in ColdFusion 11

#back to top

© 2014 Adobe Systems Incorporated. All rights reserved.

Manipulating PDF Forms in ColdFusion

You can use Adobe ColdFusion to manipulate PDF forms created in Adobe® Acrobat® Professional and Adobe® LiveCycle™ Designer.

© 2014 Adobe Systems Incorporated. All rights reserved.

About PDF forms

Adobe ColdFusion lets you incorporate interactive PDF forms in your application. You can extract data submitted from the PDF forms, populate form fields from an XML data file or a database, and embed PDF forms in PDF documents created in ColdFusion.

ColdFusion supports interactive forms created with Adobe Acrobat forms and with LiveCycle. In Adobe Acrobat 6.0 or earlier, you can create interactive Acroforms. Using Adobe LiveCycle Designer, which is provided with Adobe Acrobat Professional 7.0 and later, you can generate interactive forms.

The type of form is significant because it affects how you manipulate the data in ColdFusion. For example, you cannot use an XML data file generated from a form created in Acrobat to populate a form created in LiveCycle, or the reverse, because the XML file formats differ between the two types of forms.

Forms created in Acrobat use the XML Forms Data Format (XFDF) file format. Forms created in LiveCycle use the XML Forms Architecture (XFA) format introduced in Acrobat and Adobe Reader 6. For examples, see Populating a PDF form with XML data. The file format also affects how you prefill fields in a form from a data source, because you map the data structure as well as the field names. For examples, see Prefilling PDF form fields.

The use of JavaScript also differs based on the context. The JavaScript Object Model in a PDF file differs from the HTML JavaScript Object Model. Consequently, scripts written in HTML JavaScript do not apply to PDF files. Also, JavaScript differs between forms created in Acrobat and those forms created in LiveCycle: scripts written in one format do not work with other.

ColdFusion 9 introduced several tags for manipulating PDF forms:

Tag	Description
`cfpdfform`	Reads data from a form and writes it to a file or populates a form with data from a data source.
`cfpdfformparam`	A child tag of the `cfpdfform` tag or the `cfpdfsubform` tag; populates individual fields in PDF forms.
`cfpdfsubform`	A child tag of the `cfpdfform` tag; creates the hierarchy of the PDF form so that form fields are filled properly. The `cfpdfsubform` tag contains one or more `cfpdpformparam` tags.

The following table describes a few of the tasks that you can perform with PDF forms:

Task	Tags and actions
Populate a PDF form with XML data	`populate` action of the cfpdf tag
Prefill individual fields in a PDF form with data from a data source	`populate` action of the cfpdfform tag with the `cfpdfsubform` and `cfpdfparam` tags
Determine the structure of a PDF form	`read` action of the `cfpdfform` tag with the `cfdump` tag

© 2014 Adobe Systems Incorporated. All rights reserved.

Embed an interactive PDF form within a PDF document	`populate` action of the `cfpdfform` tag within the `cfdocument` tag.Note: The `cfpdfform` tag must be at the same level as the `cfdocumentsection` tags, not contained within them.
Write a PDF form directly to the browser	`populate` action of the `cfpdfform` tag with the `destination` attribute not specified
Write PDF form output to an XML file	`read` action of the `cfpdfform` tag
Print a PDF form from ColdFusion	cfprint tag
Extract data from a PDF form submission	`source="#PDF.Content#"` for the `read` action of the `cfpdfform` tag
Write data extracted from a PDF form submission to a PDF file	`source="#PDF.Content#"` for the `populate` action of the `cfpdfform` tag, and the `destination` attribute
Write data in a form generated in LiveCycle to an XDP file	`source="#PDF.Content#"` for the `populate` action of the `cfpdfform` tag, and an XDP extension for the output file
Extract data from an HTTP post submission	`cfdump` tag determines the structure of the form data; map the form fields to the output fields
Flatten forms generated in Acrobat (not used forms generated in LiveCycle)	cfpdf For more information, see *Flattening forms created in Acrobat* in Using shortcuts for common tasks.
Merge forms generated in Acrobat or LiveCycle with other PDF documents	`cfpdf action="merge"` For more information, see *Merging PDF documents* in Using shortcuts for common tasks.

© 2014 Adobe Systems Incorporated. All rights reserved.

Populating a PDF form with XML data

Some applications submit PDF form data in an XML data file. For example, the e-mail submit option in forms created in LiveCycle generates an XML data file and delivers it as an attachment to the specified e-mail address. This technique is an efficient way to transmit and archive data because XML data files are smaller than PDF files. However, XML files are not user-friendly: to view the file in its original format, the user has to open the PDF form template in Acrobat and import the XML data file.
ColdFusion automates the process of reuniting XML data with the PDF form that generated it. To reunite them, you use the `populate` action of the `cfpdfform` tag, specify the source, which is the PDF form used as a template, and specify the XML data file that contains the information submitted by the person who completed the form. You also have the option to save the result to a new file, which lets you save the completed forms in their original format (and not just the form data). In the following example, ColdFusion populates the payslipTemplate.pdf form with data from the formdata.xml data file and writes the form to a new PDF file called employeeid123.pdf:

```
<cfpdfform source="c:\payslipTemplate.pdf"
destination="c:\empPayslips\employeeid123.pdf" action="populate"
XMLdata="c:\formdata.xml"/>
```

For forms created in LiveCycle, you have the option to write the output to an XML Data Package (XDP) file rather than a PDF file. For more information, see *Writing LiveCycle form output to an XDP file* in Extracting data from a PDF form submission.

Note

If you do not specify a destination, the `populate` action displays the populated PDF form in a browser window.

When you populate a form with an XML data file, ensure that the XML data is in the appropriate format. The format of the XML data file differs based on whether it was generated from Acrobat or LiveCycle. Acrobat generates an XML Forms Data Format (XFDF) file format. The following example shows the XFDF format:

```
<?xml version="1.0" encoding="UTF-8"?>
- <xfdf xmlns="http://ns.adobe.com/xfdf/" xml:space="preserve">
 - <fields>
  - <field name="textname">
    <value>textvalue</value>
   </field>
  - <field name="textname1">
    <value>textvalue1</value>
   </field>
  </fields>
 </xfdf>
```

Forms created in LiveCycle require an XML Forms Architecture (XFA) format. The following example shows an XFA format:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<?xml version="1.0" encoding="UTF-8"?>
- <xfa:data xmlns:xfa="http://www.xfa.org/schema/xfa-data/1.0/">
- <form1>
  <SSN>354325426</SSN>
  <fname>coldfusion</fname>
  <num>354325426.00</num>
 - <Subform1>
   <SSN />
  </Subform1>
</form1>
</xfa>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

Prefilling PDF form fields

ColdFusion lets you prefill individual form fields with data extracted from a data source. For example, you can run a query to extract returning customer information from a data source based on a user name and password and populate the related fields in an order form. The customer can complete the rest of the fields in the form and submit it for processing. To do so, map the field names and the data structure of the PDF form to the fields in the data source.
To determine the structure of the PDF form, use the read action of the `cfpdfform` tag, as the following example shows:

```
<cfpdfform source="c:\forms\timesheet.pdf" result="resultStruct" action="read"/>
```

Then use the `cfdump` tag to display the structure:

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

The result structure for a form created in Acrobat form could look something like the following example:

struct	
firstName	empty string
lastName	empty string
department	empty string
...	...

To prefill the fields in ColdFusion, you add a `cfpdfformparam` tag for each of the fields directly under the `cfpdfform` tag:

```
<cfpdfform action="populate" source="c:\forms\timsheet.PDF">
 <cfpdfformparam name="firstName" value="Boris">
 <cfpdfformparam name="lastName" value="Pasternak">
 <cfpdfformparam name="department" value="Marketing">
...
</cfpdfform>
```

Forms created in LiveCycle from the standard blank forms contain a subform called form1. The result structure of a form created in LiveCycle could look like the following example:

struct		
form1	struct	

© 2014 Adobe Systems Incorporated. All rights reserved.

txtfirstName	empty string
txtlastName	empty string
txtdepartment	empty string
...	...

To prefill the fields in ColdFusion, add a `cfpdfsubform` tag for form1 and a `cfpdfformparam` tag for each of the fields to fill directly below the `cfpdfsubform` tag:

```
<cfpdfform source="c:\forms\timesheetForm.pdf" action="populate">
 <cfpdfsubform name="form1">
  <cfpdfformparam name="txtfirstName" value="Harley">
  <cfpdfformparam name="txtlastName" value="Davidson">
  <cfpdfformparam name="txtDeptName" value="Engineering">
  ...
 </cfpdfsubform>
</cfpdfform>
```

Note

In dynamic forms created in LiveCycle forms (forms saved as Dynamic PDF Form Files in LiveCycle Designer), you have the option to mark how many times a record is repeated. Therefore, if no record exists for a subform, the subform does not appear in the structure returned by the `read` action of the `cfpdfform` tag. You view these forms in LiveCycle Designer to see the hierarchy.

Nesting subforms

Although Acrobat forms do not contain subforms, some contain complex field names. For example, an Acrobat form could contain the following fields: form1.x.f1, form1.x.f2, form1.x.f3, and so on.
Because the `cfpdfparam` tag does not handle field names with periods in them, ColdFusion treats forms with complex field names created in Acrobat the same way as subforms created in LiveCycle. Therefore, the result structure of an Acrobat form with complex field names would look like the following example:

struct	
form1	struct
x	struct
f1	empty string
f2	empty string
...	...

© 2014 Adobe Systems Incorporated. All rights reserved.

In ColdFusion, to prefill the fields in forms created in Acrobat, nest the field names as subforms:

```
<cfpdfform action="populate" source="acrobatForm.pdf">
 <cfpdfsubform name="form1">
  <cfpdfsubform name="x">
   <cfpdfformparam name="f1" value="AGuthrie">
   <cfpdfformparam name="f2" value="123">
   <cfpdfformparam name="f3" value="456">
  </cfpdfsubform>
 </cfpdfsubform>
</cfpdfform>
```

Often, forms created in LiveCycle contain subforms within the form1 subform. For example, the following grant application contains nested subforms:

struct		
form1	struct	
	grantapplication	struct
	page1	struct
	orgAddress	empty string
	orgCity	empty string
	orgState	empty string
	...	...
page2	struct	
	description	empty string
	pageCount	empty string
	...	...

To populate the fields in ColdFusion, map the structure by using nested `cfpdfsubform` tags:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cfpdfform source="c:\grantForm.pdf" destination="c:\employeeid123.pdf"
action="populate">
 <cfpdfsubform name="form1">
  <cfpdfsubform name="grantapplication">
   <cfpdfsubform name="page1">
    <cfpdfformparam name="orgAddress" value="572 Evergreen Terrace">
    <cfpdfformparam name="orgCity" value="Springfield">
    <cfpdfformparam name="orgState" value="Oregon">
    ...
   </cfpdfsubform>
   <cfpdfsubform name="page2">
    <cfpdfformparam name="description" value="Head Start">
    <cfpdfformparam name="pageCount" value="2">
    ...
   </cfpdfsubform>
  </cfpdfsubform>
 </cfpdfsubform>
</cfpdfform>
```

Note

A PDF file can contain only one interactive form. Therefore, if a PDF file contains subforms, a Submit button submits data for all the subforms simultaneously.

© 2014 Adobe Systems Incorporated. All rights reserved.

Embedding a PDF form in a PDF document

You can use the `cfpdfform` tag inside the `cfdocument` tag to embed an existing interactive PDF form in a PDF document. This technique is useful to include additional information with a standard interactive form. For example, a company could have a generic PDF form for maintaining employee information. You could reuse this form in different contexts to ensure that the employee information is current.
To create the static PDF pages, use the `cfdocument` tag and `cfdocumentsection` tags. Then use the `cfpdfform` tag in the `cfdocument` tag to create an interactive form in the PDF document. When the user updates the form and prints or submits it, all of the pages in the document, including the static PDF pages, are printed or submitted with the form.

> **Note**
>
> You can embed only one interactive form in a PDF document; therefore, include only one `cfpdfform` tag in a `cfdocument` tag. However, each `cfpdfform` tag can include multiple `cfpdfsubform` tags and `cfpdfformparam` tags.

Use at least one `cfdocumentsection` tag with the `cfpdfform` tag, but do not place the `cfpdfform` tag within the `cfdocumentsection` tag. Instead, insure that the `cfpdfform` and `cfdocumentsection` tags are at the same level, the following example shows:

```
<cfdocument format="pdf">
 <cfdocumentitem type="header">
 <font size="+1">This is the Header</font>
 </cfdocumentitem>
 <cfdocumentitem type="footer">
 <font size="+1">This is the Footer</font>
 </cfdocumentitem>

 <cfdocumentsection>
 <p>This is the first document section.</p>
 </cfdocumentsection>

 <cfpdfform source="c:\forms\embed.pdf" action="populate">
  <cfpdfsubform name="form1">
   <cfpdfformparam name="txtManagerName" value="Janis Joplin">
   <cfpdfformparam name="txtDepartment" value="Sales">
  </cfpdfsubform>
 </cfpdfform>

 <cfdocumentsection>
 <p>This is another section</p>
 </cfdocumentsection>
</cfdocument>
```

The contents of the `cfpdfform` tag start on a new page. Any text or code directly after the `cfdocument` tag and before the `cfpdfform` tag applies to the document sections but not to the interactive PDF form in the `cfpdfform` tag.
The headers and footers that are part of the embedded PDF form do not apply to the rest of the PDF document, and the headers and footers that are defined in the `cfdocument` tag do not apply to the interactive form. However, header and footer information defined in the `cfdocumentitem` tags resumes in the sections that follow the embedded form and account for the pages in the embedded form.

© 2014 Adobe Systems Incorporated. All rights reserved.

Note

The read action of the `cfpdfform` tag is not valid when you embed a PDF form. Also, you cannot specify a destination in the `cfpdfform` tag. However, you can specify a filename in the `cfdocument` tag to write the PDF document with the PDF form to an output file. If you do not specify a filename, ColdFusion displays the PDF form in the context of the PDF document in the browser.

© 2014 Adobe Systems Incorporated. All rights reserved.

Extracting data from a PDF form submission

Data extraction differs based on how the PDF form is submitted. ColdFusion supports two types of PDF form submission: HTTP post, which submits the form data, but not the form itself, and PDF, which submits the entire PDF file.
One use for PDF submission is for archival purpose: because the form is submitted with the data, you can write the output to a file. HTTP post submissions process faster because only the field data is transmitted, which is useful for updating a database or manipulating specific data collected from the form, but you cannot write an HTTP post submission directly to a file.

> **Note**
>
> Although forms created in LiveCycle Designer allow several types of submission, including XDP and XML, ColdFusion can extract data from HTTP post and PDF submissions only.

In LiveCycle Designer, the XML code for an HTTP post submission looks like the following example:

```
<submit format="formdata" target="http://localhost:8500/pdfforms/pdfreceiver.cfm"
textEncoding="UTF-8"/>
```

In LiveCycle Designer, the XML code for a PDF submission looks like the following example:

```
<submit format="pdf" target="http://localhost:8500/pdfforms/pdfreceiver.cfm"
textEncoding="UTF-16" xdpContent="pdf datasets xfdf"/>
```

> **Note**
>
> Acrobat forms are submitted in binary format, not XML format.

Extracting data from a PDF submission

Use the following code to extract data from a PDF submission and write it to a structure called fields:

```
<!--- The following code reads the submitted PDF file and generates a result
structure called fields. --->
<cfpdfform source="#PDF.content#" action="read" result="fields"/>
```

Use the `cfdump` tag to display the data structure, as follows:

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

© 2014 Adobe Systems Incorporated. All rights reserved.

> **Note**
>
> When you extract data from a PDF submission, always specify `"#PDF.content#"` as the source.

You can set the form fields to a variable, as the following example shows:

```
<cfset empForm="#fields.form1#">
```

Use the populate action of the `cfpdfform` tag to write the output to a file. Specify `"#PDF.content#"` as the source. In the following example, the unique filename is generated from a field on the PDF form:

```
<cfpdfform action="populate" source="#PDF.content#"
 destination="timesheets\#empForm.txtsheet#.pdf" overwrite="yes"/>
```

Writing LiveCycle form output to an XDP file

For Acrobat forms, you can write the output to a PDF file only. For LiveCycle forms, you have the option to write the output to an XDP file. The filename extension determines the file format: to save the output in XDP format, simply use an XDP extension in the destination filename, as the following example shows:

```
<cfpdfform action="populate" source="#PDF.content#"
 destination="timesheets\#empForm.txtsheet#.xdp" overwrite="yes"/>
```

An XDP file is an XML representation of a PDF file. In LiveCycle Designer, an XDP file contains the structure, data, annotations, and other relevant data to LiveCycle forms, which renders the form at run time.
ColdFusion XDP files contain the XDP XML code and the PDF image. Therefore, the file size is larger than a PDF file. Only write PDF forms to XDP files if you must incorporate them into the LiveCycle Designer workflow on a LiveCycle server.

Writing PDF output to an XML file

ColdFusion lets you extract data from a PDF form and write the output to an XML data file. To do so, you must save the form output as a PDF file. (The `cfpdfform` tag source must always be a PDF file.)
To write the output of a PDF file to an XML file, use the `read` action of the `cfpdfform` tag, as the following example shows:

```
<cfpdfform action="read" source="#empForm.txtsheet#.pdf"
 XMLdata="timesheets\#empForm.txtsheet#.xml"/>
```

To save disk space, you can delete the PDF file and maintain the XML data file. As long as you keep the blank PDF form used as the template, you can use the `populate` action to regenerate the PDF file. For more information on populating forms, see Populating a PDF form with XML data.

© 2014 Adobe Systems Incorporated. All rights reserved.

Extracting data from an HTTP post submission

For an HTTP post submission, use the `cfdump` tag with the form name as the variable to display the data structure, as follows:

```
<cfdump var="#FORM.form1#">
```

Note

When you extract data from an HTTP post submission, always specify the form name as the source. For example, specify `"#FORM.form1#"` for a form generated from a standard template in LiveCycle.

Notice that the structure is not necessarily the same as the structure of the PDF file used as the template (before submission). For example, the structure of a form before submission could look like the following example:

struct	
form1	struct
txtDeptName	empty string
txtEMail	empty string
txtEmpID	empty string
txtFirstName	empty string
txtLastName	empty string
txtPhoneNum	empty string

After submission by using HTTP post, the resulting structure would look like the following example:

struct	
FORM1	struct
SUBFORM	struct
HEADER	struct
HTTPSUBMITBUTTON1	empty string
TXTDEPTNAME	Sales

© 2014 Adobe Systems Incorporated. All rights reserved.

TXTFIRSTNAME	Carolynn
TXTLASTNAME	Peterson
TXTPHONENUM	(617) 872-9178
TXTEMPID	1
TXTEMAIL	carolynp@company

Note

When data extraction using the `cfpdfform` tag results in more than one page, instead of returning one structure, the extraction returns one structure per page.

The difference in structure reflects internal rules applied by Acrobat for the HTTP post submission.
To extract the data from the HTTP post submission and update a database with the information, for example, map the database columns to the form fields, as the following code shows:

```
<cfquery name="updateEmpInfo" datasource="cfdocexamples">
UPDATE EMPLOYEES
 SET FIRSTNAME = "#FORM1.SUBFORM.HEADER.TXTFIRSTNAME#",
  LASTNAME = "#FORM1.SUBFORM.HEADER.TXTLASTNAME#",
  DEPARTMENT = "#FORM1.SUBFORM.HEADER.TXTDEPTNAME#",
  IM_ID = "#FORM1.SUBFORM.TXTEMAIL#",
  PHONE = "#FORM1.SUBFORM.HEADER.TXTPHONENUM#"
 WHERE EMP_ID = <cfqueryparam value="#FORM1.SUBFORM.TXTEMPID#">
</cfquery>
```

You can set a variable to create a shortcut to the field names, as the following code shows:

```
<cfset fields=#form1.subform.header#>
```

Use the `cfoutput` tag to display the form data:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<h3>Employee Information</h3>
<cfoutput>
 <table>
  <tr>
   <td>Name:</td>
   <td>#fields.txtfirstname# #fields.txtlastname#</td>
  </tr>
  <tr>
   <td>Department:</td>
   <td>#fields.txtdeptname#</td>
  </tr>
  <tr>
   <td>E-Mail:</td>
   <td>#fields.txtemail#</td>
  <tr>
   <td>Phone:</td>
   <td>#fields.txtphonenum#</td>
  </tr>
 <table>
</cfoutput>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

Application examples that use PDF forms

The following examples show how you can use PDF forms in your applications.

PDF submission example

The following example shows how to populate fields in a PDF form created in LiveCycle Designer based on an employee login information. When the employee completes the form and clicks the PDF Submit button, the entire PDF form with the data is submitted to a second processing page where ColdFusion writes the completed form to a file.
On the ColdFusion login page, an employee enters a user name and password:

```
<!--- The following code creates a simple form for entering a user name and
password.
 The code does not include password verification. --->
<h3>Timesheet Login Form</h3>
<p>Please enter your user name and password.</p>
<cfform name="loginform" action="loginform_proc.cfm" method="post">
<table>
 <tr>
  <td>user name:</td>
  <td><cfinput type="text" name="username" required="yes"
    message="A user name is required."></td>
 </tr>
 <tr>
  <td>password:</td>
  <td><cfinput type="password" name="password" required="yes"
    message="A password is required."></td>
 </tr>
</table>
 <br/>
 <cfinput type="submit" name="submit" value="Submit">
</cfform>
```

On the first processing page, a query retrieves all of the information associated with the user name from the cfdocexamples database. The `cfpdfform` tag populates an associated PDF form created in LiveCycle Designer (called timesheetForm.pdf) with the employee name, phone number, e-mail address, and department. ColdFusion displays the populated form in the browser, where the employee can complete the form and submit it.

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<!--- The following code retrieves all of the employee information for the user name
entered
 on the login page. --->
<cfquery name="getEmpInfo" datasource="cfdocexamples">
 SELECT * FROM EMPLOYEES
 WHERE EMAIL = <cfqueryparam value="#FORM.username#">
</cfquery>

<!---
The following code populates the template called "timesheetForm.pdf" with data from
the query and displays the interactive PDF form in the browser. A field in the PDF
form contains the name of the output file being written. It is a combination of the
user name and the current date.
--->

<!--- Notice the use of the cfpdfsubform tag. Forms created from templates in
LiveCycle Designer include a subform called form1. Use the cfpdfsubform tag to match
the structure of the form in ColdFusion. Likewise, the field names in the
cfpdfformparam tags must match the field names in the PDF form. If the form
structures and field names do not match exactly, ColdFusion does not populate the
form fields. --->

<cfpdfform source="c:\forms\timesheetForm.pdf" action="populate">
 <cfpdfsubform name="form1">
  <cfpdfformparam name="txtEmpName" value="#getEmpInfo.FIRSTNAME#
   #getEmpInfo.LASTNAME#">
  <cfpdfformparam name="txtDeptName" value="#getEmpInfo.DEPARTMENT#">
  <cfpdfformparam name="txtEmail" value="#getEmpInfo.IM_ID#">
  <cfpdfformparam name="txtPhoneNum" value="#getEmpInfo.PHONE#">
  <cfpdfformparam name="txtManagerName" value="Randy Nielsen">
  <cfpdfformparam name="txtSheet"
   value="#form.username#_#DateFormat(Now())#">
 </cfpdfsubform>
</cfpdfform>
```

When the user completes the timesheet form (by filling in the time period, projects, and hours for the week) and clicks the Submit button, Acrobat sends the PDF file in binary format to a second ColdFusion processing page.

Note

In LiveCycle Designer, use the standard Submit button on the PDF form and specify "submit as: PDF" in the button Object Properties. Also, ensure that you enter the URL to the ColdFusion processing page in the Submit to URL field.

The `cfpdfform` tag `read` action reads the PDF content into a result structure named `fields`. The `cfpdfform` tag `populate` action writes the completed form to a file in the timesheets subdirectory.

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<!--- The following code reads the PDF file submitted in binary format and generates
a result structure called fields. The cfpdfform populate action and the cfoutput
tags reference the fields in the structure. --->

<cfpdfform source="#PDF.content#" action="read" result="fields"/>
<cfset empForm="#fields.form1#">
<cfpdfform action="populate" source="#PDF.content#"
destination="timesheets\#empForm.txtsheet#.pdf" overwrite="yes"/>

<h3>Timesheet Completed</h3>
<p><cfoutput>#empForm.txtempname#</cfoutput>,</p>
<p>Thank you for submitting your timesheet for the week of
<cfoutput>#DateFormat(empForm.dtmForPeriodFrom, "long")#</cfoutput> through
<cfoutput>#DateFormat(empForm.dtmForPeriodto, "long")#</cfoutput>. Your manager,
<cfoutput>#empForm.txtManagerName#</cfoutput>, will notify you upon approval.</p>
```

HTTP post example

The following example shows how to extract data from a PDF form submitted with HTTP post and use it to update an employee database. The form was created in LiveCycle Designer.
On the ColdFusion login page, an employee enters a user name and password:

```
<!--- The following code creates a simple form for entering a user name and
password. The code does not include password verification. --->

<h3>Employee Update Login Form</h3>
<p>Please enter your user name and password.</p>
<cfform name="loginform" action="loginform_procHTTP.cfm" method="post">
<table>
 <tr>
  <td>user name:</td>
  <td><cfinput type="text" name="username" required="yes"
    message="A user name is required."></td>
 </tr>
 <tr>
  <td>password:</td>
  <td><cfinput type="password" name="password" required="yes"
    message="A password is required."></td>
 </tr>
</table>
 <br/>
 <cfinput type="submit" name="submit" value="Submit">
</cfform>
```

On the first processing page, a query retrieves all of the information associated with the user name from the cfdocexamples database. The `cfpdfform` tag populates an associated PDF form created in LiveCycle Designer (called employeeInfoHTTP.pdf) with the employee name, phone number, e-mail address, and department. The form also includes the employee ID as a hidden field. ColdFusion displays the populated form in the browser where the employee can change personal information in the form and submit it.

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<!--- The following code retrieves all of the employee information for the user name
entered on the form page. --->
<cfquery name="getEmpInfo" datasource="cfdocexamples">
SELECT * FROM EMPLOYEES
WHERE EMAIL = <cfqueryparam value="#FORM.username#">
</cfquery>

<!--- The following code populates the template called "employeeInfoHTTP.pdf" with
data from the query. As in the previous example, notice the use of the cfpdfsubform
tag. The txtEmpID field is a hidden field on the PDF form. --->

<cfquery name="getEmpInfo" datasource="cfdocexamples">
SELECT * FROM EMPLOYEES
WHERE EMAIL = <cfqueryparam value="#FORM.username#">
</cfquery>

<cfpdfform source="c:\forms\employeeInfoHTTP.pdf" action="populate">
 <cfpdfsubform name="form1">
  <cfpdfformparam name="txtFirstName" value="#getEmpInfo.FIRSTNAME#">
  <cfpdfformparam name="txtLastName" value="#getEmpInfo.LASTNAME#">
  <cfpdfformparam name="txtDeptName" value="#getEmpInfo.DEPARTMENT#">
  <cfpdfformparam name="txtEmail" value="#getEmpInfo.IM_ID#">
  <cfpdfformparam name="txtPhoneNum" value="#getEmpInfo.PHONE#">
  <cfpdfformparam name="txtEmpID" value="#getEmpInfo.Emp_ID#">
 </cfpdfsubform>
</cfpdfform>
```

When the employee updates the information in the form and clicks the HTTP post Submit button, Acrobat sends the form data (but not the form itself) to a second ColdFusion processing page.

Note

In LiveCycle Designer, use the HTTP Submit button on the PDF form. Also, ensure that you enter the URL to the ColdFusion processing page in the URL field of button Object Properties.

Reproduce the structure, not just the field name, when you reference form data. To determine the structure of the form data, use the `cfdump` tag.

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<!--- The following code reads the form data from the PDF form and uses it to update

 corresponding fields in the database. --->

<cfquery name="updateEmpInfo" datasource="cfdocexamples">
UPDATE EMPLOYEES
 SET FIRSTNAME = "#FORM1.SUBFORM.HEADER.TXTFIRSTNAME#",
  LASTNAME = "#FORM1.SUBFORM.HEADER.TXTLASTNAME#",
  DEPARTMENT = "#FORM1.SUBFORM.HEADER.TXTDEPTNAME#",
  IM_ID = "#FORM1.SUBFORM.HEADER.TXTEMAIL#",
  PHONE = "#FORM1.SUBFORM.HEADER.TXTPHONENUM#"
 WHERE EMP_ID = <cfqueryparam value="#FORM1.SUBFORM.TXTEMPID#">
</cfquery>
<h3>Employee Information Updated</h3>
<p><cfoutput>#FORM1.SUBFORM.HEADER.TXTFIRSTNAME#</cfoutput>,</p>
<p>Thank you for updating your employee information in the employee database.</p>
```

Embedded PDF form example

The following example shows how to embed an interactive PDF form in a PDF document created with the `cfdocument` tag.
On the login page, an employee enters a user name and password:

```
<h3>Employee Login Form</h3>
<p>Please enter your user name and password.</p>
<cfform name="loginform" action="embed2.cfm" method="post">
 <table>
  <tr>
   <td>user name:</td>
   <td><cfinput type="text" name="username" required="yes"
    message="A user name is required."></td>
  </tr>
  <tr>
   <td>password:</td>
   <td><cfinput type="password" name="password" required="yes"
    message="A password is required."></td>
  </tr>
 </table>
 <br/>
 <cfinput type="submit" name="submit" value="Submit">
</cfform>
```

On the processing page, a query populates an interactive PDF form from the cfdocexamples database. The interactive PDF form is embedded in a PDF document created with the `cfdocument` tag. The PDF document comprises three sections: the `cfdocumentsection` tags define the first and last sections of the document; the `cfpdfform` tag defines the second section embedded in the PDF document. Each section starts a new page in the PDF document. The Print button on the PDF form prints the entire document, including the pages in the sections before and after the interactive PDF form.

```
<cfquery name="getEmpInfo" datasource="cfdocexamples">
```

© 2014 Adobe Systems Incorporated. All rights reserved.

```
SELECT * FROM EMPLOYEES
WHERE EMAIL = <cfqueryparam value="#FORM.username#">
</cfquery>

<!--- The following code creates a PDF document with headers and footers.
--->
<cfdocument format="pdf">
 <cfdocumentitem type="header">
 <font size="-1" align="center"><i>Nondisclosure Agreement</i></font>
 </cfdocumentitem>
 <cfdocumentitem type="footer">
 <font size="-1"><i>Page <cfoutput>#cfdocument.currentpagenumber#
of#cfdocument.totalpagecount#</cfoutput></i></font>
 </cfdocumentitem>

<!--- The following code creates the first section in the PDF document. --->
 <cfdocumentsection>
 <h3>Employee Nondisclosure Agreement</h3>
 <p>Please verify the information in the enclosed form. Make any of the necessary
changes
 in the online form and click the <b>Print</b> button. Sign and date the last page.
Staple
 the pages together and return the completed form to your manager.</p>
 </cfdocumentsection>

<!--- The following code embeds an interactive PDF form within the PDF document with
fields
 populated by the database query. The cfpdpfform tag automatically creates a section
in
 the PDF document. Do not embed the cfpdfform within cfdocumentsection tags. --->

 <cfpdfform action="populate" source="c:\forms\embed.pdf">
  <cfpdfsubform name="form1">
   <cfpdfformparam name="txtEmpName"
    value="#getEmpInfo.FIRSTNAME# #getEmpInfo.LASTNAME#">
   <cfpdfformparam name="txtDeptName" value="#getEmpInfo.DEPARTMENT#">
   <cfpdfformparam name="txtEmail" value="#getEmpInfo.IM_ID#">
   <cfpdfformparam name="txtPhoneNum" value="#getEmpInfo.PHONE#">
   <cfpdfformparam name="txtManagerName" value="Randy Nielsen">
  </cfpdfsubform>
 </cfpdfform>

<!--- The following code creates the last document section. Page numbering resumes
in this  section. --->
 <cfdocumentsection>
 <p>I, <cfoutput>#getEmpInfo.FIRSTNAME# #getEmpInfo.LASTNAME#</cfoutput>, hereby
attest
 that the information in this document is accurate and complete.</p>
 <br/><br/>
 <table border="0" cellpadding="20">
  <tr><td width="300">
  <hr/>
  <p><i>Signature</i></p></td>
  <td width="150">
  <hr/>
  <p><i>Today's Date</i></p></td></tr>
 </table>
 </cfdocumentsection>
</cfdocument>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

Update PDF form example

The following example shows how ColdFusion lets you update a PDF form while retaining existing data. The application lets a user create an office supply request from a blank form created in LiveCycle or modify an existing supply request. The user has the option to submit the completed form as an e-mail attachment.

```
<!--- supplyReq1.cfm --->
<!--- The following code prefills fields in a blank form in LiveCycle and writes the
prefilled
 form to a new file called NewRequest.pdf in the supplyReqs directory. --->
<cfpdfform source="SupplyReq.pdf" action="populate"
destination="supplyReqs/NewRequest.pdf"
 overwrite="yes">
 <cfpdfsubform name="form1">
  <cfpdfformparam name="txtContactName" value="Constance Gardner">
  <cfpdfformparam name="txtCompanyName" value="Wild Ride Systems">
  <cfpdfformparam name="txtAddress" value="18 Melrose Place">
  <cfpdfformparam name="txtPhone" value="310-654-3298">
  <cfpdfformparam name="txtCity" value="Hollywood">
  <cfpdfformparam name="txtStateProv" value="CA">
  <cfpdfformparam name="txtZipCode" value="90210">
 </cfpdfsubform>
</cfpdfform>

<!--- The following code lets users choose an existing supply request form or create
a new
 request from the NewRequest.pdf form. --->
<h3>Office Supply Request Form</h3>
<p>Please choose the office supply request form that you would like to open. Choose
<b>New
 Supply Request</b> to create a request.</p>

<!--- The following code populates a list box in a form with files located in the
specified
 directory. --->
<cfset thisDir = expandPath(".")>
<cfdirectory directory="#thisDir#/supplyReqs" action="list" name="supplyReqs">

<cfif #supplyReqs.name# is "NewRequest.pdf">
 <cfset #supplyReqs.name# = "---New Supply Request---">
</cfif>

<cfform name="fileList" action="supplyReq2.cfm" method="post">
 <cfselect name="file" query="supplyReqs" value="name" display="name"
  required="yes" size="8" multiple="no"/><br/>
 <cfinput type="submit" name="submit" value="OK">
</cfform>

<!--- supplyReq2.cfm --->
<!--- The following code displays the PDF form that the user selected. --->
<cfif #form.file# is "---New Supply Request---">
 <cfset #form.file# = "NewRequest.pdf">
</cfif>
<cfpdfform source="supplyReqs/#form.file#" action="populate"/>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<!--- supplyReq3.cfm --->
<!--- The following code reads the PDF file content from the submitted PDF form.
--->
<cfpdfform source="#PDF.content#" action="read" result="fields"/>

<!--- The following code writes the PDF form to a file and overwrites the file if it
exists. --->
<cfpdfform action="populate" source="#PDF.content#"
destination="SupplyReqs/supplyReq_#fields.form1.txtRequestNum#.pdf"
overwrite="yes"/>

<!--- The following code customizes the display based on field values extracted from
the PDF
 form. --->
<p><cfoutput>#fields.form1.txtRequester#</cfoutput>,</p>
<p>Your changes have been recorded for supply request
<cfoutput>#fields.form1.txtRequestNum#</cfoutput>.</p>
<p>If the form is complete and you would like to submit it to
<cfoutput>#fields.form1.txtContactName#</cfoutput> for processing, click
<b>Submit</b>.

<!--- The following code gives the option to e-mail the submitted form as an
attachment or
 return to the home page. --->
<cfform name="send" method="post" action="supplyReq4.cfm">
 <cfinput type="hidden"
value="SupplyReqs/supplyReq_#fields.form1.txtRequestNum#.pdf" name="request">
 <cfinput type="hidden" value="#fields.form1.txtRequester#" name="requester">
 <cfinput type="submit" value="Submit" name="Submit">
</cfform>
<p>If you would like to modify your request or choose another request,
<a href="supplyReq1.cfm">click here</a>.</p>

<!--- supplyReq4.cfm --->
<!--- The following code sends the completed PDF form as an attachment to the person

 responsible for processing the form. --->
<p>Your request has been submitted.</p>
<cfmail from="#form.requester#@wildride.com" to="cgardener@wildride.com"
 subject="see attachment">
 Please review the attached PDF supply request form.
```

© 2014 Adobe Systems Incorporated. All rights reserved.

```
 <cfmailparam file="#form.request#">
</cfmail>
```

#back to top

© 2014 Adobe Systems Incorporated. All rights reserved.

Assembling PDF Documents

You can use Adobe ColdFusion to assemble PDF documents. You create a unified document from multiple source files or pages from multiple files by using the `cfpdf` and `cfpdfparam` tags.

© 2014 Adobe Systems Incorporated. All rights reserved.

About assembling PDF documents

You use the `cfpdf` tag to assemble PDF documents in Adobe ColdFusion. The tag provides several actions for creating unified output files from multiple sources, as the following table shows:

Action	Description
`addWatermark`	Adds a watermark image to one or more pages in a PDF document.
`deletePages`	Deletes one or more pages from a PDF document.
`addheader`	Adds header to a PDF document.
`addfooter`	Adds footer to a PDF document.
`removeheaderfooter`	Removes header and footer from a PDF document.
`optimize`	Reduces the quality of PDF documents by image downsampling and removing unused objects.
`extracttext`	Extracts text from the specified pages or the entire PDF documents
`extractimage`	Extracts images from the specified pages or the entire PDF document
`transform`	Performs page level transformations
`getInfo`	Extracts information associated with the PDF document, such as the author, title, and creation date.
`merge`	Assembles PDF documents or pages from PDF source files into one output file.
`processddx`	Extends the `cfpdf` tag by providing a subset of Adobe® LiveCycle™ Assembler functionality. This action is the default.
`protect`	Password-protects and encrypts a PDF document.
`read`	Reads a PDF document into a ColdFusion variable.
`removeWatermark`	Removes watermarks from specified pages in a PDF document.
`setInfo`	Sets the Title, Subject, Author, and Keywords for a PDF document,

© 2014 Adobe Systems Incorporated. All rights reserved.

`thumbnail`	Generates thumbnail images from specified pages in a PDF document.
`write`	Writes PDF output to a file. Also use to flatten forms created in Acrobat and linearize documents.

Note

You cannot use the `cfpdf` tag to create a PDF document from scratch. To create a PDF document from HTML content, use the `cfdocument` tag. Also, you can use Report Builder to generate a report as a PDF document. Instead of writing a PDF document to file, you can specify a PDF variable generated as the source for the `cfpdf` tag.

All but one of the `cfpdf` tag actions provide shortcuts to common tasks; for example, with one line of code, you can add a watermark image to one or more pages in an output file, merge all the PDF documents in a directory into a single output file, or password-protect a PDF document. ColdFusion provides two ways to extend the functionality of the `cfpdf` tag: the `cfpdfparam` tag and the `processddx` action.
You use the `cfpdfparam` tag only with the `merge` action of the `cfpdf` tag. The `cfpdfparam` tag gives you more control over which files are included in the output file; for example you can merge pages from multiple files in different directories.
The `processddx` action extends the `cfpdf` tag by providing a subset of Adobe LiveCycle Assembler functionality. You use the `processddx` action to process Document Description XML (DDX) instructions explained in Using DDX to perform advanced tasks. Using DDX instructions requires more coding, but it lets you perform complex tasks, such as generating a table of contents and adding automatic page numbers.
Also, ColdFusion provides three functions for PDF file, DDX file, and PDF variable verification:

Function	Description
IsDDX	Determines whether a DDX file, path, and instructions are not null and are valid. Also verifies that ColdFusion supports the schema used for the DDX instructions.
IsPDFFile	Determines whether a PDF source file, path, and version are valid and supported on the server running ColdFusion. Also verifies whether a PDF file is corrupted.
IsPDFObject	Determines whether a PDF object stored in memory is valid. Also verifies the contents of PDF variables generated by the `cfdocument` and `cfpdf` tags.

The following table describes a few document assembly tasks that you can perform with ColdFusion:

Task	Action
Add a generated table of contents to a PDF document	`cfpdf action="processddx"` with the `TableOfContents` DDX element{{cfpdf action="extracttext"}} can also be used.

© 2014 Adobe Systems Incorporated. All rights reserved.

Add automatic page numbers to a PDF document	`cfpdf action="processddx"` with the `_PageNumber` and `_LastPagenumber` built-in keys. Valid only in the `Header` and `Footer` DDX elements.
Add headers and footers to a PDF document	`cfpdf action="processddx" with` the `Header` and `Footer` DDX elementsor{{cfpdf action="addheader"}} and `cfpdf action="addfooter"`
Add or remove watermarks	`cfpdf action="processddx"` with the `Watermark` and `Background` DDX elements{{cfpdf action="addWatermark"}} and `cfpdf action="removeWatermark"`
Change the encryption algorithm for PDF documents	`cfpdf action="protect" encrypt="`*`encryption algorithm`*`"`
Change user permissions on a PDF document	`cfpdf action="protect" newOwnerPassword="`*`xxxxx`*`"permissions="`*`comma-separated list`*`"`
Delete pages from a PDF document	`cfpdf action="deletePages"`
Extract text from a PDF document and export it to an XML file	`cfpdf action="processddx"` with the `DocumentText` DDX element
Flatten (remove interactivity from) forms created in Acrobat	`cfpdf action="write" flatten="yes"`
Generate thumbnail images from PDF document pages	`cfpdf action="thumbnail"pages="}}page numbers{{"`
Linearize PDF documents for faster web display	`cfpdf action="write" saveOption="linear"`
Merge pages and page ranges from multiple documents in different locations into one PDF document	`cfpdf action="merge"` with multiple `cfpdfparam` tags
Merge PDF documents in a directory into one PDF document	`cfpdf action="merge" directory="`*`path`*`"`
Password-protect PDF documents	`cfpdf action="protect" newUserPassword="`*`xxxx`*`"`
Set the initial view for a PDF document	`cfpdf action="processddx"` with the `InitialViewProfile` DDX element
Create different versions of a PDF document	`Duplicate` function to clone PDF variables

© 2014 Adobe Systems Incorporated. All rights reserved.

Using shortcuts for common tasks

You use the `cfpdf` tag actions to perform shortcuts to common PDF document assembly and manipulation.

Adding and removing watermark images

Use the `addWatermark` and `removeWatermark` actions to add and remove watermarks from PDF documents. You can create a watermark and apply it to a PDF document in one of the following ways:

- Use an image file as a watermark.
- Specify a variable that contains an image file.
- Specify a ColdFusion image.
- Use the first page of a PDF document as a watermark.

> **Note**
>
> You can also use the `Watermark` or `Background` DDX elements with the `processddx` action to create a text-string watermark. For more information, see Using DDX to perform advanced tasks.

Using an image file as a watermark

The following example shows how to specify an image file as a watermark:

```
<cfpdf action="addWatermark" source="artBook.pdf"
 image="../cfdocs/images/artgallery/raquel05.jpg" destination="output.pdf"
 overwrite="yes">
```

By default, ColdFusion centers the image on the page, sets the opacity of the image to 3 out of 10 (opaque), and displays the image in the background of each page in the output file. In the following example, ColdFusion displays the watermark in the foreground, offset 100 pixels from the left margin of the page and 100 pixels from the bottom margin of the page. Because the opacity is set to 1, the image does not obscure the page content.

```
<cfpdf action="addWatermark" source="artBook.pdf"
 image="../cfdocs/images/artgallery/raquel05.jpg" destination="output.pdf"
 overwrite="yes" foreground="yes" opacity=1 showOnPrint="no" position="100,100">
```

For a complete list of attributes and settings, see the cfpdf tag in the *CFML Reference*.
With the ColdFusion 9 release, the `addWatermark` action now supports the `rgb` and `argb` formats also. The following example shows that if you set the parameters for a new image to rgb or argb and then use the cfpdf action=addwatermark, ColdFusion allows this action:

```
<!---setting the argb format for myImage--->
<cfset myImage = ImageNew("",200,200,"argb","gray")>

<!---adding watermark for myImage--->
<cfpdf action="addwatermark" rotation="45" foreground="true" image="#myImage#"
source="RemoveArts.pdf" destination="dest.pdf"  overwrite="yes">
```

© 2014 Adobe Systems Incorporated. All rights reserved.

Using a variable that contains an image file

You can specify a variable that contains an image as a watermark. The following example shows how to create a form from which the user can select an image:

```
<!--- The following code creates a form where you can choose an image to use
 as a watermark. --->
<h3>Choosing a Watermark</h3>
<p>Please choose the image you would like to use as a watermark.</p>

<!--- Create the ColdFusion form to select an image. --->
<table>
<cfform action="addWatermark2.cfm" method="post"
enctype="multipart/form-data">
 <tr>
  <td><img src="../cfdocs/images/artgallery/maxwell01.jpg"/><br/>
  <cfinput type="radio" name="art" value="../cfdocs/images/artgallery/maxwell01.jpg"

   checked="yes">
  Birch Forest</td>
  <td><img src="../cfdocs/images/artgallery/raquel05.jpg"/><br/>
  <cfinput type="radio" name="art" value="../cfdocs/images/artgallery/raquel05.jpg">

   Lounging Woman</td>
  <td><img src="../cfdocs/images/artgallery/jeff01.jpg"/><br/>
  <cfinput type="radio" name="art"
   value="../cfdocs/images/artgallery/jeff01.jpg">Celebration</td>
  <td><img src="../cfdocs/images/artgallery/paul01.jpg"/><br/>
  <cfinput type="radio" name="art"
   value="../cfdocs/images/artgallery/paul01.jpg">Guitarist
  </td>
 </tr>
</table>
<br/>
<cfinput type="Submit" name="submit" value="Submit"></p>
</cfform>
```

The processing page uses the image selected from the form as the watermark for a PDF file:

```
<!--- ColdFusion applies the image selected from the form as the watermark in a PDF
document
 by using the input variable form.art. --->
<cfpdf action="addwatermark" source="check.pdf" image="#form.art#"
destination="output.pdf"
 foreground="yes" overwrite="true">
<p>The watermark has been added to your personalized checks.</p>
```

Using a ColdFusion image as a watermark

You can specify a ColdFusion image as a watermark. You can extract an image from a database and manipulate the image in memory, but you don't have to write the manipulated image to a file. Instead, you can apply the manipulated image as a watermark in a PDF document.

© 2014 Adobe Systems Incorporated. All rights reserved.

In the following example, the first ColdFusion page extracts images from a database and populates a pop-up menu with the titles of the artwork:

```
<!--- Create a query to extract artwork from the cfartgallery database. --->
<cfquery name="artwork" datasource="cfartgallery">
SELECT ARTID, ARTNAME, LARGEIMAGE
FROM ART
ORDER BY ARTNAME
</cfquery>

<!--- Create a form that lists the artwork titles generated by the query. Set the
value to
 LARGEIMAGE so that the image file is passed to the processing page. --->
<cfform action="addWatermarkB.cfm" method="post">
<p>Please choose a title:</p>
<cfselect name="art" query="artwork" display="ARTNAME" value="LARGEIMAGE"
required="yes"
 multiple="no" size="8">
</cfselect>
<br/>
<cfinput type="submit" name="submit" value="OK">
</cfform>
```

The action page generates a ColdFusion image from the selected file by using the `cfimage` tag. The `ImageScaleToFit` function resizes the image and applies the bicubic interpolation method to improve the resolution. To use the manipulated image as a watermark, specify the image variable, as the following example shows:

```
<!--- Verify that an image file exists and is in a valid format. --->
<cfif IsImageFile("../cfdocs/images/artgallery/#form.art#")>
<!--- Use the cfimage tag to create a ColdFusion image from the file chosen from the
list. --->
<cfimage source="../cfdocs/images/artgallery/#form.art#" action="read"
name="myWatermark">

<!--- Use the ImageScaleToFit function to resize the image by using the bicubic
interpolation
 method for better resolution. --->
<cfset ImageScaleToFit(myWatermark,450,450,"bicubic")>

<!--- Use the ColdFusion image variable as the watermark in a PDF document. --->
<cfpdf action="addWatermark" source="title.pdf" image="#myWatermark#"
 destination="watermarkTitle.pdf" overwrite="yes">
<cfelse>
 <p>I'm sorry, no image exists for that title. Please click the Back button and try
  again.</p>
</cfif>
```

For more information on ColdFusion images, see Creating and Manipulating ColdFusion Images.

Creating a text image and using it as a watermark

You can use the `ImageDrawText` function to create a text image in ColdFusion and apply the image as a

© 2014 Adobe Systems Incorporated. All rights reserved.

watermark, as the following example shows:

```
<!--- Create a blank image that is 500 pixels square. --->
<cfset myImage=ImageNew("",500,500)>
<!--- Set the background color for the image to white. --->
<cfset ImageSetBackgroundColor(myImage,"white")>
<!---Clear the rectangle specified on myImage and apply the background color. --->
<cfset ImageClearRect(myImage,0,0,500,500)>
<!--- Turn on antialiasing. --->
<cfset ImageSetAntialiasing(myImage)>

<!--- Draw the text. --->
<cfset attr=StructNew()>
<cfset attr.size=50>
<cfset attr.style="bold">
<cfset attr.font="Verdana">
<cfset ImageSetDrawingColor(myImage,"blue")>
<cfset ImageDrawText(myImage,"PROOF",100,250,attr)>

<!--- Write the text image to a file. --->
<cfimage action="write" source="#myImage#" destination="text.tiff" overwrite ="yes">


<!--- Use the text image as a watermark in the PDF document. --->
<cfpdf action="addwatermark" source="c:/book/1.pdf" image="text.tiff"
 destination="watermarked.pdf" overwrite="yes">
```

For more information on ColdFusion images, see Creating and Manipulating ColdFusion Images. For an example of using DDX elements to create a text-string watermark, see *Adding text-string watermarks* in Using DDX to perform advanced tasks.

Using a PDF page as a watermark

Use the `copyFrom` attribute to create a watermark from the first page of a PDF file and apply it to another PDF document. In the following example, ColdFusion creates a watermark from the first page of image.PDF, applies the watermark to the second page of artBook.pdf, and writes the output to a new file called output.pdf:

```
<cfpdf action="addWatermark" copyFrom="image.pdf" source="artBook.pdf" pages="2"
 destination="output.pdf" overwrite="yes">
```

In this example, image.pdf appears in the background of the second page of artBook.pdf. ColdFusion does not change the size of the watermark image to fit the page. The page used as a watermark can contain text, graphics, or both.

Removing watermarks

Use the `removeWatermark` action to remove a watermark from one or more pages in a PDF document. The following example shows how to remove a watermark from the entire PDF document and write the document to a new output file:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cfpdf action="removeWatermark" source="artBook.pdf" destination="noWatermark.pdf">
```

The following example shows how to remove a watermark from the first two pages of a PDF document and overwrite the source document:

```
<cfpdf action="removeWatermark" source="artBook.pdf" destination="artBook.pdf"
 overwrite="yes" pages="1-2">
```

Because the source and the destination are the same and the `overwrite` attribute is set to `yes`, ColdFusion overwrites the source file with the output file.

Optimizing PDF documents

To optimize a PDF document you can reduce the quality of the document. To reduce the quality of a PDF document you can either downsample the images or remove unused objects from the document.
To downsample images the `algos` attribute is used with values `bilinear`, `bicubic`, and `nearest_neighbour`. The following code snippet generates a PDF after image downsampling:

```
<cfpdf action = "optimize" algo = "bicubic" source = "..\myBook.pdf" name =
#myBook#>
```

To discard unused objects such as comments, JavaScripts, attachments, bookmarks, and metadata from your PDF document using the following attributes with `optimize` action:

```
<cfpdf action = "optimize"
 noJavaScript
 noThumbnails
 noBookmarks
 noComments
 noMetadata
 noFileAttachments
 noLinks
 nofonts>
```

Optimizing page count using encodeall

The new `encodeall` attribute encodes all the unencoded streams in the source. However, it does not discriminate between dumb encodings like `LZW` and encodings like `flate`, so only unencoded streams get `flate` encoded. Example:

```
<cfpdf action=write  source="./inputFiles/Source.pdf"
destination="./outputFiles/Output.pdf" encodeAll="yes">
```

© 2014 Adobe Systems Incorporated. All rights reserved.

Adding and deleting headers and footers from a PDF

You can add a header and footer to a PDF document using the `addheader` and `addfooter` actions, as shown in the following snippet:

```
<!---addfooter--->
<cfpdf action = "addfooter"
source = "../myBook.pdf"
destination = "../myBookwithfooter.pdf"
image = "adobelogo.JPG" // Use this attribute to add an image in the footer
align = "right"> // By default, the alignemnt is center

<!---addheader--->
 <cfpdf action = "addheader"
 source = "../myBook.pdf"
 destination = "../myBookwithheader.pdf"
 text = "Adobe"
 align = "left">
```

Specify the source where the PDF document is located and the destination where the new PDF document with the header and footer will be saved.
You can also specify an image or text that you need to insert in the footer along with various other attributes such as `align`, `bottommargin`, `leftmargin`, `numberformat`, and `opacity`.
To remove header and footer from a PDF document, use the removeheaderfooter action, as shown in the following snippet:

```
<cfpdf action = "removeheaderfooter" source="..\mybook.pdf" destination = "new.pdf">
```

Use this action to remove the header and footer from a PDF document or from specified pages in a document.

Extracting images and text

You can extract text and images from a PDF document using the `extracttext` and `extractimage` actions.
The `extracttext` action extracts all words from the specified page numbers in the PDF document, as shown in the following code snippet:

```
<cfpdf action = "extracttext" source = "../myBook.pdf" pages = "5-20, 29, 80"
destination ="../adobe/textdoc.txt"
```

The `extractimage` action extracts all images from the specified page number in a PDF document, as shown in the following code snippet:

```
<cfpdf action = "extractimage" source = "../myBook.pdf" pages = "1-200" destination
= "..\mybookimages" imageprefix = "mybook">
```

The images are extracted and saved in the directory that you specify in the destination attribute. You can specify a

© 2014 Adobe Systems Incorporated. All rights reserved.

prefix for the images (imageprefix) being extracted, otherwise the system prefixes the image name similar to "cf+page number". To save the images in a specific format, use the format attribute.

Performing page level transformations

You can scale a page, specify the position, and rotation values for pages in a PDF document using the `transform` action. This action has four attributes that define the size (hscale, vscale), position(position), and rotation (rotation) of a page. The following code snippet shows the usage. The value for rotation needs to be in steps (0, 90, 180, 270). If you specify any other value, the system generates an error.

```
<cfpdf action = "transform"
source = "..\myBook.pdf"
destination = "..\new\myBook.pdf">
hscale = ".5"
vscale = ".15"
position = "8, 10"
rotation = "180">
```

Deleting pages from a PDF document

Use the `deletepages` action to remove pages from a PDF document and write the result to a file. You can specify a single page, a page range (for example, "81-97"), or a comma-separated list of pages to delete, as the following example shows:

```
<cfpdf action="deletePages" source="myBook.pdf" pages="10-15,21,89"
 destination="abridged.pdf" overwrite="yes">
```

Protecting PDF files

Use the `protect` action to password-protect, set permissions, and encrypt PDF documents for security.

Setting passwords

ColdFusion supports two types of passwords: an *owner password* and a *user password*. An owner password controls the ability to change the permissions on a document. When you specify an owner password, you set permissions to restrict the operations users can perform, such as the ability to print a document, change its content, and extract content. The following code creates an owner password for a document:

```
<cfpdf action="protect" newOwnerPassword="splunge" source="timesheet.pdf"
 destination="timesheet.pdf" overwrite="yes" permissions="AllowPrinting">
```

To password-protect a document, set the user password. A user password controls the ability to open a document. If you set a user password for a document, any person attempting to open the file is prompted to enter a password. The following example sets the user password for a document:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cfpdf action="protect" newUserPassword="openSesame" source="timesheet.pdf"
 destination="myTimesheet.pdf">
```

In the previous example, no restrictions apply to the PDF document after the user enters the correct password. To restrict usage and password-protect a document, add a user password and an owner password. Use the owner password to set the permissions, as the following example shows:

```
<cfpdf action="protect" newUserPassword="openSesame" newOwnerPassword="topSecret"
 source="timesheet.pdf" destination="myTimesheet.pdf" overwrite="yes"
 permissions="AllowPrinting>
```

In the previous example, a person who enters the user password (openSesame) can print the document only. A person who enters the owner password (topSecret) is considered the owner of the document, has full access to the file, and can change the user permissions for that file.

Setting permissions on a PDF document

To set permissions on a PDF document, specify a `newOwnerPassword`. Conversely, you cannot set the `newOwner Password` without also setting the `permissions` attribute. Only an owner can change permissions or add passwords. For a list of permissions that an owner can set for PDF documents, see cfpdf in the *CFML Reference*. Except for `all` or `none`, owners can specify a comma-separated list of permissions on a document, as the following example shows:

```
<cfpdf action="protect"
permissions="AllowinPrinting,AllowDegradedPrinting,AllowSecure"
 source="timesheet.pdf" newOwnerPassword="private" newUserPassword="openSesame"
 destination="myTimesheet.pdf">
```

In this example, a user must enter the password openSesame before opening the PDF form. All users can print the document at any resolution, but only the owner can modify the document or change the permissions.

Encrypting PDF files

When you specify the `protect` action for a PDF file, ColdFusion encrypts the file with the RC4 128-bit algorithm by default. Depending on the version of Acrobat running on the ColdFusion server, you can set the encryption to protect the document contents and prevent search engines from accessing the PDF file metadata.
You can change the encryption algorithm by using the `encrypt` attribute. For a list of supported encryption algorithms, see cfpdf in the *CFML Reference*.
The following example changes the password encryption algorithm to RC4 40-bit encryption:

```
<cfpdf action="protect" source="confidential.pdf" destination="confidential.pdf"
 overwrite="yes" newOwnerPassword="paSSword1" newUserPassword="openSesame"
 encrypt="RC4_40">
```

© 2014 Adobe Systems Incorporated. All rights reserved.

To prevent ColdFusion from encrypting the PDF document, set the encryption algorithm to `none`, as the following example shows:

```
<cfpdf action="protect" source="confidential.pdf" encrypt="none"
destination="public.pdf">
```

To decrypt a file, provide the owner or user password and write the output to another file. The following code decrypts the confidential.pdf file and writes it to a new file called myDocument.pdf:

```
<cfpdf action="write" source="confidential.pdf" password="paSSword1"
 destination="myDocument.pdf">
```

Managing PDF document information

To retrieve information stored with a source PDF document, such as the creation date, the application used to create the PDF document, and the name of the person who created the document, use the `getInfo` action. For a list of data elements, see PDF file information elements in the *CFML Reference*.

Use the `setInfo` action to specify information, such as the author, subject, title, and keywords associated with the output file. This information is useful for archiving and searching PDF documents. PDF document information is not displayed or printed with the document.

The following example shows how to set keywords for tax documents. The information is useful for assembling the documents based on the tax filing requirements for different business types (Sole Proprietor, Partnership, and S Corporation). Some business types share the same forms and documents. By setting the business type keywords for each document, you can store the documents in one directory and search them based on keyword values. The following code sets three keywords for the p535.pdf tax booklet:

```
<cfset taxKeys=StructNew()>
<cfset taxKeys.keywords="Sole Proprietor,Partnership,S Corporation">
<cfpdf action="setInfo" source="taxes\p535.pdf" info="#taxKeys#"
 destination="taxes\p535.pdf" overwrite="yes">
```

When you use the `setInfo` action, ColdFusion overwrites any existing information for that key-value pair. In the previous example, if the pc535.pdf document contained a keyword of "tax reference", ColdFusion overwrites that keyword with "Sole Proprietor, Partnership, S Corporation".

To retrieve all of the information associated with the tax file, use the `cfdump` tag with the `getInfo` action, as the following example shows:

```
<cfpdf action="getInfo" source="taxes\p535.pdf" name="taxInfo">
<cfdump var="#taxInfo#">
```

To retrieve just the keywords for the PDF document, use this code:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cfpdf action="getInfo" source="taxes\p535.pdf" name="taxInfo">
<cfoutput>#taxInfo.keywords#</cfoutput>
```

Using the name attribute with write action

You can now use the name attribute with <cfpdf action = "write"> and specify a variable name for your PDF document. For example:

```
<cfpdf action="write" source="myBook" name=#myBook# version="1.4">
```

This feature is available with both `cfpdf` and `cfpdfform` tags.

Merging PDF documents

ColdFusion lets you merge PDF documents in the following ways:

- Merge all of the PDF files in a specified directory.
- Merge a comma-separated list of PDF files.
- Merge individual PDF files, and pages within those files, explicitly, even if the source files are stored in different locations.
- Merge the contents of a PDF variable generated by the `cfdocument` tag or a `cfpdf` tag
- Create PDF packages
  To merge the contents of a directory, use the `merge` action and specify the directory where the source PDF files are located, as the following example shows:

```
<cfpdf action="merge" directory="c:/BookFiles" destination="myBook.pdf"
overwrite="yes">
```

By default, ColdFusion merges the source files in descending order by timestamp. You can control the order in which the PDF files are added to the book by setting the `order` and `ascending` attributes. The following code merges the files in ascending order according to the timestamp on the files:

```
<cfpdf action="merge" directory="c:/BookFiles" destination="myBook.pdf" order="name"

 ascending="yes" overwrite="yes">
```

By default, ColdFusion continues the merge process even if it encounters a file that is not a valid PDF document in the specified directory. To override this setting, set the `stopOnError` attribute to `yes`, as the following example shows:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cfpdf action="merge" directory="c:/BookFiles" destination="myBook.pdf" order="time"

 ascending="yes" overwrite="yes" stopOnError="yes">
```

You can merge a comma-separated list of PDF files. To do this merge, specify the absolute path for each file, as the following example shows:

```
<cfpdf action="merge"
 source="c:\coldfusion\wwwroot\lion\Chap1.pdf,c:\coldfusion\wwwroot\lion\Chap2.pdf"
 destination="twoChaps.pdf" overwrite="yes">
```

For more control over which files are added to the merged document, use the `cfpdfparam` tag with the `cfpdf` tag. The `cfpdfparam` tag merges documents or pages from documents located in different directories into a single output file. When you use the `cfpdfparam` tag, the PDF files are added to the output file in the order they appear in the code. In the following example, the cover, title, and copyright pages are followed by the first five pages of the introduction, then all of the pages in Chapter 1, and then the first page followed by pages 80-95 in Chapter 2:

```
<!--- Use the cfdocument tag to create PDF content and write the output to a
variable called coverPage.--->
<cfdocument format="PDF" name="coverPage">
<html>
<body>
 <h1>Cover page</h1>
 <p>Please review the enclosed document for technical accuracy and completeness.</p>

</body>
</html>
</cfdocument>

<!--- Use the cfpdf tag to merge the cover page generated in ColdFusion with pages
from PDF  files in different locations. --->
<cfpdf action="merge" destination="myBook.pdf" overwrite="yes" keepBookmark="yes">
 <cfpdfparam source="coverPage">
 <cfpdfparam source="title.pdf">
 <cfpdfparam source="e:\legal\copyright.pdf">
 <cfpdfparam source="boilerplate\intro.pdf" pages="1-5">
 <cfpdfparam source="bookfiles\chap1.pdf">
 <cfpdfparam source="bookfiles\chap2.pdf" pages="1,80-95">
</cfpdf>
```

Because the `keepbookmark` attribute is set to `yes`, ColdFusion retains the bookmarks from the source documents in the output file.

Note

You cannot use the `cfpdf` tag to create bookmarks in a PDF document.

© 2014 Adobe Systems Incorporated. All rights reserved.

Creating PDF Portfolios

You can now create PDF packages using the `package = "true"` attribute with the `merge` action:

```
<cfpdf action="merge" package="yes" destination="./myBook/adobetest.pdf"
overwrite="yes">
 <cfpdfparam source="./inputFiles/c.zip" >
 <cfpdfparam source="./inputFiles/d.jpg" >
 <cfpdfparam source="./inputFiles/a.pdf" >
 <cfpdfparam source="./inputFiles/z.txt" >
 <cfpdfparam source="./inputFiles/MSTribute.pps" >
 <cfpdfparam source="./inputFiles/Test1.docx" >
 <cfpdfparam source="./inputFiles/NewMovie.mp3" >
 <cfpdfparam source="./inputFiles/testserver.air" >
 <cfpdfparam source="./inputFiles/123.xml" >
 <cfpdfparam source="./inputFiles/New_test_case.xls" >
</cfpdf>
```

Flattening forms created in Acrobat

Flattening forms involves removing the interactivity from the form fields. This action is useful for displaying form data and presenting it without allowing it to be altered. Use the `write` action to flatten PDF forms, as the following example shows:

```
<cfpdf action="write" flatten="yes" source="taxForms\f1040.pdf"
 destination="taxforms/flatForm.pdf" overwrite="yes">
 <a href="http://localhost:8500/Lion/taxforms/flatForm.pdf">1040 form</a>
```

Note

If you flatten a prefilled form created in Acrobat, ColdFusion flattens the form and removes the data from the form fields. When you specify a form created in Acrobat as a source file for `merge` action of the `cfpdf` tag, ColdFusion automatically flattens the form and removes data from the form fields, if the fields are filled in. ColdFusion does not support flattening forms created in LiveCycle.

Linearizing PDF documents for faster web display

For efficient access of PDF files over the web, linearize PDF documents. A linearized PDF file is structured in a way that displays the first page of the PDF file in the browser before the entire file is downloaded from the web server. As a result linear PDF documents open almost instantly.
To linearize PDF documents, specify the `saveOption` attribute of the `write` action. The following example saves the output file in linear format:

```
<cfpdf action="write" saveOption="linear" source="myBook.pdf"
destination="fastBook.pdf"
 overwrite="yes">
```

© 2014 Adobe Systems Incorporated. All rights reserved.

> **Note**
>
> Linearization can decrease performance when handling large documents.

Generating thumbnail images from PDF pages

Use the `thumbnail` action to generate thumbnail images from PDF pages. If you specify only the `source` attribute with the thumbnail action, ColdFusion automatically creates a directory relative to the CFM page called thumbnails where it stores a generated JPEG image for each page in the document. The filenames are in the following format:

```
PDFdocumentName_page_n.JPG
```

For example, assume that the source file in the following example has 100 pages:

```
<cfpdf action="thumbnail" source="myBook.pdf">
```

ColdFusion generates the following files and stores them in the thumbnails directory:

```
myBook_page_1.jpg
myBook_page_2.jpg
myBook_page_3.jpg
...
myBook_page_100.jpg
```

If you specify a destination, ColdFusion does not create the thumbnails directory and stores the files in the specified directory instead. The following code generates a thumbnail image called myBook_page_1.jpg from the first page of myBook.pdf and stores it in a directory called images, which is relative to the CFM page:

```
<cfpdf action="thumbnail" source="myBook.pdf" pages="1" destination="images">
```

You change the prefix for the thumbnail filename and the change image file format to PNG or TIFF by specifying the `imagePrefix` and `format` attributes. The following code generates a file called TOC_page_2.PNG from the second page of myBook.pdf:

```
<cfpdf action="thumbnail" source="myBook.pdf" pages="2" imagePrefix="TOC"
format="PNG"
 destination="images">
```

The following code generates thumbnails from a range of pages and changes the image background to transparent (the default is opaque):

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cfpdf action="thumbnail" source="myBook.pdf" pages="1-10,15,8-16,59"
transparent="yes"
 destination="\myBook\subset" imagePrefix="abridged">
```

For an example of how to generate thumbnail images and link them to pages in the source PDF document, see the cfpdf tag in the *CFML Reference*.
ColdFusion 9 release has introduced some new attributes for the thumbnail action:

- `hires`: You can set this attribute to true to extract high resolution images from the page. If a document contains high resolution images and you want to retain the resolution of the images, then this attribute is useful. For example:

```
<cfpdf action="thumbnail" source="./WORK/myBook.pdf"
destination="./WORK/Testing_CFPDF" overwrite="true" hires="yes">
```

- `overridepage`: If you set this attribute to true, the thumbnail generated does not adhere to the PDF page size, but to the image size that is present in that page. If the image is not present, the size is set to the maximum size of the page.
- `compresstiffs`: Use this attribute to compress the size of the thumbnail images. As the name of the attribute suggests, it is only valid for the TIFF format. Following is an example:

```
<cfpdf action="thumbnail" source="C:\WORK\myBook.pdf"
destination="C:\WORK\Testing_CFPDF" overwrite="true" hires="yes" format="tiff"
compresstiffs="yes">
```

- `maxscale` : Use this attribute to specify an integer value for the maximum scale of the thumbnail images.
- `maxlength`: Use this attribute to specify an integer value of the maximum length of the thumbnail images.
- `maxbreadth`: Use this attribute to specify an integer value of the maximum width of the thumbnail.The following example illustrates the use of maxscale, maxlength, and maxbreadth:

```
<cfpdf action="thumbnail" source="./WORK/myBook.pdf"
destination="./WORK/Testing_CFPDF" overwrite="true" format="jpg" maxscale="3"
maxlength="300" maxbreadth="200" hires="yes" scale="100">
```

Note

Typically, the value of the `scale` attribute is set to '100' when using the `maxscale` attribute.

Using the Duplicate function to create versions of a PDF document

You can use the `Duplicate` function to clone PDF variables, which is an efficient way to create different versions of a PDF document from a single source file. For example, you can customize PDF output based on your audience by

© 2014 Adobe Systems Incorporated. All rights reserved.

creating clones of a PDF variable and performing different actions on each clone. The following example shows how to create a clone of a PDF document in memory, and create one version of the document with a watermark and another version of the document where permissions are restricted:

```
<cfset filename="coldfusion.pdf">
<!--- This code reads a PDF document into a PDF variable called pdfVar1.
  --->
<cfpdf action="read" source="#filename#" name="pdfVar1">
<!--- This code uses the Duplicate function to create a clone of pdfVar1 called
pdfVar2. --->
<cfset pdfVar2=Duplicate(pdfVar1)>
<!--- This code creates a watermarked version of the source PDF document from the
pdfVar1
 variable. --->
<cfpdf action="addwatermark" source="pdfVar1" rotation="45" image="watermark.jpg"
 destination="watermark_coldfusion.pdf" overwrite="yes">
<!--- This code creates a protected version of the source PDF document from the
pdfVar2
 variable. --->
<cfpdf action=protect source="pdfVar2" encrypt="RC4_128" permissions="none"
 newownerpassword="owner1" destination="restricted_coldfusion.pdf" overwrite="yes">
```

© 2014 Adobe Systems Incorporated. All rights reserved.

Using DDX to perform advanced tasks

LiveCycle Assembler is a server-based application that processes DDX, a declarative markup language used to define PDF output files.
The `processddx` action lets you process DDX instructions without installing LiveCycle Assembler. In addition to all of the functionality available with the other `cfpdf` actions, you can use DDX instructions to perform advanced tasks, such as adding a generated table of contents to a PDF document, adding headers and footers with automatic page numbers, and creating groups of PDF documents to which you can apply formatting instructions.
ColdFusion does not provide complete LiveCycle Assembler functionality. For a list of DDX elements that you can access from ColdFusion, see Supported DDX elements in the *CFML Reference*.
For complete DDX syntax, see the *Adobe LiveCycle Assembler Document Description XML Reference.*

Using DDX instructions with ColdFusion

Although you can type DDX instructions directly in ColdFusion, typically you use an external DDX file. A DDX file is basically an XML file with a DDX extension (for example, merge.ddx). You can use any text editor to create a DDX file. The DDX syntax requires that you enclose the instructions within DDX start and end tags. In the following example, the `PDF` element provides instructions for merging two PDF source files (Doc1 and Doc2) into a result file (Out1):

```
<?xml version="1.0" encoding="UTF-8"?>
<DDX xmlns="http://ns.adobe.com/DDX/1.0/"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://ns.adobe.com/DDX/1.0/ coldfusion_ddx.xsd">
<PDF result="Out1">
<PDF source="Doc1"/>
<PDF source="Doc2"/>
</PDF>
</DDX>
```

In ColdFusion, you verify the source DDX file with the `IsDDX` function:

```
<!--- The following code verifies that the DDX file exists and the DDX instructions
are
 valid. --->
<cfif IsDDX("merge.ddx")>
```

To implement the DDX instructions in ColdFusion, you create two structures: an input structure that maps the DDX input instructions to the PDF source files, and an output structure that maps the DDX output instructions to a PDF output file,
The following code maps two files called Chap1.pdf and Chap2.pdf to the Doc1 and Doc2 sources that you defined in the DDX file:

```
<!--- This code creates a structure for the input files. --->
<cfset inputStruct=StructNew()>
<cfset inputStruct.Doc1="Chap1.pdf">
<cfset inputStruct.Doc2="Chap2.pdf">
```

© 2014 Adobe Systems Incorporated. All rights reserved.

The following code maps the output file called twoChaps.pdf to the Out1 result instruction that you defined in the DDX file:

```
<!--- This code creates a structure for the output file. --->
<cfset outputStruct=StructNew()>
<cfset outputStruct.Out1="twoChaps.pdf">
```

To process the DDX instructions, you use the `processddx` action of the `cfpdf` tag, in which you reference the DDX file, the input structure, and the output structure, as the following example shows:

```
<cfpdf action="processddx" ddxfile="merge.ddx" inputfiles="#inputStruct#"
 outputfiles="#outputStruct#" name="myBook">
```

The `name` attribute creates a variable that you can use to test the success or failure of the action. If the action is successful, ColdFusion generates an output file with the name and location specified in the output structure. The following code returns a structure that displays a success, reason for failure, or failure message (if the reason is unknown) for each output file, depending on the result:

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

The previous example performs the same task as the `merge` action in ColdFusion, as the following example shows:

```
<cfpdf action="merge" destination="twoChaps.pdf" overwrite="yes">
 <cfpdfparam source="Chap1.pdf">
 <cfpdfparam source="Chap2.pdf">
</cfpdf>
</cfif>
```

In this situation, it makes more sense to use the `merge` action because it is easier. DDX is useful when you have to perform tasks that you can't perform with other actions in the `cfpdf` tag, or you require more control over specific elements.

Adding a table of contents

You use DDX instructions to add a generated table of contents page to the PDF output file. Generating a table of contents is useful if you are assembling documents from multiple sources. You can generate a table of contents that contains active links to pages in the assembled PDF document. The following code shows how to create DDX instructions to merge two documents and add a table of contents:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<?xml version="1.0" encoding="UTF-8"?>
<DDX xmlns="http://ns.adobe.com/DDX/1.0/"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://ns.adobe.com/DDX/1.0/ coldfusion_ddx.xsd">
 <PDF result="Out1">
   <PDF source="DocumentTitle"/>
   <TableOfContents/>
   <PDF source="Doc1"/>
   <PDF source="Doc2"/>
 </PDF>
</DDX>
```

The `TableOfContents` element generates a table of contents from the `PDF source` elements that follow it. Order is important: in the previous example, the table of contents appears on a separate page after the Title and before Doc 1 and Doc 2. The table of contents contains entries from Doc 1 and 2, but not from the title page, because the title page precedes the table of contents in the order of instructions.
You do not reference the `TableOfContents` element on the corresponding ColdFusion page, as the following example shows:

```
<!--- The following code verifies that the DDX file exists and the DDX instructions
are
 valid. --->
<cfif IsDDX("makeBook.ddx")>

<!--- This code creates a structure for the input files. --->
<cfset inputStruct=StructNew()>
<cfset inputStruct.Title="Title.pdf">
<cfset inputStruct.Doc1="Chap1.pdf">
<cfset inputStruct.Doc2="Chap2.pdf">

<!--- This code creates a structure for the output file. --->
<cfset outputStruct=StructNew()>
<cfset outputStruct.Out1="Book.pdf">

<!--- This code processes the DDX instructions and generates the book. --->
<cfpdf action="processddx" ddxfile="makeBook.ddx" inputfiles="#inputStruct#"
 outputfiles="#outputStruct#" name="myBook">
</cfif>
```

ColdFusion generates a table of contents from the DDX instructions and inserts it in the PDF document in the location that you provided in the DDX file. By default, the table of contents contains active links to the top-level bookmarks in the merged PDF document.
You can change the default `TableOfContents` settings in the DDX file, as the following example shows:

```
<TableOfContents maxBookmarkLevel="infinite" bookmarkTitle="Table of Contents"
 includeInTOC="false"/>
```

Use the `maxBookmarkLevel` attribute to specify the level of bookmarks included on the table of contents page.

© 2014 Adobe Systems Incorporated. All rights reserved.

Valid values are `infinite` or an integer. Use the `bookmarkTitle` attribute to add a bookmark to the table of contents page in the output file. The `includeInTOC` attribute specifies whether the bookmark title is included on the table of contents page.

> **Note**
>
> You cannot specify keywords as the source for DDX. For example, if you specify <PDF source = "Title"/> and then add the <_BookmarkTitle/> tag in the DDX file, ColdFusion throws an exception. This is because, the _BookmarkTitle tag is converted to TITLE and DDX is case-sensititve.

For more information on the `TableOfContents` element, see the *Adobe LiveCycle Assembler Document Description XML Reference.*

Adding headers and footers

To add headers and footers to a PDF document, specify the `Header` and `Footer` elements in the DDX file. The following example specifies headers and footers for the PDF source called Doc2:

```
<?xml version="1.0" encoding="UTF-8"?>
<DDX xmlns="http://ns.adobe.com/DDX/1.0/"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://ns.adobe.com/DDX/1.0/ coldfusion_ddx.xsd">
 <PDF result="Out1">
  <PDF source="Title"/>
  <TableOfContents/>
  <PDF source="Doc2" >
   <Header>
    <Right>
     <StyledText><p>Right-justified header text</p></StyledText>
    </Right>
    <Left>
     <StyledText><p>Left-justified header text</p></StyledText>
    </Left>
   </Header>
   <Footer>
    <Center>
     <StyledText><p>Centered Footer</p></StyledText>
    </Center>
   </Footer>
  </PDF>
 </PDF>
</DDX>
```

In this example, the `Header` and `Footer` elements apply only to Doc2 because they are contained within that PDF source start and end tags; they do not apply to the table of contents or to the title page, which precede the `Header` and `Footer` elements.

Formatting headers and footers

You use DDX instructions to perform the following tasks:

- Add automatic page numbers to headers and footers
- Use style profiles

© 2014 Adobe Systems Incorporated. All rights reserved.

- Group documents in the PDF output file

Adding automatic page numbers

To add automatic page numbers, use the `_PageNumber` and `_LastPagenumber` built-in keys within the `Header` or `Footer` elements. The following code shows how to create footers with right-justified automatic page numbers:

```
<Footer>
 <Right>
  <StyledText>
   <p>Page <_PageNumber/> of <_LastPageNumber/></p>
  </StyledText>
 </Right>
</Footer>
```

The first page of the output file is numbered "Page 1 of *n*", and so on.
For more information on built-in keys, see the *Adobe LiveCycle Assembler Document Description XML Reference.*

Using style profiles

The previous example uses the `StyledText` element to define inline text formatting. To define styles that you can apply by reference, use the `StyleProfile` element. Style profiles let you apply a set of styles to different elements in the PDF output file. The following code shows how to define a style profile for the table of contents `Header`:

```
<StyleProfile name="TOCheaderStyle">
 <Header>
  <Center>
   <StyledText>
   <p> color="red" font-weight="bold" font="Arial">Table of Contents</p>
   </StyledText>
  </Center>
 </Header>
</StyleProfile>
```

To apply the style profile, specify the `StyleProfile` name by using the `styleReference` attribute of the `Header` element, as the following example shows:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<?xml version="1.0" encoding="UTF-8"?>
<DDX xmlns="http://ns.adobe.com/DDX/1.0/"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://ns.adobe.com/DDX/1.0/ coldfusion_ddx.xsd">
<PDF result="Out1">
 <PDF source="Title"/>
 <TableOfContents>
  <Header styleReference="TOCheaderStyle"/>
 </TableOfContents>
  <PDF source="Doc1"/>
  <PDF source="Doc2"/>
  <PDF source="Doc3"/>
  <PDF source="Doc4"/>
 </PDF>

 <StyleProfile name="TOCheaderStyle">
 <Header>
  <Center>
   <StyledText>
   <p> color="red" font-weight="bold" font="Arial">Table of Contents</p>
   </StyledText>
  </Center>
 </Header>
 </StyleProfile>
</DDX>
```

Grouping PDF documents

To apply a style profile to a group of documents in the output PDF file, use the `PDFGroup` element. The following example shows how to create a group of chapters in the output file and apply a style profile to the `Footer` element for all of the documents in the group:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<?xml version="1.0" encoding="UTF-8"?>
<DDX xmlns="http://ns.adobe.com/DDX/1.0/"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://ns.adobe.com/DDX/1.0/ coldfusion_ddx.xsd">
<PDF result="Out1">
 <PageLabel prefix="page " format="Decimal"/>
 <PDF source="Title"/>
 <TableOfContents>
  ...
 </TableOfContents>
 <PDFGroup>
  <Footer styleReference="FooterStyle" />
  <PDF source="Doc1"/>
  <PDF source="Doc2"/>
  <PDF source="Doc3"/>
  <PDF source="Doc4"/>
 </PDFGroup>
 </PDF>

<StyleProfile name="FooterStyle">
 <Footer>
  <Left>
   <StyledText>
   <p font-size="9pt"><i>CFML Reference</i></p>
   </StyledText>
  </Left>
  <Right>
   <StyledText>
   <p font-size="9pt">Page <_PageNumber/> of <_LastPageNumber/></p>
   </StyledText>
  </Right>
 </Footer>
 </StyleProfile>
</DDX>
```

For a complete example, see *Using DDX instructions to create a book* in Application examples.

Setting the initial view of a PDF document

To set the initial view of a PDF document, use the `InitialViewProfile` DDX element. Setting the initial view determines how the PDF output file is displayed on the screen when it is first opened in Adobe Acrobat Reader. You reference the `InitialViewProfile` by using the `InitialView` attribute of the `PDF result` element, as the following example shows:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<?xml version="1.0" encoding="UTF-8"?>
<DDX xmlns="http://ns.adobe.com/DDX/1.0/"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://ns.adobe.com/DDX/1.0/ coldfusion_ddx.xsd">
<PDF result="Out1" initialView="firstView">
  ...
<InitialViewProfile name="firstView" show="BookmarksPanel" magnification="FitPage"
 openToPage="2"/>
...
</DDX>
```

In this example, the first time the PDF document is displayed in Acrobat Reader, the document is opened to page two and the bookmark panel is displayed. The magnification of the document is adjusted to fit the page.
For more information on `IntialViewProfile` settings, see the *Adobe LiveCycle Assembler Document Description XML Reference.*

Adding text-string watermarks

You use the `processddx` action with the `Background` or `Watermark` DDX elements to create a text-string watermark. `Background` elements appear in the background (behind the contents of the page); `Watermark` eleme nts display in the foreground (over the contents of the page). The syntax for both the elements is the same.
The following example shows the DDX page for using the text string `"DRAFT"` as a watermark. The watermark appears on every page of the output file. By default, the watermark appears in the middle of the page. In this example, the watermark is rotated 30 degrees:

```
<?xml version="1.0" encoding="UTF-8"?>
<DDX xmlns="http://ns.adobe.com/DDX/1.0/"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://ns.adobe.com/DDX/1.0/ coldfusion_ddx.xsd">
<PDF result="Out1">
  <Watermark rotation="30" opacity="65%">
   <StyledText><p font-size="50pt" font-weight="bold" color="lightgray"
    font="Arial">DRAFT</p></StyledText>
  </Watermark>
...
 </PDF>
</DDX>
```

The following example shows how to add different backgrounds on alternating pages. The `verticalAnchor` attribu te displays the background text at the top of the page:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<?xml version="1.0" encoding="UTF-8"?>
<DDX xmlns="http://ns.adobe.com/DDX/1.0/"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://ns.adobe.com/DDX/1.0/ coldfusion_ddx.xsd">
<PDF result="Out1">
  <Background alternation="EvenPages" verticalAnchor="Top">
   <StyledText><p font-size="20pt" font-weight="bold" color="gray"
    font="Arial">DRAFT</p></StyledText>
  </Background>
  <Background alternation="OddPages" verticalAnchor="Top">
  <StyledText><p font-size="20pt" font-weight="bold" color="gray"
   font="Arial"><i>Beta 1</i></p></StyledText>
  </Background>
...
 </PDF>
</DDX>
```

Instead of applying watermarks to the entire output file, you can apply them to individual source files. The following example applies a different background to the first three chapters of a book. The fourth chapter has no background:

```
<?xml version="1.0" encoding="UTF-8"?>
<DDX xmlns="http://ns.adobe.com/DDX/1.0/"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://ns.adobe.com/DDX/1.0/ coldfusion_ddx.xsd">
<PDF result="Out1">
   <PDF source="Doc1">
   <Background>
   <StyledText><p font-size="20pt" font-weight="bold" color="lightgray"
    font="Arial">CHAPTER 1</p></StyledText>
   </Background>
  </PDF>
  <PDF source="Doc2">
   <Background>
    <StyledText><p font-size="20pt" font-weight="bold"
     color="lightgray" font="Arial">CHAPTER 2</p></StyledText>
   </Background>
  </PDF>
  <PDF source="Doc3">
   <Background>
    <StyledText><p font-size="20pt" font-weight="bold"
     color="lightgray" font="Arial">CHAPTER 3</p></StyledText>
   </Background>
  </PDF>
  <PDF source="Doc4"/>
 </PDF>
</DDX>
```

For more information on using DDX instructions to create watermarks, see the *Adobe LiveCycle Assembler Document Description XML Reference.*

Extracting text from a PDF document

© 2014 Adobe Systems Incorporated. All rights reserved.

You can use the `DocumentText` DDX element to return an XML file that contains the text in one or more PDF documents. As with the `PDF` element, you specify a result attribute the `DocumentText` element and enclose one or more `PDF source` elements within the start and end tags, as the following example shows:

```
<?xml version="1.0" encoding="UTF-8"?>
<DDX xmlns="http://ns.adobe.com/DDX/1.0/"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://ns.adobe.com/DDX/1.0/ coldfusion_ddx.xsd">
 <DocumentText result="Out1">
  <PDF source="doc1"/>
 </DocumentText>
</DDX>
```

The following code shows the CFM page that calls the DDX file. Instead of writing the output to a PDF file, you specify an XML file for the output:

```
<cfif IsDDX("documentText.ddx">
 <cfset ddxfile = ExpandPath("documentText.ddx")>
 <cfset sourcefile1 = ExpandPath("book1.pdf")>
 <cfset destinationfile = ExpandPath("textDoc.xml")>

 <cffile action="read" variable="myVar" file="#ddxfile#"/>

 <cfset inputStruct=StructNew()>
 <cfset inputStruct.Doc1="#sourcefile1#">

 <cfset outputStruct=StructNew()>
 <cfset outputStruct.Out1="#destinationfile#">

 <cfpdf action="processddx" ddxfile="#myVar#" inputfiles="#inputStruct#"
outputfiles="#outputStruct#" name="ddxVar">

 <!--- Use the cfdump tag to verify that the PDF files processed successfully. --->
 <cfdump var="#ddxVar#">
</cfif>
```

The XML file conforms to a schema specified in doctext.xsd. For more information, see http://ns.adobe.com/DDX/DocText/1.0

When you specify more than one source document, ColdFusion aggregates the pages into one file. The following example shows the DDX code for combining a subset of pages from two documents into one output file:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<?xml version="1.0" encoding="UTF-8"?>
<DDX xmlns="http://ns.adobe.com/DDX/1.0/"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://ns.adobe.com/DDX/1.0/ coldfusion_ddx.xsd">
<DocumentText result="Out1">
 <PDF source="doc1" pages="1-10"/>
 <PDF source="doc2" pages="3-5"/>
</DocumentText>
</DDX>
```

Support for restricted DDX elements

In ColdFusion 11, the restricted DDX elements are supported for the Enterprise Edition. See this document for the list.

© 2014 Adobe Systems Incorporated. All rights reserved.

FDF format support for AcroForms

PDF Acro Form now support the Forms Data Format (FDF). You can now export, import, and populate Acroforms in FDF using the read and populate actions. The action attribute of the `cfpdfform` tag contains a new attribute called `fdfdata`, where you provide the name of the PDF form to be exported or imported.
Following is an example of how can export a PDF form in FDF format using the `read` action:

```
<cfpdfform source= "acroform_export.pdf" action="read" fdfdata="abc.fdf" >
</cfpdfform>
```

To import form data:

```
<cfpdfform source= "write_acrroform.pdf" action="populate" fdfdata="abc.fdf"
destination="hello.pdf">
</cfpdfform>
```

Form data can now be populated in a PDF document using the `populate` action. The new `fdf` attribute with populate allows you use the FDF format internally. Following code snippet illustrates this feature:

```
<cfpdfform source="acroform2.pdf" destinat ion="source_result1 7.pdf"
action="populate"  overwrite="true" fdf ="true">
 <cfpdfsubform  name="Text1">
 <cfpdf subform name="0">
 <cfp dfformparam name="0" val ue="Test1.0.0">
 <cfpd fformparam name="1" value ="Test1.0.1">
 <cfpdfformparam name="2" value="Test1.0.2">
 </cfpdfsubform>
  <cfpdfsubform name="1">
  <cfpdfformparam name="0" value="Test1.1.0">
  <cfpdfformparam name="1" value="Test1.1.1">
  <cfpdfformparam name="2" value="Test1.1.2">
 </cfpdfsubform>

 </cfpdfsubform>
 <cfpdfsubform name="Text2">
 <cfpdfformparam name="0" value="Test2.0">
 <cfpdfformparam name="1" value="Test2.1">
 <cfpdfformparam name="2" value="Test2.2">
 <cfpdfformparam name="3" value="Test2.3">
 </cfpdfsubform><cfpdfformparam name="Text3" value="Test3">
 <cfpdfformparam name="Text4" value="Test4">
 <cfpdfformparam name="checkbox1" value="Yes">
 <cfpdfformparam name="listbox1" value="item4">
 <cfpdfformparam name="radiobutton1" value="2">
</cfpdfform>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

Application examples

The following examples show you how to use the `cfpdf` tag to perform PDF document operations in simple applications.

Merging documents based on a keyword search

The following example shows how to use the `getInfo` and `merge` actions to assemble a PDF document from multiple tax files based on business type (Sole Proprietor, Partnership, or S Corporation). The application assembles the tax forms and information booklets based on a radio button selection. Some tax forms and booklets apply to more than one business type (for example, Partnership and S Corporations both use the tax form f8825.pdf). For instructions on setting keywords for PDF documents, see Managing PDF document information.
This example shows how to perform the following tasks:

- Use the `getInfo` action to perform a keyword search on PDF files in a directory.
- Create a comma-separated list of files that match the search criteria.
- Use the `merge` action to merge the PDF documents in the comma-separated list into an output file.
  The first CFM page creates a form for selecting the tax documents based on the business type:

```
<h3>Downloading Federal Tax Documents</h3>
<p>Please choose the type of your business.</p>
<!--- Create the ColdFusion form to determine which PDF documents to merge. --->
<table>
<cfform action="cfpdfMergeActionTest.cfm" method="post">
 <tr><td><cfinput type="radio" name="businessType"
   Value="Sole Proprieter">Sole Proprietor</td></tr>
 <tr><td><cfinput type="radio" name="businessType"
   Value="Partnership">Partnership</td></tr>
 <tr><td><cfinput type="radio" name="businessType" Value="S Corporation">
   S Corporation</td></tr>
 <cfinput type = "hidden" name = "selection required" value = "must make a
selection">
 <tr><td><cfinput type="Submit" name="OK" label="OK"></td></tr>
 </tr>
</cfform>
</table>
```

The action page loops through the files in the taxes subdirectory and uses the `getInfo` action to retrieve the keywords for each file. If the PDF file contains the business type keyword (Sole Proprietor, Partnership, or S Corporation), ColdFusion adds the absolute path of the file to a comma-separated list. The `merge` action assembles the files in the list into an output PDF file:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<!--- Create a variable for the business type selected from the form. --->
<cfset bizType=#form.businessType#>
<!--- Create a variable for the path of the current directory. --->
<cfset thisPath=ExpandPath(".")>

<!--- List the files in the taxes subdirectory. --->
<cfdirectory action="list" directory="#thisPath#\taxes" name="filelist">

<!--- The following code loops through the files in the taxes subdirectory. The
getInfo
 action to retrieves the keywords for each file and determines whether the business
type
 matches one of the keywords in the file. If the file contains the business type
keyword,
 ColdFusion adds the file to a comma-separated list. --->
<cfset tempPath="">
<cfloop query="filelist">
 <cfset fPath="#thisPath#\taxes\#filelist.name#">
 <cfpdf action="GetInfo" source="#fPath#" name="kInfo"></cfpdf>
 <cfif #kInfo.keywords# contains "#bizType#">
  <cfset tempPath=#tempPath# & #fPath# & ",">
 </cfif>
</cfloop>

<!--- Merge the files in the comma-separated list into a PDF output file called
"taxMerge.pdf". --->
<cfpdf action="merge" source="#tempPath#" destination="taxMerge.pdf"
overwrite="yes"/>

<h3>Assembled Tax Document</h3>
<p>Click the following link to view your assembled tax document:</p>
<a href="http://localhost:8500/Lion/taxmerge.pdf">
  <p>Your Assembled Tax Document</a></p>
```

Using DDX instructions to create a book

The following example shows how to create a book using DDX instructions with the `processddx` action. Specifically, it shows how to perform the following tasks:

- Merge several PDF documents into an output file.
- Add a generated table of contents page.
- Add headers and footers.
- Add automatic page numbers.
- Apply different styles to the table of contents and the body of the book.
  The following code shows the DDX file:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<?xml version="1.0" encoding="UTF-8"?>
<DDX xmlns="http://ns.adobe.com/DDX/1.0/"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://ns.adobe.com/DDX/1.0/ coldfusion_ddx.xsd">
<PDF result="Out1">
 <PDF source="Doc0"/>
<TableOfContents maxBookmarkLevel="3" bookmarkTitle="Table of Contents"
 includeInTOC="false">
  <Header styleReference="TOCheaderStyle"/>
  <Footer styleReference="TOCFooterStyle"/>
 </TableOfContents>
 <PDFGroup>
  <Footer styleReference="FooterStyle"/>
  <PDF source="Doc1"/>
  <PDF source="Doc2"/>
  <PDF source="Doc3"/>
  <PDF source="Doc4"/>
 </PDFGroup>
 </PDF>

 <StyleProfile name="TOCheaderStyle">
 <Header>
  <Center>
   <StyledText>
   <p color="red" font-weight="bold" font="Arial">Table of Contents</p>
   </StyledText>
  </Center>
 </Header>
 </StyleProfile>

 <StyleProfile name="TOCFooterStyle">
 <Footer>
  <Right>
   <StyledText>
   <p font-size="9pt">Page <_PageNumber/> of <_LastPageNumber/></p>
   </StyledText>
  </Right>
 </Footer>
 </StyleProfile>

 <StyleProfile name="FooterStyle">
 <Footer>
  <Left>
   <StyledText>
   <p font-size="9pt"><i>CFML Reference</i></p>
   </StyledText>
  </Left>
  <Right>
   <StyledText>
   <p font-size="9pt">Page <_PageNumber/> of <_LastPageNumber/></p>
   </StyledText>
  </Right>
 </Footer>
 </StyleProfile>

</DDX>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

The following code shows the ColdFusion page that processes the DDX instructions:

```
<cfif IsDDX("Book.ddx")>
 <cfset inputStruct=StructNew()>
 <cfset inputStruct.Doc0="Title.pdf">
 <cfset inputStruct.Doc1="Chap1.pdf">
 <cfset inputStruct.Doc2="Chap2.pdf">
 <cfset inputStruct.Doc3="Chap3.pdf">
 <cfset inputStruct.Doc4="Chap4.pdf">

 <cfset outputStruct=StructNew()>
 <cfset outputStruct.Out1="myBook.pdf">

 <cfpdf action="processddx" ddxfile="book.ddx" inputfiles="#inputStruct#"
  outputfiles="#outputStruct#" name="ddxVar">

 <cfoutput>#ddxVar.Out1#</cfoutput>
</cif>
```

Applying a watermark to a form created in Acrobat

The following example shows how to prefill an interactive Acrobat tax form and apply a text-string watermark to the completed form that the user posted. Specifically, this example shows how to perform the following tasks:

- Use the `cfpdfform` and `cfpdfformparam` tags to populate a form created in Acrobat.
- Use the `cfpdfform` tag to write the output of a PDF post submission to a file.
- Use the `cfpdf processddx` action to apply a text-string watermark to the completed form.

> **Note**
>
> This example uses the cfdocexamples database and the 1040 and 1040ez Federal tax forms. A valid user name is "cpeterson." To download the 1040 and 1040ez IRS tax forms used in this example, go to the IRS website. Open the forms in Acrobat (not LiveCycle Designer) and add a submit button that points to the URL for the ColdFusion processing page. Also, add a hidden field with a variable that contains a unique filename used for the completed tax form.

The first ColdFusion page creates a login form that prompts for the user name and Social Security Number:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<!--- The following code creates a simple form for entering a user name and
password. The
 code does not include password verification. --->

<h3>Tax Login Form</h3>
<p>Please enter your user name and your social security number.</p>
<cfform name="loginform" action="TaxFile2.cfm" method="post">
<table>
 <tr>
  <td>User name:</td>
  <td><cfinput type="text" name="username" required="yes"
    message="A user name is required."></td>
 </tr>
 <tr>
  <td>SSN#:</td>
  <td><cfinput type="text" name="SS1" maxLength="3" size="3"
   required="yes" mask="999"> -
  <cfinput type="text" name="SS2" maxLength="2" size="2" required="yes"
   mask="99"> -
  <cfinput type="text" name="SS3" maxLength="4" size="4" required="yes"
   mask="9999"></td>
 </tr>
</table>
 <br/>
 <cfinput type="submit" name="submit" value="Submit">
</cfform>
```

The second ColdFusion page retrieves the user information from the cfdocexamples database. Also, it creates a pop-up menu with a list of available tax forms:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<!--- The following code retrieves all of the employee information for the
   user name entered on the login page. --->
<cfquery name="getEmpInfo" datasource="cfdocexamples">
SELECT * FROM EMPLOYEES
WHERE EMAIL = <cfqueryparam value="#FORM.username#">
</cfquery>

<h3>Choose a tax form</h3>
<p>Hello <cfoutput>#getEmpInfo.firstname#</cfoutput>,</p>
<p>Please choose a tax form from the list:</p>
<!--- Create a pop-up menu with a list of tax forms. --->
<cfset thisPath=ExpandPath(".")>
<!--- Create a variable called filerID that is a combination of the username and the
last
 three digits of the Social Security number. --->
<cfset filerID="#form.username#_#form.SS3#">
<cfdirectory action="list" name="taxForms" directory="#thisPath#/taxforms">
<cfform name="taxList" method="post" action="TaxFile3.cfm">
 <cfselect query="taxForms" value="name" size="10" required="yes" multiple="no"
  name="myTaxForm"/>
 <br/><br/>
 <cfinput type="Submit" name="OK" label="OK">
 <!--- Use hidden fields to pass the first name, last name, and the three parts of
  the SSN# to the tax form. Also, create a hidden field for the filerID variable.
--->
 <cfinput type="hidden" name="FirstName" value="#getEmpInfo.FirstName#">
 <cfinput type="hidden" name="LastName" value="#getEmpInfo.LastName#">
 <cfinput type="hidden" name="Phone" value="#getEmpInfo.Phone#">
 <cfinput type="hidden" name="SS1" value="#form.SS1#">
 <cfinput type="hidden" name="SS2" value="#form.SS2#">
 <cfinput type="hidden" name="SS3" value="#form.SS3#">
 <cfinput type="hidden" name="taxFiler" value="#filerID#">
</cfform>
```

The third ColdFusion page uses the `cfpdfform` and `cfpdfformparam` tags to populate the tax form with the user information. ColdFusion displays the tax prefilled tax form in the browser window where the user can complete the rest of the form fields. When the user clicks the submit button, Acrobat sends the completed PDF form to the ColdFusion processing page.

Note

To prefill forms, map each PDF form field name to the corresponding data element in a `cfpdfformparam` tag. To view the form fields, open the form in Acrobat Professional and select Forms > Edit Forms in Acrobat. For more information about prefilling forms, see Manipulating PDF Forms in ColdFusion.

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<!--- The following code populates the tax form template chosen from the list with
 information from the database query and the login form. Because no destination is
 specified, ColdFusion displays the interactive PDF form in the browser. A hidden
field
 in the PDF form contains the name of the output file to write. It is a combination
of
 the user name and the last three numerals of the user SSN#. The submit button added
to
 the form created in Acrobat contains a URL to the ColdFusion processing page. --->

<cfpdfform source="taxForms/#form.myTaxForm#" action="populate">
 <cfif "taxForms/#form.myTaxForm#" is "taxForms/f1040.pdf">
  <cfpdfformparam name="f1_04(0)" value="#form.Firstname#">
  <cfpdfformparam name="f1_05(0)" value="#form.Lastname#">
  <cfpdfformparam name="f2_115(0)" value="#form.Phone#">
  <cfpdfformparam name="f1_06(0)" value="#form.SS1#">
  <cfpdfformparam name="f1_07(0)" value="#form.SS2#">
  <cfpdfformparam name="f1_08(0)" value="#form.SS3#">
  <cfpdfformparam name="filerID" value="#form.taxFiler#_1040">
 <cfelseif "taxForms/#form.myTaxForm#" is "taxForms/f1040ez.pdf">
  <cfpdfformparam name="f1_001(0)" value="#form.Firstname#">
  <cfpdfformparam name="f1_002(0)" value="#form.Lastname#">
  <cfpdfformparam name="f1_070(0)" value="#form.Phone#">
  <cfpdfformparam name="f1_003(0)" value="#form.SS1#">
  <cfpdfformparam name="f1_004(0)" value="#form.SS2#">
  <cfpdfformparam name="f1_005(0)" value="#form.SS3#">
  <cfpdfformparam name="filerID" value="#form.taxFiler#_1040ez">
 </cfif>
</cfpdfform>
```

The fourth ColdFusion page uses the `cfpdfform` tag to process the PDF post submission and generate an output file. The filename is generated from the value of the hidden field in the tax form. The `processddx` action of the `cfpdf` tag uses the DDX instructions in the watermark.ddx file to generate a text-string watermark and apply it to the form.

The following code shows the contents of the watermark.ddx file:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<?xml version="1.0" encoding="UTF-8"?>
<DDX xmlns="http://ns.adobe.com/DDX/1.0/"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://ns.adobe.com/DDX/1.0/ coldfusion_ddx.xsd">
<PDF result="Out1">
 <PDF source="Doc1">
  <Watermark rotation="30" opacity="65%">
   <StyledText><p font-size="85pt" font-weight="bold" color="gray"
    font="Arial">FINAL</p></StyledText>
  </Watermark>
 </PDF>
 </PDF>
</DDX>

<!--- The following code reads the PDF file submitted in binary format and generates
a result
 structure called fields. The cfpdfform populate action and the cfoutput tags
reference
 the fields in the structure. --->
<cfpdfform source="#PDF.content#" action="read" result="fields"/>

<cfpdfform action="populate" source="#PDF.content#"
 destination="FiledForms\#fields.filerID#.pdf" overwrite="yes"/>

<!--- The following code verifies that the DDX file exists and the DDX instructions
are
 valid. --->
<cfif IsDDX("watermark.ddx")>
 <!--- The following code uses the processddx action of the cfpdf tag to create a
text-
  string watermark. --->

 <!--- This code creates a structure for the input files. --->
 <cfset inputStruct=StructNew()>
 <cfset inputStruct.Doc1="FiledForms\#fields.filerID#.pdf">

 <!--- This code creates a structure for the output file. --->
 <cfset outputStruct=StructNew()>
 <cfset outputStruct.Out1="FiledForms\#fields.filerID#.pdf">

 <!--- This code processes the DDX instructions and applies the watermark to the
form. --->
 <cfpdf action="processddx" ddxfile="watermark.ddx" inputfiles="#inputStruct#"
outputfiles="#outputStruct#" name="Final">
</cfif>

<h3>Tax Form Completed</h3>
<p>Thank you for filing your tax form on line. Copy this URL to view or download
your filed
 tax form:</p>
<cfoutput>
<a href="http://localhost:8500/Lion/FiledForms/#fields.filerID#.pdf">
 Link to your completed tax form</a>
</cfoutput>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

#back to top

© 2014 Adobe Systems Incorporated. All rights reserved.

Creating and Manipulating ColdFusion Images

You can use Adobe ColdFusion to create and manipulate images, retrieve and store images in a database, retrieve image information for indexes and searches, convert images from one format to another, and write images to the hard drive.

© 2014 Adobe Systems Incorporated. All rights reserved.

About ColdFusion images

ColdFusion lets you create and manipulate images dynamically. With ColdFusion, you can automate many image effects and drawing functions that you perform manually in Adobe Photoshop or other imaging software packages and integrate the images in your application. For example, contributors to a website can upload photos in different formats. You can add a few lines of code to your ColdFusion application to verify the images, reformat the images to a standard size and appearance, write the modified images to a database, and display the images in a browser. The following table describes a few of the tasks you can perform with ColdFusion images:

Task	Functions and actions
Verify whether a ColdFusion variable returns an image	`IsImage` function
Verify whether a file is a valid image	`IsImageFile` function
Create thumbnail images	`ImageScaleToFit` function, the `ImageResize` function, or the `resize` action of the `cfimage` tag
Create a watermark	`ImageSetDrawingTransparency` function with any of the `ImageDraw` functions and the `ImagePaste` function
Get information about an image (for example, so you can enforce size restrictions)	`ImageGetHeight` and the `ImageGetWidth` functions or the `ImageInfo` function
Enforce compression on JPEG images	`quality` attribute of the `write` action of the `cfimage` tag or the `ImageWrite` function
Convert an image from one image file format to another (for example, convert a BMP file to a JPEG)	`cfimage` tag or `ImageRead` and `ImageWrite` functions
Convert an image file to a Base64 string	`cfimage` tag or the `ImageWriteBase64` function
Create a ColdFusion image from a Base64 string	`ImageReadBase64` function
Insert a ColdFusion image as a Binary Large Object Bitmap (BLOB) in a database	`ImageGetBlob` function within a `cfquery` statement
Create an image from a BLOB in a database	`cfimage` tag or the `ImageNew` function with a `cfquery` statement
Create an image from a binary object	`cffile` tag to convert an image file to a binary object and then pass the binary object to the `ImageNew` function
Create a Completely Automated Public Turing test to tell Computers and Humans Apart (CAPTCHA) image	`captcha` action of the `cfimage` tag

The ColdFusion image

© 2014 Adobe Systems Incorporated. All rights reserved.

A ColdFusion_ image_ is a construct that is native to ColdFusion. The ColdFusion image contains image data that it reads from a source. The source can be an image file or another ColdFusion image, which is expressed as a ColdFusion image variable. The ColdFusion image variable lets you manipulate information dynamically in memory. Optionally, you can write a ColdFusion image to a file, to a database column, or to a browser.

The cfimage tag

You use the `cfimage` tag to create a ColdFusion image and as a shortcut to commonly performed image functions, such as resizing an image, adding a border to an image, and converting an image to a different file format. You can use the `cfimage` tag independently or in conjunction with `Image` functions. You can pass a ColdFusion image created with the `cfimage` tag to one or more `Image` functions to perform complex image manipulation operations. The following table summarizes the `cfimage` tag actions:

Action	Description
`border`	Creates a rectangular border around the outer edge of an image.
`captcha`	Creates a CAPTCHA image.
`convert`	Converts an image from one file format to another.
`info`	Creates a ColdFusion structure that contains information about the image, including the color model, height, width, and source of the image.
`read`	Reads an image from the specified local file path or URL. If you do not specify an action explicitly, ColdFusion uses `read` as the default value.
`resize`	Resizes the height and width of an image.
`rotate`	Rotates an image by degrees.
`write`	Writes the image to a file. You can use the `write` actio n to generate lower-resolution JPEG files. Also, use the `write` action to convert images to other file formats, such as PNG and GIF.
`writeToBrowser`	Writes one or more images directly to a browser. Use this action to test the appearance of a single image or write multiple images to the browser without saving the images to files.

For more information, see the cfimage tag in the *CFML Reference*.

Image functions

ColdFusion provides more than 50 `Image` functions that expand on the functionality of the `cfimage` tag. You can pass images created with the `cfimage` tag to `Image` functions or create images with the `ImageNew` function. The

© 2014 Adobe Systems Incorporated. All rights reserved.

following table groups the `Image` functions by category:

Category	Image functions
Verifying images and supported image formats	IsImage, IsImageFile, GetReadableImageFormats, GetWriteableImageFormats
Retrieving image information	ImageGetEXIFTag, ImageGetEXIFMetadata, ImageGetIPTCMetadata, ImageGetHeight, ImageGetIPTCTag, ImageGetWidth, ImageInfo, ImageGetMetadata
Reading, writing, and converting images	ImageGetBlob, ImageGetBufferedImage, ImageNew, ImageRead, ImageReadBase64, ImageWrite, ImageWriteBase64
Manipulating images	ImageAddBorder, ImageBlur, ImageCopy, ImageCrop, ImageFlip, ImageGrayscale, ImageNegative, ImageOverlay, ImagePaste, ImageResize, ImageRotate, ImageScaleToFit, ImageSharpen, ImageShear, ImageTranslate
Drawing lines, shapes, and text	ImageDrawArc, ImageDrawBeveledRect, ImageDrawCubicCurve, ImageDrawLine, ImageDrawLines, ImageDrawOval, ImageDrawPoint, ImageDrawQuadraticCurve, ImageDrawRect, ImageDrawRoundRect, ImageDrawText
Setting drawing controls	ImageClearRect, ImageRotateDrawingAxis, ImageSetAntialiasing, ImageSetBackgroundColor, ImageSetDrawingColor, ImageSetDrawingStroke, ImageSetDrawingTransparency, ImageShearDrawingAxis, ImageTranslateDrawingAxis, ImageXORDrawingMode

© 2014 Adobe Systems Incorporated. All rights reserved.

Creating ColdFusion images

The ColdFusion image contains image data in memory. Before you can manipulate images in ColdFusion, you create a ColdFusion image. The following table shows the ways to create a ColdFusion image:

Task	Functions and tags
Create a ColdFusion image from an existing image file.	`cfimage` tag or the `ImageNew` function
Create a blank image from scratch.	`ImageNew` function
Create a ColdFusion image from BLOB data in a database.	`ImageNew` function with the `cfquery` tag
Create a ColdFusion image from a binary object.	`cffile` tag with the `ImageNew` function
Create a ColdFusion image from a Base64 string.	`ImageReadBase64` function and the `ImageNew` functi on or the `cfimage` tag
Create a ColdFusion image from another ColdFusion image.	`ImageCopy` function with the `ImageWrite` function or the `Duplicate` function, or by passing the image to the `ImageNew` function or the `cfimage` tag

Using the cfimage tag

The simplest way to create a ColdFusion image with the `cfimage` tag is to specify the `source` attribute, which is the image file that ColdFusion reads, and the `name` attribute, which is the variable that defines the image in memory:

```
<cfimage source="../cfdocs/images/artgallery/jeff01.jpg" name="myImage">
```

You do not have to specify the `read` action because it is the default action. Specify the `name` attribute for the `read` action, which creates a variable that contains the ColdFusion image, for example, *myImage*.
You can pass the *myImage* variable to another `cfimage` tag or to `Image` functions. The following example shows how to specify a ColdFusion image variable as the source:

```
<cfimage source="#myImage#" action="write" destination="test_myImage.jpg">
```

The `write` action writes the file to the specified destination, which can be an absolute or relative path. The following example shows how to create a ColdFusion image from a URL and write it to a file on the local storage drive:

```
<cfimage source="http://www.google.com/images/logo_sm.gif" action="write"
  destination="c:\images\logo_sm.gif">
```

Specify the destination for the write action.

© 2014 Adobe Systems Incorporated. All rights reserved.

When you specify a `destination`, set the `overwrite` attribute to "`yes`" to write to the same file more than once. Otherwise, ColdFusion generates an error:

```
<cfimage source="#myImage#" action="write" destination="images/jeff01.jpg"
overwrite="yes">
```

Using the ImageNew function

You create a ColdFusion image with the `ImageNew` function the same way you define a ColdFusion variable. The following example creates a ColdFusion image variable named "myImage" from the jeff01.jpg source file:

```
<cfset myImage=ImageNew("../cfdocs/images/artgallery/jeff01.jpg")>
```

This line produces the same result as the following code:

```
<cfimage source="../cfdocs/images/artgallery/jeff01.jpg" name="myImage">
```

As with the `cfimage` tag, you can specify an absolute or relative path, a URL, or another ColdFusion image as the source. In the following example, ColdFusion reads a file from the local drive and passes it to the `ImageWrite` funct ion, which writes the image to a new file:

```
<cfset myImage=ImageNew("../cfdocs/images/artgallery/jeff01.jpg")>
<cfset ImageWrite(myImage,"myImageTest.png")>
```

The following code produces the same result:

```
<cfimage source="../cfdocs/images/artgallery/jeff01.jpg" name="myImage">
<cfimage source="#myImage#" action="write" destination="myImageTest.png">
```

Also, you can create a blank image. When using the `ImageNew` function, you do not specify the source to create a blank image. However, you can specify the width and height, respectively. The following example shows how to create a blank canvas that is 300 pixels wide and 200 pixels high:

```
<cfset myImage=ImageNew("",300,200)>
```

Optionally, you can specify the image type, as in the following example:

```
<cfset myImage=ImageNew("",200,300,"rgb")>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

Other valid image types are `argb` and `grayscale`. You can use blank images as canvasses for drawing functions in ColdFusion. For examples, see Creating watermarks.
Also, you can use the `ImageNew` function to create ColdFusion images from other sources, such as Base64 bytearrays, file paths, and URLs. The following example creates a ColdFusion image from a JPEG file (\x), and then creates another ColdFusion image (\y) from the image in memory:

```
<cfset x = ImageNew("c:\abc.jpg")>
<cfset y = ImageNew(x)>
```

For more information about the `ImageNew` function, see the *CFML Reference*.

Creating an image from a binary object

You can use the `cffile` tag to write an image file to ColdFusion variable. Then, you can pass the variable to the `ImageNew` function to create a ColdFusion image from the binary object, as the following example shows:

```
<!--- Use the cffile tag to read an image file, convert it to binary format, and
write the
 result to a variable. --->
<cffile action = "readBinary" file = "jeff05.jpg" variable = "aBinaryObj">
<!--- Use the ImageNew function to create a ColdFusion image from the variable. --->

<cfset myImage=ImageNew(aBinaryObj)>
```

Creating images from BLOB data

Many databases store images as BLOB data. To extract BLOB data from a database, create a query with the `cfquery` tag. The following example shows how to extract BLOB data and use the `cfimage` tag to write them to files in PNG format:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<!--- Use the cfquery tag to retrieve employee photos and last names from the
database. --->
<cfquery
 name="GetBLOBs" datasource="myblobdata">
 SELECT LastName,Image
 FROM Employees
</cfquery>
<cfset i = 0>
<table border=1>
 <cfoutput query="GetBLOBs">
 <tr>
  <td>
  #LastName#
  </td>
  <td>
  <cfset i = i+1>
<!--- Use the cfimage tag to write the images to PNG files. --->
  <cfimage source="#GetBLOBs.Image#" destination="employeeImage#i#.png"
   action="write">
  <img src="employeeImage#i#.png"/>
  </td>
 </tr>
</cfoutput>
</table>
```

The following example shows how to use the `ImageNew` function to generate ColdFusion images from BLOB data:

```
<!--- Use the cfquery tag to retrieve all employee photos and employee IDs from a
database. --->
<cfquery name="GetBLOBs" datasource="myBlobData">
SELECT EMLPOYEEID, PHOTO FROM Employees
</cfquery>
<!--- Use the ImageNew function to create a ColdFusion images from the BLOB data
that was
 retrieved from the database. --->
<cfset myImage = ImageNew(GetBLOBs.PHOTO)>
<!--- Create thumbnail versions of the images by resizing them to fit in a 100-pixel
square,
 while maintaining the aspect ratio of the source image. --->
<cfset ImageScaleToFit(myImage,100,100)>
<!--- Convert the images to JPEG format and save them to files in the thumbnails
subdirectory,
 using the employee ID as the filename. --->
<cfimage source="#myImage#" action="write"
 destination="images/thumbnails/#GetBLOBs.EMPLOYEID#.jpg">
```

For information on converting a ColdFusion image to a BLOB, see *Inserting an image as a BLOB in a database* in Converting images.

Creating an image from a Base64 string

Base64 is a way to describe binary data as a string of ASCII characters. Some databases store images in Base64

© 2014 Adobe Systems Incorporated. All rights reserved.

format rather than as BLOB data. You can use the `cfimage` tag or the `ImageReadBase64` function to read Base64 data directly from a database. Doing so eliminates the intermediary steps of binary encoding and decoding.
The following examples show how to use the `cfimage` tag to create a ColdFusion image from a Base64 string:

```
<!--- This example shows how to create a ColdFusion image from a Base64 string with
headers
 (used for display in HTML). --->
<cfimage source="....."
 destination="test_my64.jpeg" action="write" isBase64="yes">

<!--- This example shows how to use the cfimage tag to write a Base64 string without
headers. --->
<cfimage source="/9j/4AAQSkZJRgABAQA....." destination="test_my64.jpeg"
 action="write" isBase64="yes">
```

The following examples show how to use the `ImageReadBase64` function to create a ColdFusion image from a Base64 string:

```
<!--- This example shows how to use the ImageReadBase64 function to read a Base64
string
 with headers. --->
<cfset
myImage=ImageReadBase64(".....")>

<!--- This example shows how to read a Base64 string without headers. --->
<cfset myImage=ImageReadBase64("/9j/4AAQSkZJRgABAQA.....")>
```

For more information on Base64 strings, see *Converting an image to a Base64 string* in Converting images .

Copying an image

You use the `ImageCopy` function to copy a rectangular area of an existing image and generate a new ColdFusion image from it. You can paste the new ColdFusion image onto another image, or write it to a file, as the following example shows:

```
<!--- Use the cfimage tag to create a ColdFusion image from a JPEG file.
--->
<cfimage source="../cfdocs/images/artgallery/lori05.jpg" name="myImage">
<!--- Turn on antialiasing to improve image quality. --->
<cfset ImageSetAntialiasing(myImage)>
<!--- Copy the rectangular area specified by the coordinates (25,25,50,50) in the
image to
 the rectangle beginning at (75,75), and return this copied rectangle as a new
ColdFusion
 image. --->
<cfset dupArea = ImageCopy(myImage,25,25,50,50,75,75)>
<!--- Write the new ColdFusion image (dupArea) to a PNG file. --->
<cfimage source="#dupArea#" action="write" destination="test_myImage.png"
overwrite="yes">
```

© 2014 Adobe Systems Incorporated. All rights reserved.

Duplicating an image

Another way to create a ColdFusion image is to duplicate it. Duplicating an image creates a *clone*, which is a copy of an image that is independent of it: if the original image changes, those changes do not affect the clone, and the reverse. This technique is useful if you want to create several versions of the same image. Duplicating an image can improve processing time because you retrieve image data from a database or a file once to create the ColdFusion image. Then you can create several clones and manipulate them in memory before writing them to files. For example, you could create a thumbnail version, a grayscale version, and an enlarged version of an image uploaded to a server. To do so, you use the `cfimage` tag or the `ImageNew` function to create a ColdFusion image from the uploaded file. You use the `Duplicate` function to create three clones of the ColdFusion image.
To create a clone, you can pass a ColdFusion image variable to the `Duplicate` function:

```
<!--- Use the ImageNew function to create a ColdFusion image from a JPEG file. --->
<cfset myImage=ImageNew("../cfdocs/images/artgallery/paul01.jpg")>
<!--- Turn on antialiasing to improve image quality. --->
<cfset ImageSetAntialiasing(myImage)>
<!--- Use the Duplicate function to create three clones of the ColdFusion image.
--->
<cfset cloneA=Duplicate(myImage)>
<cfset cloneB=Duplicate(myImage)>
<cfset cloneC=Duplicate(myImage)>
<!--- Create a grayscale version of the image. --->
<cfset ImageGrayscale(cloneA)>
<!--- Create a thumbnail version of the image. --->
<cfset ImageScaleToFit(cloneB,50,"")>
<!--- Create an enlarged version of the image. --->
<cfset ImageResize(cloneC,"150%","")>
<!--- Write the images to files. --->
<cfset ImageWrite(myImage,"paul01.jpg","yes")>
<cfset ImageWrite(cloneA,"paul01_bw.jpg","yes")>
<cfset ImageWrite(cloneB,"paul01_sm.jpg","yes")>
<cfset ImageWrite(cloneC,"paul01_lg.jpg","yes")>
<!--- Display the images. --->
<img src="paul01.jpg">
<img src="paul01_bw.jpg">
<img src="paul01_sm.jpg">
<img src="paul01_lg.jpg">
```

Also, you can use the `cfimage` tag and the `ImageNew` function to duplicate images, as the following example shows:

```
<!--- Use the cfimage tag to create a ColdFusion image (myImage) and make a copy of
it (myImageCopy). --->
<cfimage source="../cfdocs/images/artgallery/paul01.jpg" name="myImage">
<cfimage source="#myImage#" name="myImageCopy">
<!-- Use the ImageNew function to make a copy of myImage called myImageCopy2. --->
<cfset myImageCopy2 = ImageNew(myImage)>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

Converting images

ColdFusion makes it easy to convert images from one file format to another. Also, you can convert an image file to a binary object, BLOB data, or a Base64 string.

Converting an image file

The extension of the destination file determines the file format of the image. Therefore, to convert an image, simply change the filename extension in the destination file. The following example shows how to convert a JPEG file to a GIF file:

```
<cfimage source="../cfdocs/images/artgallery/jeff01.jpg" action="write" destination="jeff01.gif"
>
```

Similarly, you can use the `ImageWrite` function with the `ImageNew` function:

```
<cfset myImage=ImageNew("../cfdocs/images/artgallery/jeff01.jpg")>
<cfset ImageWrite(myImage,"jeff01.gif")>
```

In both examples, the `convert` action is implied.
The `write` action does not create a ColdFusion image; it simply writes an image to a file. To convert an image and generate a ColdFusion image variable, use the `convert` action:

```
<cfimage source="../cfdocs/images/artgallery/jeff01.jpg" action="convert"
 destination="jeff01.gif" name="myImage">
```

ColdFusion reads and writes most standard image formats. For more information, see *Supported image file formats* in cfimage in the *CFML Reference*.

Converting an image to a Base64 string

To convert a ColdFusion image to a Base64 string, use the ImageWriteBase64 function. In the following example, the `yes` value determines that the output includes the headers required for display in HTML:

```
<!--- This example shows how convert a BMP file to a Base64 string. --->
<cfset ImageWriteBase64(myImage,"jeffBase64.txt","bmp","yes")>
```

Note

Microsoft Internet Explorer does not support Base64 strings.

Inserting an image as a BLOB in a database

Many databases store images as BLOB data. To insert a ColdFusion image into a BLOB column of a database, use the ImageGetBlob function within a `cfquery` statement, as the following example shows:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<!--- This example shows how to add a ColdFusion image to a BLOB column of a
database. --->
<!--- Create a ColdFusion image from an existing JPEG file. --->
<cfimage source="aiden01.jpg" name="myImage">
<!--- Use the cfquery tag to insert the ColdFusion image as a BLOB in the database.
--->
<cfquery name="InsertBlobImage" datasource="myBlobData">
INSERT into EMPLOYEES (FirstName,LastName,Photo)
VALUES ("Aiden","Quinn",<cfqueryparam value="#ImageGetBlob(myImage)#"
cfsqltype="cf_sql_blob">)
</cfquery>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

Verifying images

Use the `IsImage` function to test whether an image variable represents a valid ColdFusion image. This function takes a variable name as its only parameter and returns a Boolean value.

> **Note**
>
> You cannot use the `IsImage` function to verify whether files are valid images. Instead, use the `IsImageFile` function.

Also, ColdFusion provides two tags for determining which image file formats are supported on the server where the ColdFusion application is deployed: GetReadableImageFormats and GetWriteableImageFormats. For more information, see the *CFML Reference*.

© 2014 Adobe Systems Incorporated. All rights reserved.

Enforcing size restrictions

ColdFusion provides several functions for retrieving information associated with images, including the height and width of an image. For example, you can use the `ImageGetWidth` and `ImageGetHeight` functions to determine whether an image is too large to upload to a website or database.
The following example shows how to prevent users from uploading images that are greater than 300 pixels wide or 300 pixels high:

```
<!--- Create a ColdFusion image named "myImage" from a file uploaded to the server.
--->
<cfimage action="read" source="#fileUpload.serverFile#" name="myImage">
<!--- Determine whether the file is greater than 300 pixels in width or height. --->

<cfif ImageGetHeight(myImage) gt 300 or ImageGetWidth(myImage) gt 300>
 <!--- If the file exceeds the size limits, delete it from the server. --->
 <cffile action="delete"
file="#fileUpload.serverDirectory#/#fileUpload.serverFile#">
 <cfoutput>
<!--- Display the following message. --->
 <p>
 The image you uploaded was too big. It must be less than 300 pixels wide and 300
pixels
  high. Your image was #imageGetWidth(myImage)# pixels wide and
  #imageGetHeight(myImage)# pixels high.
 </p>
</cfif>
```

For information about retrieving image metadata, see the ImageGetEXIFTag, ImageGetIPTCTag, and ImageInfo functions in the *CFML Reference*.

© 2014 Adobe Systems Incorporated. All rights reserved.

Compressing JPEG images

To reduce the size of large files, you can convert a JPEG file to a lower quality image by using the `write` action of the `cfimage` tag. Specify a value between 0 (low) and 1 (high) for the `quality` attribute, as the following example shows:

```
<cfimage source="../cfdocs/images/artgallery/jeff05.jpg" action="write"
 destination="jeff05_lq.jpg" quality="0.5" overwrite="yes">
```

You can perform the same operation by using the `ImageWrite` function:

```
<cfset myImage=ImageNew("jeff05.jpg")>
<cfset ImageWrite(myImage,"jeff05_lq.jpg","0.5")>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

Manipulating ColdFusion images

You can perform a few common manipulation operations on ColdFusion images. For more information on manipulating ColdFusion images, see the *CFML Reference*.

Adding borders to images

To create a simple border, use the `cfimage` tag. The following example creates a ColdFusion image with a 5-pixel blue border:

```
<cfimage source="../cfdocs/images/artgallery/jeff01.jpg" action="border"
thickness="5"
 color="blue" destination="testMyImage.jpg" overwrite="yes">
<img src="testMyImage.jpg">
```

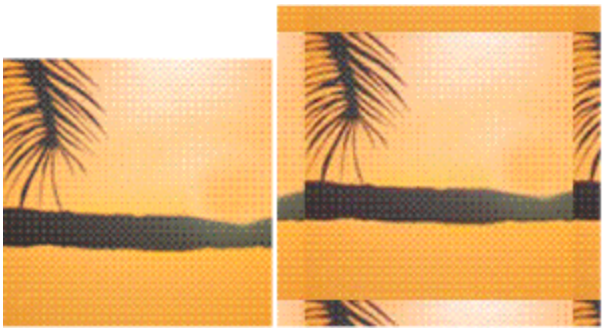
The border is added to the outside edge of the source image. This increases the area of the image.
To create complex borders, use the `ImageAddBorder` function. The following example shows how to nest borders:

```
<!--- Create a ColdFusion image from a JPEG file. --->
<cfset myImage=ImageNew("../cfdocs/images/artgallery/jeff01.jpg")>
<!--- Add a 5-pixel blue border around the outside edge of the image. --->
<cfset ImageAddBorder(myImage,5,"blue")>
<!--- Add a 10-pixel magenta border around the blue border. --->
<cfset ImageAddBorder(myImage,10,"magenta")>
<!--- Add a 5-pixel green border around the magenta border. --->
<cfset ImageAddBorder(myImage,20,"green")>
<!--- Write the ColdFusion image to a file. --->
<cfset ImageWrite(myImage,"testMyImage.jpg")>
<img src="testMyImage.jpg"/>
```

Also, with the `ImageAddBorder` function, you can add a border that is an image effect. For example, you can use the `wrap` parameter to create a tiled border from the source image. The `wrap` parameter creates a tiled border by adding the specified number of pixels to each side of the image, as though the image were tiled.
In the following example, 20 pixels from the outside edge of the source image are tiled to create the border:

```
<cfset myImage=ImageNew("../cfdocs/images/artgallery/jeff03.jpg")>
<cfset ImageAddBorder(myImage,20,"","wrap")>
<cfset ImageWrite(myImage,"testMyImage.jpg")>
<img src="testMyImage.jpg"/>
```

© 2014 Adobe Systems Incorporated. All rights reserved.



For examples of other border types, see the ImageAddBorder function in the *CFML Reference*.

Creating text images

You can create two types of text images:

- A CAPTCHA image, in which ColdFusion randomly distorts the text
- A text image, in which you control the text attributes

Creating a CAPTCHA image

You use the `captcha` action of the `cfimage` tag to create a distorted text image that is human-readable but not machine readable. When you create a CAPTCHA image, you specify the text that is displayed in the CAPTCHA image; ColdFusion randomly distorts the text. You can specify the height and width of the text area, which affects the spacing between letters, the font size, the fonts to use for the CAPTCHA text, and the level of difficulty, which affects readability. Do not use spaces in the text string specified for the `text` attribute: users cannot detect the spaces as part of the CAPTCHA image.
The following example shows how to write a CAPTCHA image directly to the browser.

```
<!--- This example shows how to create a CAPTCHA image with the text "rEadMe" and
write the
 image directly to the browser. --->
<cfimage action="captcha" fontSize="25" width="162" height="75" text="rEadMe"
 fonts="Verdana,Arial,Courier New,Courier">
```

Note

For the CAPTCHA image to display, the `width` value must be greater than: `fontSize` times the number of characters specified in `text` times 1.08. In this example, the minimum `width` is 162.

ColdFusion 9 supports CAPTCHA images in PNG format only.

Note

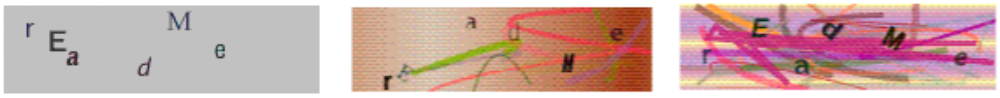
If you specify the `destination` attribute to write CAPTCHA images to files, use unique names for the CAPTCHA image files so that when multiple users access the CAPTCHA images, the files are not overwritten.

The following example shows how to create CAPTCHA images with a high level of text distortion.

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<!--- Use the GetTickCount function to generate unique names for the CAPTCHA files.
--->
<cfset tc = GetTickCount()>
<!--- Set the difficulty to "high" for a higher level of text distortion. --->
<cfimage action="captcha" fontSize="15" width="180" height="50" text="rEadMe"
 destination="readme#tc#.png" difficulty="high">
```

For a detailed example, see *Using CAPTCHA to verify membership* in Application examples that use ColdFusion images.
The following image shows three CAPTCHA images with low, medium, and high levels of difficulty, respectively:



Using the ImageDrawText function

To create a text image by using the `ImageDrawText` function, specify the text string and the x and y coordinates for the location of the beginning of the text string. You can draw the text on an existing image or on a blank image, as the following examples show:

```
<!--- This example shows how to draw a text string on a blank image. --->
<cfset myImage=ImageNew("",200,100)>
<cfset ImageDrawText(myImage, "Congratulations!",10,50)>
<cfimage source="#myImage#" action="write" destination="myImage.png"
overwrite="yes">
<img src="myImage.png">

<!--- This example shows how to draw a text string on an existing image.
--->
<cfset myImage2=ImageNew("../cfdocs/images/artgallery/jeff01.jpg")>
<cfset ImageDrawText(myImage2,"Congratulations!",10,50)>
<cfimage source="#myImage2#" action="write" destination="myImage2.png"
overwrite="yes">
<img src="myImage2.png">
```

In the previous examples, the text is displayed in the default system font and font size. To control the appearance of the text, you specify a collection of text attributes, as the following example shows:

```
<cfset attr = StructNew()>
<cfset attr.style="bolditalic">
<cfset attr.size=20>
<cfset attr.font="verdana">
<cfset attr.underline="yes">
```

To apply the text attributes to the text string, include the attribute collection name in the `ImageDrawText` definition. In the following examples, the `"attr"` text attribute collection applies the text string `"Congratulations!"`:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
...
<cfset ImageDrawText(myImage,"Congratulations!",10,50,attr)>
```

To change the color of the text, use the `ImageSetDrawingColor` function. This function controls the color of all subsequent drawing objects on an image. In the following example, two lines of text, "Congratulations!" and "Gabriella", inherit the color magenta.

```
<!--- This example shows how to draw a text string on a blank image. --->
<cfset myImage=ImageNew("../cfdocs/images/artgallery/jeff01.jpg")>
<cfset ImageSetDrawingColor(myImage,"magenta")>
<cfset attr = StructNew()>
<cfset attr.style="bolditalic">
<cfset attr.size=20>
<cfset attr.font="verdana">
<cfset attr.underline="yes">
<cfset ImageDrawText(myImage,"Congratulations!",10,50,attr)>
<cfset ImageDrawText(myImage,"Gabriella",50,125,attr)>
<cfimage source="#myImage#" action="write" destination="myImage.jpg"
overwrite="yes">
<img src="myImage.jpg"/>
```

For a list of valid named colors, see the cfimage tag in the *CFML Reference*.

Drawing lines and shapes

ColdFusion provides several functions for drawing lines and shapes. For shapes, the first two values represent the x and y coordinates, respectively, of the upper-left corner of the shape. For simple ovals and rectangles, the two numbers following the coordinates represent the width and height of the shape in pixels. For a line, the values represent the x and y coordinates of the start point and end point of the line, respectively. To create filled shapes, set the `filled` attribute to true. The following example shows how to create an image with several drawing objects:

```
<!--- Create an image that is 200-pixels square. --->
<cfset myImage=ImageNew("",200,200)>
<!--- Draw a circle that is 100 pixels in diameter. --->
<cfset ImageDrawOval(myImage,40,20,100,100)>
<!--- Draw a filled rectangle that is 40 pixels wide and 20 pixels high.
--->
<cfset ImageDrawRect(myImage,70,50,40,20,true)>
<!--- Draw a 100-pixel square. --->
<cfset ImageDrawRect(myImage,40,40,100,100)>
<!--- Draw two lines. --->
<cfset ImageDrawLine(myImage,130,40,100,200)>
<cfset ImageDrawLine(myImage,50,40,100,200)>
<!--- Write the ColdFusion image to a file. --->
<cfimage source="#myImage#" action="write" destination="testMyImage.gif"
overwrite="yes">
<img src="testMyImage.gif"/>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

> **Note**
>
> To draw a sequence of connected lines, use the `ImageDrawLines` function. For more information, see the CFML Reference.

Setting drawing controls

ColdFusion provides several functions for controlling the appearance of drawing objects. As shown in the `ImageDrawText` example, you use the `ImageSetDrawingColor` function to define the color of text in an image. This function also controls the color of lines and shapes. To control line attributes (other than color), use the `ImageSetDrawingStroke` function. The `ImageSetDrawingStroke` function uses a collection to define the line attributes. Drawing controls apply to all subsequent drawing functions in an image; therefore, order is important. In the following example, the drawing stroke attributes defined in the attribute collection apply to the square and the two lines. Similarly, the color green applies to the rectangle and the square, while the color red applies only to the two lines. You can reset a drawing control as many times as necessary within an image to achieve the desired effect.

```
<!--- Create an attribute collection for the drawing stroke. --->
<cfset attr=StructNew()>
<cfset attr.width="4">
<cfset attr.endcaps="round">
<cfset attr.dashPattern=ArrayNew(1)>
<cfset dashPattern[1]=8>
<cfset dashPattern[2]=6>
<cfset attr.dashArray=dashPattern>
<cfset myImage=ImageNew("",200,200)>
<cfset ImageDrawOval(myImage,40,20,100,100)>
<!--- Set the drawing color to green for all subsequent drawing functions. --->
<cfset ImageSetDrawingColor(myImage,"green")>
<cfset ImageDrawRect(myImage,70,50,40,20,true)>
<!--- Apply the attribute collection to all subsequent shapes and lines in the
image. --->
<cfset ImageSetDrawingStroke(myImage,attr)>
<cfset ImageDrawRect(myImage,40,40,100,100)>
<!--- Set the drawing color to red for all subsequent drawing functions. --->
<cfset ImageSetDrawingColor(myImage,"red")>
<cfset ImageDrawLine(myImage,130,40,100,200)>
<cfset ImageDrawLine(myImage,50,40,100,200)>
<cfimage source="#myImage#" action="write" destination="testMyImage.gif"
overwrite="yes">
<img src="testMyImage.gif"/>
```

Resizing images

ColdFusion makes it easy to resize images. You can reduce the file size of an image by changing its dimensions, enforce uniform sizes on images, and create thumbnail images. The following table describes the ways to resize images in ColdFusion:

Task	Functions and actions
Resize an image	`ImageResize` function, or the `resize` action of the `cfimage` tag

© 2014 Adobe Systems Incorporated. All rights reserved.

Resize images so that they fit in a defined square or rectangle and control the interpolation method	`ImageScaleToFit` function
Resize an image and control the interpolation method	`ImageResize` function

Using the cfimage tag resize action

Use the `cfimage` tag `resize` action to resize an image to the specified height and width. You can specify the height and width in pixels or as a percentage of the original dimensions of the image. To specify a percentage, include the percent symbol (%) in the height and width definitions.

```
<!--- This example shows how to specify the height and width of an image in pixels.
--->
<cfimage source="../cfdocs/images/artgallery/jeff01.jpg" action="resize" width="100"

 height="100" destination="jeff01_sm.jpg">
<!--- This example shows how to specify the height and width of an image as
percentages. --->
<cfimage source="../cfdocs/images/artgallery/jeff02.jpg" action="resize" width="50%"

 height="50%" destination="jeff02_sm.jpg">

<!--- This example shows how to specify the height of an image in pixels and its
width as a
 percentage. Notice that this technique can distort the image. --->
<cfimage source="../cfdocs/images/artgallery/jeff03.jpg" action="resize" width="50%"

 height="100" destination="jeff03_sm.jpg" overwrite="yes">
```

The `cfimage` tag requires that you specify both the height and the width for the `resize` action.
The `cfimage` tag `resize` action uses the `highestQuality` interpolation method for the best quality image (at the cost of performance). For faster display, use the `ImageResize` function or the `ImageScaleToFit` function.

Using the ImageResize function

The `ImageResize` function is like the `cfimage` tag `resize` action. To ensure that the resized image is proportional, specify a value for the height or width and enter a blank value for the other dimension:

```
<!--- This example shows how to resize an image to 50% of original size and resize
it
 proportionately to the new width. The height value is blank. --->
<cfset myImage=ImageNew("http://www.google.com/images/logo_sm.gif")>
<cfset ImageResize(myImage,"50%","")>
<!--- Save the modified image to a file. --->
<cfimage source="#myImage#" action="write" destination="test_myImage.jpeg"
overwrite="yes">
<!--- Display the source image and the resized image. --->
<img src="http://www.google.com/images/logo_sm.gif"/>
<img src="test_myImage.jpeg"/>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

The `ImageResize` function also lets you specify the type of interpolation used to resize the image. Interpolation lets you control the trade-off between performance and image quality. By default, the `ImageResize` function uses the `highestQuality` interpolation method. To improve performance (at the cost of image quality), change the interpolation method. Also, you can set the blur factor for the image. The default value is 1 (not blurred). The highest blur factor is 10 (very blurry). The following example shows how to resize an image using the `highPerformance` form of interpolation with a blur factor of 10:

```
<cfset myImage=ImageNew("../cfdocs/images/artgallery/aiden01.jpg")>
<cfset ImageResize(myImage,"","200%","highPerformance", 10)>
<cfimage action="writeToBrowser" source="#myImage#">
```

Note

Increasing the blur factor reduces performance.

For a complete list of interpolation methods, see ImageResize in the *CFML Reference*.

Using the ImageScaleToFit function

To create images of a uniform size, such as thumbnail images or images displayed in a photo gallery, use the `ImageScaleToFit` function. You specify the area of the image in pixels. ColdFusion resizes the image to fit the square or rectangle and maintains the aspect ratio of the source image. Like the `ImageResize` function, you can specify the interpolation, as the following example shows:

```
<!--- This example shows how to resize an image to a 100-pixel square, while
maintaining the aspect ratio of the source image. --->
<cfimage source="../cfdocs/images/artgallery/jeff05.jpg" name="myImage"
action="read">
<!--- Turn on antialiasing. --->
<cfset ImageSetAntialiasing(myImage)>
<cfset ImageScaleToFit(myImage,100,100,"mediumQuality")>
<!--- Display the modified image in a browser. --->
<cfimage source="#myImage#" action="writeToBrowser">
```

To fit an image in a defined rectangular area, specify the width and height of the rectangle, as the following example shows:

```
<!--- This example shows how to resize an image to fit in a rectangle that is 200
pixels
 wide and 100 pixels high, while maintaining the aspect ratio of the source image.
--->
<cfimage source="../cfdocs/images/artgallery/jeff05.jpg" name="myImage">
<!--- Turn on antialiasing. --->
<cfset ImageSetAntialiasing(myImage)>
<cfset ImageScaleToFit(myImage,200,100)>
<!--- Display the modified image in a browser. --->
<cfimage source="#myImage#" action="writeToBrowser">
```

© 2014 Adobe Systems Incorporated. All rights reserved.

In this example, the width of the resulting image is less than or equal to 200 pixels and the height of the image is less than or equal to 100 pixels.
Also, you can specify just the height or just the width of the rectangle. To do so, specify an empty string for the undefined dimension. The following example resizes the image so that the width is exactly 200 pixels and the height of the image is proportional to the width:

```
<!--- This example shows how to resizes an image so that it is 200 pixels wide,
while
 maintaining the aspect ratio of the source image. The interpolation method is set
to
 maximize performance (which reduces image quality). --->
<cfimage source="../cfdocs/images/artgallery/jeff05.jpg" name="myImage">
<!--- Turn on antialiasing. --->
<cfset ImageSetAntialiasing(myImage)>
<cfset ImageScaleToFit(myImage,200,"","highestPerformance")>
<!--- Display the modified image in a browser. --->
<cfimage source="#myImage#" action="writeToBrowser">
```

For more information, see ImageScaleToFit in the *CFML Reference*.

Creating watermarks

A *watermark* is a semitransparent image that is superimposed on another image. One use for a watermark is for protecting copyrighted images. To create a watermark in ColdFusion, you use the `ImageSetDrawingTransparency` function with the `ImagePaste` function. You can create a watermark image in one of three ways:

- Create a watermark from an existing image file. For example, you can use a company logo as a watermark.
- Create a text image in ColdFusion and apply the image as a watermark. For example, you can create a text string, such as *Copyright* or *PROOF* and apply it to all the images in a photo gallery.
- Create a drawing image in ColdFusion and use it as a watermark. For example, you can use the drawing functions to create a green check mark and apply it to images that have been approved.

Creating a watermark from an image file

The following example shows how to create a watermark from an existing GIF image located on a website:

```
<!--- This example shows how to create a watermark from an existing image. --->
<!--- Create two ColdFusion images from existing JPEG files. --->
<cfimage source="../cfdocs/images/artgallery/raquel05.jpg" name="myImage">
<cfimage source="http://www.google.com/images/logo_sm.gif" name="myImage2">
<cfimage source="#myImage#" action="write" destination="logo.jpg" overwrite="yes">
<cfset ImageSetDrawingTransparency(myImage,50)>
<!--- Paste myImage2 on myImage at the coordinates (0,0). --->
<cfset ImagePaste(myImage,myImage2,0,0)>
<!--- Write the result to a file. --->
<cfimage source="#myImage#" destination="watermark.jpg" action="write"
overwrite="yes">
<!--- Display the result. --->
<img src="watermark.jpg"/>
```

Creating a watermark from a text image

© 2014 Adobe Systems Incorporated. All rights reserved.

The following example shows how to create a text image in ColdFusion and use it as a watermark:

```
<!--- Create a ColdFusion image from an existing JPG file. --->
<cfset myImage=ImageNew("../cfdocs/images/artgallery/raquel05.jpg")>
<!--- Scale the image to fit in a 200-pixel square, maintaining the aspect ratio of
the
 source image. --->
 <cfset ImageScaleToFit(myImage,200,200)>
<!--- Set the drawing transparency to 75%. --->
 <cfset ImageSetDrawingTransparency(myImage,75)>
<!--- Create a ColdFusion image from scratch. --->
<cfset textImage=ImageNew("",150,140)>
<!--- Set the drawing color to white. --->
 <cfset ImageSetDrawingColor(textImage,"white")>
<!--- Create a collection of text attributes. --->
  <cfset attr=StructNew()>
  <cfset attr.size=40>
  <cfset attr.style="bold">
  <cfset attr.font="Arial">
<!--- Turn on antialiasing. --->
<cfset ImageSetAntialiasing(textImage)>
<!--- Draw the text string "PROOF" on the ColdFusion image. Apply the text
attributes that
 you specified. --->
<cfset ImageDrawText(textImage,"PROOF",1,75,attr)>
<!--- Rotate the text image by 30 degrees. --->
 <cfset ImageRotate(textImage,30)>
<!--- Scale the image to fit in a 200-pixel square, maintaining the aspect ratio of
the
 source image. --->
 <cfset ImageScaleToFit(textImage,200,200)>
<!--- Paste the text image onto myImage. --->
<cfset ImagePaste(myImage,textImage,0,0)>
<!--- Write the combined image to a file. --->
<cfimage source="#myImage#" action="write" destination="test_watermark.jpg"
overwrite="yes">
<!--- Display the image. --->
<img src="test_watermark.jpg"/>
```

Creating a watermark from a ColdFusion drawing

The following example shows how to draw an image in ColdFusion and use it as a watermark. You use the `ImageSetDrawingStroke` function to define the attributes of lines and shapes you create with drawing functions and the `ImageSetDrawingColor` function to define the color.

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<!--- This example shows how to draw a red circle with a line through it and use it
as a
 watermark. --->
<!--- Use the ImageNew function to create a ColdFusion image that is 201x201 pixels.
--->
<cfset myImage=ImageNew("",201,201)>
<!--- Set the drawing transparency of the image to 30%. --->
<cfset ImageSetDrawingTransparency(myImage,30)>
<!--- Set the drawing color to red. --->
<cfset ImageSetDrawingColor(myImage,"red")>
<!--- Create an attribute collection that sets the line width to ten pixels. --->
 <cfset attr=StructNew()>
 <cfset attr.width = 10>
<!--- Apply the attribute collection to the ImageSetDrawingStroke function. --->
 <cfset ImageSetDrawingStroke(myImage,attr)>
<!--- Draw a diagonal line starting at (40,40) and ending at (165,165) on myImage.
The drawing
 attributes you specified are applied to the line. --->
 <cfset ImageDrawLine(myImage,40,40,165,165)>
<!--- Draw a circle starting at (5,5) and is 190 pixels high and 190 pixels wide.
The drawing
 attributes you specified are applied to the oval. --->
<cfset ImageDrawOval(myImage,5,5,190,190)>
<!--- Create a ColdFusion image from a JPEG file. --->
<cfimage source="../cfdocs/images/artgallery/raquel05.jpg" name="myImage2">
<!--- Scale the image to fit in a 200-pixel square, maintaining the aspect ratio of
the
 source image. --->
<cfset ImageScaleToFit(myImage2,200,200)>
<!--- Paste the myImage2 directly over the myImage. --->
<cfset ImagePaste(myImage,myImage2,0,0)>
<!--- Save the combined image to a file. --->
<cfimage source="#myImage#" action="write" destination="test_watermark.jpg"
overwrite="yes">
<!--- Display the image in a browser. --->
<img src="test_watermark.jpg"/>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

Writing images to the browser

Use the `writeToBrowser` action of the `cfimage` tag to display images directly in the browser without writing them to files. This technique is useful for testing the appearance of a ColdFusion image. The following example shows how to test the display of two effects applied to an image:

```
<cfset myImage=ImageNew("../cfdocs/images/artgallery/paul01.jpg")>
<cfset ImageBlur(myImage,5)>
<cfset ImageNegative(myImage)>
<cfimage source="#myImage#" action="writeToBrowser">
```

The `writeToBrowser` action displays images in PNG format.
Also, you can write multiple images to the browser which is useful if you want to manipulate images in memory and display them without writing them to files. For example, you can duplicate several versions of the same image, display the versions in a browser, and allow the user to choose one of the images to write to a file. Or, you can extract images from a database, add a watermark to the images that appear in the browser, such as *Proof* or *Draft,* without having to write the modified images to files first. This way you can maintain one set of image files and change them on-the-fly. For an example of writing multiple images to the browser, see *Generating a gallery of watermarked images* in Application examples that use ColdFusion images.

© 2014 Adobe Systems Incorporated. All rights reserved.

Application examples that use ColdFusion images

You can create simple applications that automate image processes by using ColdFusion images.

Generating thumbnail images

The following example shows how to create a form for uploading images. A visitor to the site can use the form to upload an image file and generate a thumbnail image from it. You use ColdFusion image operations to perform the following tasks:

- Verify that the uploaded file is a valid image.
- Ensure that the height or the width of the image does not exceed 800 pixels.
- If the image is valid and within the size limits, generate a thumbnail version of the source image and save it to a file.
  Enter the following code on the form page:

```
<!--- This code creates a form with one field where the user enters the image file
to upload. --->
<cfform action="makeThumbnail.cfm" method="post" enctype="multipart/form-data">
Please upload an image: <cfinput type="file" name="image">
<cfinput type="submit" value="Send Image" name="Submit">
</cfform>
```

Enter the following code on the action page:

```
<cfset thisDir = expandPath(".")>
<!--- Determine whether the form is uploaded with the image. --->
<cfif structKeyExists(form,"image") and len(trim(form.image))>
 <!--- Use the cffile tag to upload the image file. --->
 <cffile action="upload" fileField="image" destination="#thisDir#"
result="fileUpload"
  nameconflict="overwrite">
 <!--- Determine whether the image file is saved. --->
 <cfif fileUpload.fileWasSaved>
 <!--- Determine whether the saved file is a valid image file. --->
  <cfif IsImageFile("#fileUpload.serverfile#")>
 <!--- Read the image file into a variable called myImage. --->
   <cfimage action="read" source="#fileUpload.serverfile#" name="myImage">
   <!--- Determine whether the image file exceeds the size limits. --->
   <cfif ImageGetHeight(myImage) gt 800 or ImageGetWidth(myImage) gt 800>
    <!--- If the file is too large, delete it from the server. --->
    <cffile action="delete"
     file="#fileUpload.serverDirectory#/#fileUpload.serverFile#">
    <cfoutput>
    <p>
    The image you uploaded was too large. It must be less than 800 pixels wide
     and 800 pixels high. Your image was #imageGetWidth(myImage)# pixels wide
     and #imageGetHeight(myImage)# pixels high.
    </p>
    </cfoutput>
    <!--- If the image is valid and does not exceed the size limits,
     create a thumbnail image from the source file that is 75-pixels
     square, while maintaining the aspect ratio of the source image.
     Use the bilinear interpolation method to improve performance.
```

© 2014 Adobe Systems Incorporated. All rights reserved.

```
      --->
    <cfelse>

  <cfset ImageScaleToFit(myImage,75,75,"bilinear")>
     <!--- Specify the new filename as the source filename with
      "_thumbnail" appended to it. --->
     <cfset newImageName = fileUpload.serverDirectory & "/" &
      fileUpload.serverFilename & "_thumbnail." &
      fileUpload.serverFileExt>
     <!--- Save the thumbnail image to a file with the new filename. --->
     <cfimage source="#myImage#" action="write"
      destination="#newImageName#" overwrite="yes">
     <cfoutput>
     <p>
     Thank you for uploading the image. We have created a thumbnail for
      your picture.
     </p>
     <p>
     <!--- Display the thumbnail image. --->
     <img src="#getFileFromPath(newImageName)#">
     </p>
     </cfoutput>
    </cfif>
   <!--- If it is not a valid image file, delete it from the server. --->
   <cfelse>
    <cffile action="delete"
     file="#fileUpload.serverDirectory#/#fileUpload.serverFile#">
    <cfoutput>
    <p>
    The file you uploaded, #fileUpload.clientFile#, was not a valid image.
    </p>
    </cfoutput>
   </cfif>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

```
 </cfif>
</cfif>
```

Generating a gallery of watermarked images

The following example extracts images and information from the `cfartgallery` database. You use ColdFusion image operations to perform the following tasks:

- Verify that an image exists for records returned from the database.
- Display the text, *SOLD!* on images that have been sold.
- Resize images to 100 pixels, maintaining the aspect ratio of the source image.
- Add a 5-pixel border to the images.
- Display the modified images directly in the browser without writing them to files.

Example

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<!--- Create a query to extract artwork and associated information from the
cfartgallery database. --->
<cfquery name="artwork" datasource="cfartgallery">
SELECT FIRSTNAME, LASTNAME, ARTNAME, DESCRIPTION, PRICE, LARGEIMAGE, ISSOLD,
MEDIATYPE
FROM ARTISTS, ART, MEDIA
WHERE ARTISTS.ARTISTID = ART.ARTISTID
AND ART.MEDIAID = MEDIA.MEDIAID
ORDER BY ARTNAME
</cfquery>
<cfset xctr = 1>
<table border="0" cellpadding="15" cellspacing="0" bgcolor="#FFFFFF">
<cfoutput query="artwork">
 <cfif xctr mod 3 eq 1>
 <tr>
  </cfif>
  <!--- Use the IsImageFile function to verify that the image files extracted
   from the   database are valid. Use the ImageNew function to create a
   ColdFusion image from valid image files. --->
  <cfif IsImageFile("../cfdocs/images/artgallery/#artwork.largeImage#")>
  <cfset myImage=ImageNew("../cfdocs/images/artgallery/#artwork.largeImage#")>
   <td valign="top" align="center" width="200">
   <cfset xctr = xctr + 1>
   <!--- For artwork that has been sold, display the text string "SOLD!"
    in white on the image. --->
     <cfif artwork.isSold>
      <cfset ImageSetDrawingColor(myImage,"white")>
       <cfset attr=StructNew()>
       <cfset attr.size=45>
       <cfset attr.style="bold">
       <cfset ImageDrawText(myImage,"SOLD!",35,195, attr)>
       </cfif>
  <!--- Resize myImage to fit in a 110-pixel square, scaled proportionately. --->
  <cfset ImageScaleToFit(myImage,110,"","bicubic")>
  <!--- Add a 5-pixel black border around the images. (Black is the default color.
--->
  <!--- Add a 5-pixel black border to myImage. --->
  <cfset ImageAddBorder(myImage,"5")>
  <!--- Write the images directly to the browser without saving them to the hard
drive.
   --->
  <cfimage source="#myImage#" action="writeToBrowser"><br>
   <strong>#artwork.artName#</strong><br>
   Artist: #artwork.firstName# #artwork.lastName#<br>
   Price: #dollarFormat(artwork.price)#<br>
   #artwork.mediaType# - #artwork.description#<br>
   </td>
  </cfif>
  <cfif xctr-1 mod 3 eq 0>
   </tr>
  </cfif>
</cfoutput>
</table>
```

Using CAPTCHA to verify membership

© 2014 Adobe Systems Incorporated. All rights reserved.

The following example shows how to create a simple form to verify whether a person (rather than a computer generating spam) is registering to receive an online newsletter. You generate the CAPTCHA image from a random text string on the form page and verify the response on the action page.

Example

Enter the following code on the form page:

```
<!--- Set the length of the text string for the CAPTCHA image. --->
<cfset stringLength=6>
<!--- Specify the list of characters used for the random text string. The following
list
 limits the confusion between upper- and lowercase letters as well as between
numbers and
 letters. --->
<cfset
 stringList="2,3,4,5,6,7,8,9,a,b,d,e,f,g,h,j,n,q,r,t,y,A,B,C,D,E,F,G,H,K,L,M,N,P,Q,R
,S,
 T,U,V,W,X,Y,Z">
<cfset rndString="">
<!--- Create a loop that builds the string from the random characters. --->
<cfloop from="1" to="#stringLength#" index="i">
 <cfset rndNum=RandRange(1,listLen(stringList))>
 <cfset rndString=rndString & listGetAt(stringList,rndNum)>
</cfloop>
<!--- Hash the random string. --->
<cfset rndHash=Hash(rndString)>

<!--- Create the user entry form. --->
<cfform action="captcha2.cfm" method="post">
<p>Please enter your first name:</p>
<cfinput type="text" name="firstName" required="yes">
<p>Please enter your last name:</p>
<cfinput type="text" name="lastName" required="yes">
<p>Please enter your e-mail address:</p>
<cfinput type="text" name="mailTo" required="yes" validate="email">
<!--- Use the randomly generated text string for the CAPTCHA image. --->
<p><cfimage action="captcha" fontSize="24" fonts="Times New Roman" width="200"
height="50"
 text="#rndString#"></p>
<p>Please type what you see: </p>
<p><cfinput type="text" name="userInput" required="yes" maxlength=6>
<cfinput type="hidden" name="hashVal" value="#rndHash#">
<p><cfinput type="Submit" name ="OK" value="OK"></p>
</cfform>
```

Enter the following code on the action page:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<!--- Verify whether the text entered by the user matches the CAPTCHA string. --->
<cfif #form.hashval# eq Hash(#form.userInput#)>
 <cfoutput>
 <p>
 Thank you for registering for our online newsletter, #form.firstName#
#form.lastName#.
 </p>
 <p>
 A notification has been sent to your e-mail address: #form.mailTo#.
 </p>
  <cfmail from="newsletter@domain.com" to="#form.mailTo#" subject="Newsletter">
  Thank you for your interest in our Newsletter.
  </cfmail>
 </cfoutput>
<cfelse>
 <p>I'm sorry; please try again.</p>
</cfif>
```

Creating versions of an image

The following example shows how to create an application that lets you generate four versions of the same image, display the versions in a form, and choose which one to save. The application comprises three ColdFusion pages that perform the following tasks:

- Dynamically populate a drop-down list box from a database query.
- Use the `cfimage` tag to create a ColdFusion image from the title selected from the list. Use the `ImageNew` fu nction to create three clones of the ColdFusion image. Use the `ImageSharpen` function to change the sharpness setting for each clone.
- Save the file chosen from the form to a new location.

Example

On the first form page, create a query that selects the artwork from the `cfartgallery` database and displays the titles in a pop-up menu:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<!--- Create a query to extract artwork from the cfartgallery database. --->
<cfquery name="artwork" datasource="cfartgallery">
SELECT ARTID, ARTNAME, LARGEIMAGE
FROM ART
ORDER BY ARTNAME
</cfquery>

<!--- Create a form that lists the artwork titles generated by the query. Set the
value to
 LARGEIMAGE so that the image file is passed to the action page. --->
<cfform action="dupImage2.cfm" method="post">
<p>Please choose a title:</p>
<cfselect name="art" query="artwork" display="ARTNAME" value="LARGEIMAGE"
required="yes"
 multiple="no" size="8">
</cfselect>
<br/><cfinput type="submit" name="submit" value="OK">
</cfform>
```

On the first action page, clone the original image three times, change the sharpness setting for each clone, and display the results:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<!--- Determine whether a valid image file exists. --->
<cfif IsImageFile("../cfdocs/images/artgallery/#form.art#")>
 <cfset original=ImageNew("../cfdocs/images/artgallery/#form.art#")>
 <!--- Use the ImageNew function to create a clone of the ColdFusion image. --->
 <cfset clone1=ImageNew(original)>
 <!--- Use the ImageSharpen function to blur the cloned image. --->
 <cfset ImageSharpen(clone1,-1)>
 <!--- Use the ImageNew function to create a second clone of the original image.
--->
 <cfset clone2=ImageNew(original)>
 <!--- Use the ImageSharpen function to sharpen the cloned image. --->
 <cfset ImageSharpen(clone2,1)>
 <!--- Use the ImageNew function to create a third clone for the original image.
--->
 <cfset clone3=ImageNew(original)>
 <!--- Use the ImageSharpen function to sharpen the cloned image to the maximum
setting.
  --->
 <cfset ImageSharpen(clone3,2)>
<!--- Create a form with a radio button for each selection. The value of the hidden
field
 is the relative path of the original image file. --->
<p>Please choose an image:</p>
<table>
 <tr>
 <cfform action="dupImage3.cfm" method="post">
  <td><cfimage source="#original#" action="writeToBrowser"><br />
  <cfinput type="radio" name="foo" value="original">Original Image</td>
  <td><cfimage source="#clone1#" action="writeToBrowser"><br />
  <cfinput type="radio" name="foo" value="blurred">Blurred Image</td>
  <td><cfimage source="#clone2#" action="writeToBrowser"><br />
  <cfinput type="radio" name="foo" value="sharper">Sharper Image</td>
  <td><cfimage source="#clone3#" action="writeToBrowser"><br />
  <cfinput type="radio" name="foo" value="sharpest">Sharpest Image</td>
  <tr><td><cfinput type="Submit" name="OK" value="OK">
  <cfinput type="hidden" name="orig_file"
   value="../cfdocs/images/artgallery/#form.art#">
  </td></tr>
 </cfform>
 </tr>
</table>
<cfelse>
 <p>There is no image associated with the title you selected. Please click the Back
button
  and try again.</p>
</cfif>
```

On the second action page, save the selected image to the C drive:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<p>The image you have chosen has been saved.</p>
<cfset img=ImageNew("#form.orig_file#")>
<cfswitch expression=#form.foo#>
 <cfcase value="blurred">
  <cfset ImageSharpen(img,-1)>
 </cfcase>
 <cfcase value="sharper">
  <cfset ImageSharpen(img,1)>
 </cfcase>
 <cfcase value="sharpest">
  <cfset ImageSharpen(img,2)>
 </cfcase>
</cfswitch>

<!--- Use the cfimage tag to write the image selected from the form to a file in the
C drive.
 Use the value of the hidden field as the source file for the image. --->
<cfimage source="#img#" action="write" destination="c:/myImage.jpg" overwrite="yes">

<img src="c:/myImage.jpg" />
```

#back to top

© 2014 Adobe Systems Incorporated. All rights reserved.

Creating Charts and Graphs

You can use the `cfchart` tag to display charts and graphs.

- About charts
- Creating a basic chart
- Charting data
- Controlling chart appearance
- Creating charts: examples
- Administering charts
- Writing a chart to a variable
- Linking charts to URLs
- Charting enhancements

© 2014 Adobe Systems Incorporated. All rights reserved.

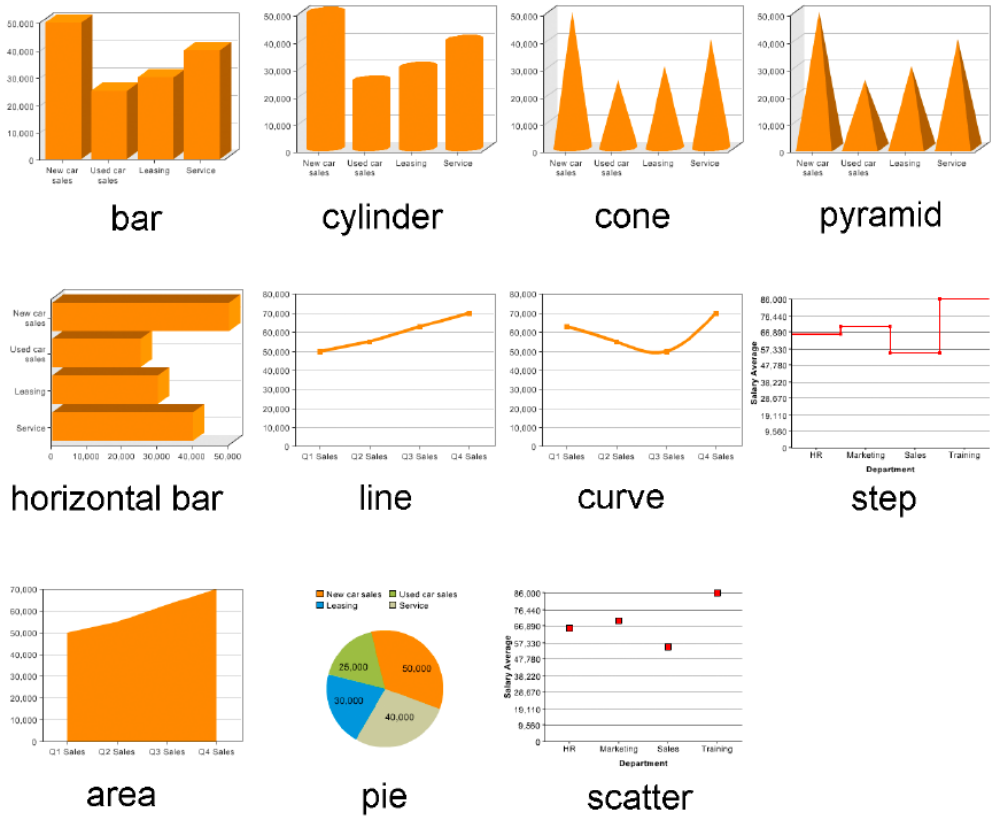
About charts

The ability to display data in a chart or graph can make data interpretation much easier. Rather than present a simple table of numeric data, you can display a bar, pie, line, or other applicable type of chart using colors, captions, and a two-dimensional or three-dimensional representation of your data.

The `cfchart` tag, along with the `cfchartseries` and `cfchartdata` tags, provide many different chart types. The attributes to these tags let you customize your chart appearance.

You can create 11 types of charts in Adobe ColdFusion in two and three dimensions. The following figure shows a sample of each type of chart.

> **Note**
>
> In two dimensions, bar and cylinder charts appear the same, as do cone and pyramid charts.



© 2014 Adobe Systems Incorporated. All rights reserved.

Creating a basic chart

You can create a chart in either of the following ways:

- Using the `cfchart`, `cfchartseries`, and `cfchartdata` tags in a ColdFusion page.
- Using the chart wizard that is included with the ColdFusion Report Builder. For more information, see Creating Reports and Documents for Printing.

Creating a chart with ColdFusion tags

To create a chart with ColdFusion tags, you use the `cfchart` tag along with at least one `cfchartseries` tag. You can optionally include one or more `cfchartdata` tags within a `cfchartseries` tag. The following table describes these tags:

Tag	Description
cfchart	Specifies the container in which the chart appears. This container defines the height, width, background color, labels, fonts, and other characteristics of the chart. Include at least one `cfchartseries` tag within the `cfchart` tag.
cfchartseries{{}}	Specifies a database query that supplies the data to the chart and one or more `cfchartdata` tags that specify individual data points. Specifies the chart type, colors for the chart, and other optional attributes.
cfchartdata{{}}	Optionally specifies an individual data point to the `cfchartseries` tag.

The following example shows an outline of the basic code that you use to create a chart:

```
<cfchart>
 <cfchartseries type="type">
   <cfchartdata item="something" value="number">
 </cfchartseries>
</chart>
```

The following example displays a simple pie chart that shows four values:

```
<cfchart>
 <cfchartseries type="pie">
  <cfchartdata item="New car sales" value="50000">
  <cfchartdata item="Used car sales" value="25000">
  <cfchartdata item="Leasing" value="30000">
  <cfchartdata item="Service" value="40000">
 </cfchartseries>
</cfchart>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

Note

If you access charting functionality using Java calls, a watermark ‘Developer Edition - Not for Production Use’ might be displayed on the chart images. To avoid this, when you assign the server object to a reference, instead of `svr.getDefaultInstance(getPageContext().getServletContext());)`, use the following code (by prefixing `svr=)` :`*svr=*svr.getDefaultInstance(getPageContext().getServletContext());)`. Ensure that you restart the server for the changes to take effect.

Creating a chart with the Report Builder wizard

The ColdFusion Report Builder includes a wizard that lets you create charts easily. The wizard lets you specify all of the chart characteristics that you can specify using the `cfchart`, `cfchartseries`, and `cfchartdata` tags. For information about using the Report Builder chart wizard, see Creating Reports and Documents for Printing.

© 2014 Adobe Systems Incorporated. All rights reserved.

Charting data

One of the most important considerations when you chart data is the way that you supply the data to the cfchart tag. You can supply data in the following ways:

- Specify individual data points by using cfchartdata tags.
- Provide all the data in a single query by using cfchartseries tags.
- Combine data from a query with additional data points from `cfchartdata` tags.
- Provide all the data in a report created with Report Builder. For more information, see Creating Reports and Documents for Printing.

> **Note**
>
> The `cfchart` tag charts numeric data only. As a result, convert any dates, times, or preformatted currency values, such as $3,000.53, to integers or real numbers.

Charting individual data points

When you chart individual data points, you specify each data point by inserting a `cfchartdata` tag in the `cfchart series` tag body. For example, the following code creates a simple pie chart:

```
<cfchart>
 <cfchartseries type="pie">
  <cfchartdata item="New Vehicle Sales" value=500000>
  <cfchartdata item="Used Vehicle Sales" value=250000>
  <cfchartdata item="Leasing" value=300000>
  <cfchartdata item="Service" value=400000>
 </cfchartseries>
</cfchart>
```

This pie chart displays four types of revenue for a car dealer. Each `cfchartdata` tag specifies the income for a department and a description for the legend.

> **Note**
>
> If two data points have the same item name, ColdFusion creates a graph of the value for the last one specified within the `cfchart` tag.

The `cfchartdata` tag lets you specify the following information about a data point:

Attribute	Description
`value`	The data value to chart. This attribute is required.
`item`	(Optional) The description for this data point. The item appears on the horizontal axis of bar and line charts, on the vertical axis of horizontal bar charts, and in the legend of pie charts.

Charting a query

© 2014 Adobe Systems Incorporated. All rights reserved.

Each bar, dot, line, or slice of a chart represents data from one row/column coordinate in your result set. A related group of data is called a chart series.
Because each bar, dot, line, or slice represents the intersection of two axes, craft the query result set such that the row and column values have meaning when displayed in a chart. Often, doing so requires you aggregate data in the query. You typically aggregate data in a query using one of the following:

- Specify a SQL aggregate function (SUM, AVG, MAX, and so on) using a GROUP BY clause in the SELECT statement.
- Use a Query of Queries.
- Retrieve data from a view, instead of a table.
  When you chart a query, you specify the query name using the `query` attribute of the `cfchartseries` tag. For example, the code for a simple bar chart could be as follows:

```
<cfchart
  xAxisTitle="Department"
  yAxisTitle="Salary Average"
 >
 <cfchartseries
  type="bar"
  query="DataTable"
  valueColumn="AvgByDept"
  itemColumn="Dept_Name"
  />
</cfchart>
```

This example displays the values in the AvgByDept column of the DataTable query. It displays the Dept_Name column value as the item label by each bar.
The following table lists the attributes of the `cfchartseries` tag that you use when working with queries:

Attribute	Description
`query`	The query that contains the data. Also specify the `valueColumn` and `itemColumn`.
`valueColumn`	The query column that contains the values to chart.
`itemColumn`	The query column that contains the description for this data point. The item normally appears on the horizontal axis of bar and line charts, on the vertical axis of horizontalbar charts, and in the legend in pie charts.

Charting a query of queries

In addition to charting the results of a query, you can also chart the results of a query of queries. For more information about using query of queries, see Using Query of Queries. Query of queries provides significant power in generating the data for the chart. For example, you can use aggregating functions such as SUM, AVG, and GROUP BY to create a query of queries with statistical data based on a raw database query. For more information, see Using Query of Queries.
You can also take advantage of the ability to dynamically reference and modify query data. For example, you can

© 2014 Adobe Systems Incorporated. All rights reserved.

loop through the entries in a query column and reformat the data to show whole dollar values.
The example in the following procedure analyzes the salary data in the cfdocexamples database using a query of queries, and displays the data as a bar chart.

1. Create a ColdFusion page with the following content:

© 2014 Adobe Systems Incorporated. All rights reserved.

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

<!--- Generate a query with statistical data for each department. --->
<cfquery dbtype = "query" name = "DeptSalaries">
 SELECT
  Dept_Name,
  AVG(Salary) AS AvgByDept
 FROM GetSalaries
 GROUP BY Dept_Name
</cfquery>

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

<html>
<head>
 <title>Employee Salary Analysis</title>
</head>

<body>
<h1>Employee Salary Analysis</h1>

<!--- Bar chart, from DeptSalaries Query of Queries. --->
<cfchart
  xAxisTitle="Department"
  yAxisTitle="Salary Average"
  font="Arial"
  gridlines=6
  showXGridlines="yes"
  showYGridlines="yes"
  showborder="yes"
  show3d="yes"
 >

 <cfchartseries
  type="bar"
  query="DeptSalaries"
  valueColumn="AvgByDept"
  itemColumn="Dept_Name"
  seriesColor="olive"
  paintStyle="plain"
 />
</cfchart>

<br>
</body>
</html>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

2. Save the page as chartdata.cfm in the myapps directory under the web root directory. For example, the directory path in Windows could be C:\Inetpub\wwwroot\myapps.
3. Return to your browser and enter the following URL to view the chartdata.cfm page:*http://localhost/myapps/chartdata.cfm*

> **Note**
>
> If a query contains two rows with the same value for the `itemColumn_` attribute, ColdFusion graphs the last row in the query for that value. For the preceding example, if the query contains two rows for the Sales department, ColdFusion graphs the value for the last row in the query for Sales._

Reviewing the code

The following table describes the code and its function:

Code	Description

`<cfquery name="GetSalaries" datasource="cfdocexamples">`
` SELECT Departmt.Dept_Name, Employee.Salary`
` FROM Departmt, Employee`
` WHERE Departmt.Dept_ID = Employee.Dept_ID`
`</cfquery>`
	Query the cfdocexamples database to get the Dept_Name and Salary for each employee. Because the Dept_Name is in the Departmt table and the Salary is in the Employee table, you need a table join in the WHERE clause. You can use the raw results of this query elsewhere on the page.

`<cfquery dbtype = "query" name = "DeptSalaries">`
` SELECT`
`  Dept_Name,`
`  AVG(Salary) AS AvgByDept`
` FROM GetSalaries`
` GROUP BY Dept_Name`
`</cfquery>`
	Generate a new query from the GetSalaries query. Use the AVG aggregating function to get statistical data on the employees. Use the GROUP BY statement to ensure that only one row exists for each department.

`<cfloop index="i" from="1" to="#DeptSalaries.RecordCount#">`
` <cfset DeptSalaries.AvgByDept[i]=Round(DeptSalaries.AvgByDept[i]/1000)*1000 >`
`</cfloop>`
	Loop through all the rows in the DeptSalaries query and round the salary data to the nearest thousand. This loop uses the RecordCount query variable to get the number of rows, and directly changes the contents of the query object.

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cfchart
  xAxisTitle="Department"
  yAxisTitle="Salary Average"
  font="Arial"
  gridlines=6
  showXGridlines="yes"
  showYGridlines="yes"
  showborder="yes"
  show3d="yes"
 >
 <cfchartseries
  type="bar"
  query="DeptSalaries"
  valueColumn="AvgByDept"
  itemColumn="Dept_Name"
  seriesColor="olive"
  paintStyle="plain"
 />
</cfchart>
```

Create a bar chart using the data from the AvgByDept column of the DeptSalaries query. Label the bars with the department names.

You can also rewrite this example to use the `cfoutput` and `cfchartdata` tags within the `cfchartseries` tag, instead of using the loop, to round the salary data, as the following code shows:

```
<cfchartseries
 type="bar"
 seriesColor="olive"
 paintStyle="plain">

 <cfoutput query="deptSalaries">
  <cfchartdata item="#dept_name#" value=#Round(AvgByDept/1000)*1000#>
 </cfoutput>

</cfchartseries>
```

Combining a query and data points

To chart data from both query and individual data values, you specify the query name and related attributes in the `cfchartseries` tag, and provide additional data points by using the `cfchartdata` tag.
ColdFusion displays the chart data specified by a `cfchartdata` tag before the data from a query, for example, to the left on a bar chart. You can use the `sortXAxis` attribute of the `cfchart` tag to sort data alphabetically along the x axis.
One use of combining queries and data points is to provide data that is missing from the database; for example, to provide the data for one department if the data for that department is missing. The example in the following procedure adds data for the Facilities and Documentation departments to the salary data obtained from the query shown in the previous section:

1. Open the chartdata.cfm file in your editor.
2. Edit the `cfchart`tag so that it appears as follows:

© 2014 Adobe Systems Incorporated. All rights reserved.

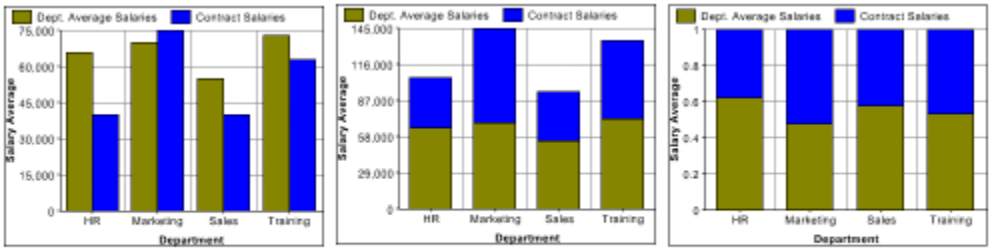
```
<cfchart chartwidth="600">
 <cfchartseries
  type="bar"
  query="DeptSalaries"
  itemColumn ="Dept_Name"
  valueColumn="AvgByDept"
  >
  <cfchartdata item="Facilities" value="35000">
  <cfchartdata item="Documentation" value="45000">
 </cfchartseries>
</cfchart>
```

3. Save the page as chartqueryanddata.cfm in the myapps directory under the web root directory. For example, the directory path in Windows could be C:\Inetpub\wwwroot\myapps.
4. Return to your browser and enter the following URL to view the chartqueryanddata.cfm page:*http://localhost/myapps/chartqueryanddata.cfm*

Charting multiple data collections

Sometimes, you could have more than one series of data to display on a single chart, or you want to compare two sets of data on the same chart. In some cases, you might want to use different charting types on the same chart. For example, you could include a line chart on a bar chart.
To combine multiple data series into a single chart, insert multiple `cfchartseries` tags within a single `cfchart` ta g. You control how the multiple data collections are charted using the `seriesPlacement` attribute of the `cfchart` tag. Using this attribute, you can specify the following options:

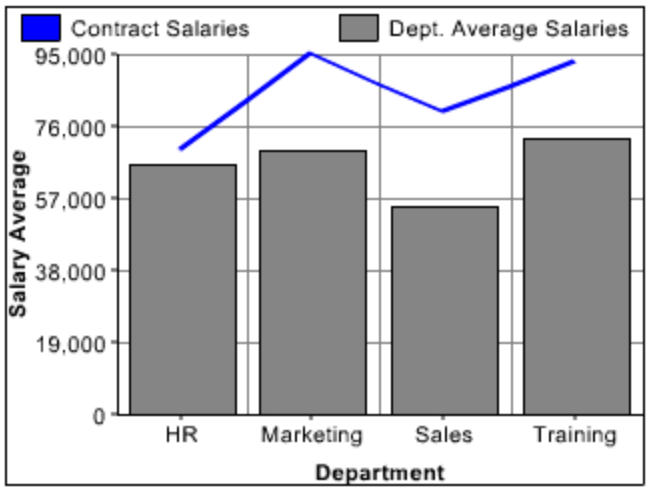
- **default** Let ColdFusion determine the best method for combining the data.
- **cluster** Place corresponding chart elements from each series next to each other.
- **stacked** Combine the corresponding elements of each series.
- **percent** Show the elements of each series as a percentage of the total of all corresponding elements. The following image shows these options for combining two bar charts:



cluster stacked percent

You can also combine chart types. The following is a combination bar and line chart:

© 2014 Adobe Systems Incorporated. All rights reserved.



The only chart type that you cannot mix with others is the pie chart. If you define one of the data series to use a pie chart, no other chart appears.

The example in the following procedure creates the chart in the previous figure, which shows a bar chart with a line chart added to it. In this example, you chart the salary of permanent employees (bar) against contract employees (line).

> **Note**
>
> The layering of multiple series depends on the order that you specify the `cfchartseries` tags. For example, if you specify a bar chart first and a line chart second, the bar chart appears in front of the line chart in the final chart.

Create a combination bar chart and line chart

1. Open the chartdata.cfm file in your editor.
2. Edit the `cfchart`tag so that it appears as follows:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cfchart
  backgroundColor="white"
  xAxisTitle="Department"
  yAxisTitle="Salary Average"
  font="Arial"
  gridlines=6
  showXGridlines="yes"
  showYGridlines="yes"
  showborder="yes"
 >
 <cfchartseries
  type="line"
  seriesColor="blue"
  paintStyle="plain"
  seriesLabel="Contract Salaries"
  >
   <cfchartdata item="HR" value=70000>
   <cfchartdata item="Marketing" value=95000>
   <cfchartdata item="Sales" value=80000>
   <cfchartdata item="Training" value=93000>
  </cfchartseries>
 <cfchartseries
  type="bar"
  query="DeptSalaries"
  valueColumn="AvgByDept"
  itemColumn="Dept_Name"
  seriesColor="gray"
  paintStyle="plain"
  seriesLabel="Dept. Average Salaries"
  />

</cfchart>
```

3. Save the file as chart2queries.cfm in the myapps directory under the web root directory.
4. Return to your browser and view the chart2queries.cfm page.

© 2014 Adobe Systems Incorporated. All rights reserved.

Controlling chart appearance

You can control the appearance of charts by doing any of the following:

- Using the default chart styles included with ColdFusion
- Using the attributes of the cfchart and cfchartseries tags
- Creating your own chart styles

Using the default chart styles included with ColdFusion

ColdFusion supplies the following chart styles:

- beige
- blue
- default
- red
- silver
- yellow
  To use any of these styles, specify the style using the `style` attribute of the `cfchart` tag. The following example illustrates using the `beige` style:

```
<cfchart style="beige">
 <cfchartseries type="pie">
  <cfchartdata item="New car sales" value="50000">
  <cfchartdata item="Used car sales" value="25000">
  <cfchartdata item="Leasing" value="30000">
  <cfchartdata item="Service" value="40000">
 </cfchartseries>
</cfchart>
```

You can specify the appearance of charts by using the attributes of the `cfchart` and `cfchartseries` tags. You can optionally specify the following characteristics to the `cfchart` tag on the types of charts indicated:

Chart characteristic	Attributes used	Description	Chart type
File type	`format`	Whether to send the chart to the user as a JPEG, PNG, or SWF file. The SWF file is the default format.	All
Size	`chartWidth` `chartHeight`	The width and height, in pixels, of the chart. This size defines the entire chart area, including the legend and background area around the chart.The default height is 240 pixels; the default width is 320 pixels.	All

© 2014 Adobe Systems Incorporated. All rights reserved.

Color	`foregroundColor`
	`dataBackgroundColor`
	`backgroundColor`
		The colors used for foreground and background objects. The default foreground color is black; the default background colors are white. You can specify 16 color names, use any valid HTML color format, or specify an 8-digit hexadecimal value to specify the RGB value and transparency. If you use numeric format, use double number signs; for example, blue or ##FF33CC. To specify the color and transparency, use the format ##xxFF33CC, where xx indicates the transparency. The value FF indicates opaque; the value 00 indicates transparent. For the complete list of colors, see *Configuring and Administering ColdFusion*.	All

© 2014 Adobe Systems Incorporated. All rights reserved.

Labels	font fontSize fontBold frontItalic labelFormat xAxisTitle yAxisTitle	The font attribute specifies the font for all text. The default value is Arial. If you are using a double-byte character set on UNIX, or using a double-byte character set in Windows with a file type of Flash, specify ArialUnicodeMs as the font.**Note:** If a chart attempts to use a font that is not installed on the ColdFusion server, it uses a different font that is available. Also, if you do not specify the font, characters that are not ASCII, such as Japanese, Chinese, Korean, and so on, can display improperly.The fontSize specifies an Integer font size used for all text. The default value is 11.The fontBold attribute specifies to display all text as bold. The default value is no.The fontItalic attribute specifies to display all text as italic. The default value is no.The labelFormat attribute specifies the format of the y-axis labels, number, currency, percent, or date. The default value is number.The xAxisTitle and yAxisTitle attributes specify the title for each axis.	All
Border	showBorder	Use the showBorder attribute to draw a border around the chart. The foregroundColor attribute specifies the border color. The default value is no.	All

© 2014 Adobe Systems Incorporated. All rights reserved.

Grid lines	`showXGridlines`
	`showYGridlines`
	`gridLines`
		Use the `showXGridlines` and `showYGridlines` attributes to display x-axis and y-axis grid lines. The default value `no` for x-axis gridlines, and `yes` for y-axis gridlines.The `gridLines` attribute specifies the total number of grid lines on the value axis, including the axis itself. The value of each grid line appears along the value axis. The `cfchart` tag displays horizontal grid lines only. The default value is 0, which means no grid lines.	AreaBarConeCurveCylinderHorizontalbarLinePyramidScatterStep

Slice style	`pieSliceStyle`	Displays the pie chart as solid or sliced. The default value is `sliced`.	Pie
Markers	`showMarkers` `markerSize`	The `showMarkers` attribute displays markers at the data points for two-dimensional line, curve, and scatter charts. The default value is `yes`.The `markerSize` attribute specifies an integer number of pixels for the marker size. ColdFusion determines the default value.	All
Value axis	`scaleFrom` `scaleTo`	The minimum and maximum points on the data axis. By default, the minimum is 0 or the lowest negative chart data value, and the maximum is the largest data value.*Note:* If you specify a `scaleFrom` or `scaleTo` attribute that would result in cropping the chart, `cfchart` uses a value that shows the entire chart without cropping.	AreaBarConeCurveCylinderHorizontalbarLinePyramidScatterStep

© 2014 Adobe Systems Incorporated. All rights reserved.

Axis type	`XAxisType` `sortXAxis`	Whether the x axis corresponds to a numeric scale or identifies different categories, and how to sort the items on the axis.If the `XAxisType` attribute value is `scale`, the x axis is numeric. All `cfchartdata item` attribute values must be numeric, and the axis is automatically sorted numerically. The `scale` value lets you create graphs of numeric relationships, such as population against age.If the attribute value is `category` (the default), the axis indicates the data category. The `sortXAxis` attribute determines the order of items when you specify the `cfchartdata item` attribute, whose values are treated as text. By default, the items are displayed in the order in which they are entered in the first chart series.	AreaBarConeCurveCylinderHorizontalbarLinePyramidScatterStep

3D appearance	`show3D`
	`xOffset`
	`yOffset`
		The `show3D` attribute displays the chart in three dimensions. The default value is `no`.The `xOffset` and `yOffset` attributes specify the amount by which to rotate the chart on a horizontal axis (`xOffset`) or vertical axis (`yOffset`). The value 0 is flat (no rotation), -1 and 1 are for a full 90 degree rotation left.(`-1`) or right (`1`). The default value is 0.1{{}}	All

© 2014 Adobe Systems Incorporated. All rights reserved.

Multiple series	`showLegendseriesPlacement`	The `showLegend` attribute lets you display the chart legend when the chart contains more than one series of data. The default value is `Yes`.The `seriesPlacement` attribute specifies the location of each series relative to the others. By default, ColdFusion determines the best placement based on the graph type of each series.	All
Tips	`tipStyle` `tipBGColor`	The `tipStyle` attribute lets you display a small pop-up window that shows information about the chart element pointed to by the mouse pointer. Options are `none`, `mousedown`, or `mouseover`. The default value is `mouseover`.The `tipBGColor` attribute specifies the background color of the tip window for Flash format only. The default value is `white`.	All

You can also use the `cfchartseries` tag to specify attributes of chart appearance. The following table describes these attributes:

Chart characteristic	Attributes used	Description	Chart type
Multiple series	`seriesLabel` `seriesColor`	The `seriesLabel` attribute specifies the text that displays for the series label.The `seriesColor` attribute specifies a single color of the bar, line, pyramid, and so on. For pie charts, the color is that of the first slice. Subsequent slices are automatically colored based on the specified initial color, or use the `colorList` attribute.	All

© 2014 Adobe Systems Incorporated. All rights reserved.

Paint	`paintStyle`	Specifies the way color is applied to a data series. You can specify solid color, raised button, linear gradient fill with a light center and darker outer edge, and gradient fill on lighter version of color. The default value is `solid`.	All
Data point colors	colorList	A comma-separated list of colors to use for each data point for bar, pyramid, area, horizontalbar, cone, cylinder, step, and pie charts.You can specify 16 color names, use any valid HTML color format, or specify an 8-digit hexadecimal value to specify the RGB value and transparency. If you use numeric format, use double number signs; for example, blue or ##FF33CC. To specify the color and transparency, use the format ##xxFF33CC, where xx indicates the transparency. The value FF indicates opaque; the value 00 indicates transparent. For the complete list of colors, see *Configuring and Administering ColdFusion*. If you specify fewer colors than data points, the colors repeat. If you specify more colors than data points, the extra colors are not used.	Pie

© 2014 Adobe Systems Incorporated. All rights reserved.

Data markers	markerStyle	Specifies the shape used to mark the data point. Shapes include `circle`, `diamond`, `letterx`, `mcross`, `rcross`, `rectangle`, `snow`, and `triangle`. Supported for two-dimensional charts. The default value is `rectangle`.	CurveLineScatter
Labels	dataLabelStyle	Specifies the way in which the color is applied to the item in the series Styles include `None`, `Value`, `Rowlabel`, `Columnlabel`, and `Pattern`.	All

Creating your own chart styles

You can create your own chart styles by doing either of the following:

- Modifying the chart style XML files
- Using WebCharts3D to create chart styles

Modifying the chart style XML files

You can modify the chart styles included with ColdFusion to create your own chart styles. The files that contain the style information are XML files located in the *cf_root*\charting\styles directory. Modify only attributes specified in the file. To specify additional attributes, follow the instructions in the section *Using WebCharts3D to create chart styles* below.

> **Note**
>
> Two XML files exist for each default chart style. For example, the beige style for pie charts is defined in the beige_pie.xml file; the beige style for all other types of charts is defined in the beige.xml file.

1. Open the XML file that you want to modify, for example beige.xml.
2. Modify the file contents.
3. Save the file with a different name; for example myBeige.xml.

Using WebCharts3D to create chart styles

Starting with ColdFusion MX 7, ColdFusion includes the WebCharts3D utility, which you can use to create chart style files.

1. Start WebCharts3D by double-clicking the webcharts.bat file in the *CFusion*\charting directory.
2. (Optional) Open an existing chart.
3. Make the changes you want to the appearance of the chart.

© 2014 Adobe Systems Incorporated. All rights reserved.

> **Note**
>
> To use the chart style file in the `cfchart_` tag, you can only make the modifications indicated in the table that follows this procedure._

4. Click the XML style tab.
5. Click the Save button in the lower-right corner.
6. Specify the name of the file; for example, mystyle.xml.
7. Specify the directory in which you want to save the chart style file.

> **Note**
>
> ColdFusion uses the same rules to look for the chart style XML files as it does for files included using the `cfinclude` tag. For more information, seecfinclude.

8. Click Save.
   The following table lists the attributes of the `cfchart` and `cfchartseries` tags and the associated WebCharts3D commands:

Attribute	WebCharts3D command
`chartHeight`	Drag the chart by handles.
`chartWidth`	Drag the chart by handles.
`dataBackgroundColor`	Background: minColor (type must be PlainColor)
`font`	font: Family (specify only supported fonts)
`fontBold`	font: check Bold
`fontItalic`	font: check Italic
`fontSize`	font: Size
`foregroundColor`	foreground
`gridlines`	X-axis: labelcount
`labelFormat`	Y-axis: LabelFormat: Number | Percent | Currency | Datetime
`markerSize`	Elements: markerSize
`pieSliceStyle`	style: solid | slice
`rotated`	Type Frame chart: Elements: Shape:

© 2014 Adobe Systems Incorporated. All rights reserved.

`scaleFrom`	Yaxis: isAbsolute; scaleMin(int)
`scaleTo`	Yaxis: isAbsolute; scaleMax(int)
`seriesPlacement`	Elements: place
`show3D`	is3D
`showBorder`	Decoration: style (none or simple)?
`showLegend`	Legend: isVisible
`showMarkers`	Elements: showMarkers
`showXGridlines`	Frame: isVGridVisible
`showYGridlines`	Frame: isHGridVisible
`tipbgColor`	Popup: background
`tipStyle`	Popup: show on MouseOver | show on MouseDown | Disabled
`url`	Elements: action Series: action
`xAxisTitle`	X-axis: TitleStyle: text (enter text)
`xAxisType`	X-axis: type: (category or scale)
`xOffset`	Frame: xDepth
`yAxisTitle`	Y-axis: TitleStyle: text (enter text)
`yAxisType`	Currently has no effect.
`yOffset`	Frame: yDepth

The following table lists the attributes of the `cfchartseries` tag and the associated WebCharts3D commands:

Attribute	WebCharts3D command
colorlist	Elements: series: Paint: color
markerStyle	Elements: series: Marker type: Rectangle | Triangle | Diamond | Circle | Letter | MCROSS | Snow | RCROSS
paintStyle	Paint: paint: Plain | Shade | Light

© 2014 Adobe Systems Incorporated. All rights reserved.

seriesColor	Elements: series: Paint: color
seriesLabel	Elements: series:
type	Type: Pie chart Type Frame chart: Elements: Shape: Bar | Line | Pyramid | Area | Curve | Step | Scatter | Cone | Cylinder | Horizontalbar

© 2014 Adobe Systems Incorporated. All rights reserved.

Creating charts: examples

Creating a bar chart

The example in the following procedure adds a title to the bar chart, specifies that the chart is three-dimensional, adds grid lines, sets the minimum and maximum y-axis values, and uses a custom set of colors.

1. Open the chartdata.cfm file in your editor.
2. Edit the `cfchart`tag so that it appears as follows:

```
<!--- Bar chart, from Query of Queries --->
<cfchart
  scaleFrom=40000
  scaleTo=100000
  font="arial"
  fontSize=16
  gridLines=4
  show3D="yes"
  foregroundcolor="##000066"
  databackgroundcolor="##FFFFCC"
  chartwidth="450"
 >

 <cfchartseries
  type="bar"
  query="DeptSalaries"
  valueColumn="AvgByDept"
  itemColumn="Dept_Name"
  seriescolor="##33CC99"
  paintstyle="shade"
  />

</cfchart>
```

3. Save the file as chartdatastyle1.cfm.
4. View the chartdatastyle1.cfm page in your browser.

Reviewing the code

The following table describes the code in the preceding example.

Code	Description
`scaleFrom=40000`	Set the minimum value of the vertical axis to 40000.
`scaleTo=100000`	Set the maximum value of the vertical axis to 100000. The minimum value is the default, `0`.
`font="arial"`	Displays text using the Arial font.
`fontSize=16`	Makes the point size of the labels 16 points.

© 2014 Adobe Systems Incorporated. All rights reserved.

`gridLines = 4`	Displays four grid lines between the top and bottom of the chart.
`show3D = "yes"`	Shows the chart in 3D.
`foregroundcolor="##000066"`	Sets the color of the text, gridlines, and labels.
`databackgroundcolor="##FFFFCC"`	Sets the color of the background behind the bars.
`seriescolor="##33CC99"`	Sets the color of the bars.
`paintstyle="shade"`	Sets the paint display style.

Creating a pie chart

The example in the following procedure adds a pie chart to the page.

1. Open the chartdata.cfm file in your editor.
2. Edit the DeptSalaries query and the `cfloop`code so that it appears as follows:

```
<!--- A query to get statistical data for each department. --->
<cfquery dbtype = "query" name = "DeptSalaries">
 SELECT
  Dept_Name,
  SUM(Salary) AS SumByDept,
  AVG(Salary) AS AvgByDept
 FROM GetSalaries
 GROUP BY Dept_Name
</cfquery>

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

3. Add the following `cfchart`tag:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<!--- Pie chart, from DeptSalaries Query of Queries. --->
<cfchart
  tipStyle="mousedown"
  font="Times"
  fontsize=14
  fontBold="yes"
  backgroundColor = "##CCFFFF"
  show3D="yes"
  >

 <cfchartseries
  type="pie"
  query="DeptSalaries"
  valueColumn="SumByDept"
  itemColumn="Dept_Name"
  colorlist="##6666FF,##66FF66,##FF6666,##66CCCC"
  />
</cfchart>
<br>
```

4. Save the file as chartdatapie1.cfm.
5. View the chartdatapie1.cfm page in your browser:

Reviewing the code

The following table describes the code and its function:

Code	Description
`SUM(Salary) AS SumByDept,`	In the DeptSalaries query, add a SUM aggregation function to get the sum of all salaries per department.
`<cfset DeptSalaries.SumByDept[i]= Round(DeptSalaries.SumByDept[i]/ 1000)*1000>`	In the cfloop tag, round the salary sums to the nearest thousand.
`<cfchart tipStyle="mousedown" font="Times" fontsize=14 fontBold="yes" backgroundColor = "##CCFFFF" show3D="yes" >`	Show a tip only when a user clicks the chart, display text in Times bold font, set the background color to light blue, and display the chart in three dimensions.

© 2014 Adobe Systems Incorporated. All rights reserved.

`<cfchartseries type="pie" query="DeptSalaries" valueColumn="SumByDept" itemColumn="Dept_Name" colorlist="##6666FF,##66FF66,##FF6666,##66CCCC" />`	Create a pie chart using the SumByDept salary sum values from the DeptSalaries query.Use the contents of the Dept_Name column for the item labels displayed in the chart legend.Get the pie slice colors from a custom list, which uses hexadecimal color numbers. The double number signs prevent ColdFusion from trying to interpret the color data as variable names.

Creating an area chart

The example in the following procedure adds an area chart to the salaries analysis page. The chart shows the average salary by start date to the salaries analysis page. It shows the use of a second query of queries to generate a new analysis of the raw data from the GetSalaries query. It also shows the use of additional `cfchart` attributes.

1. Open the chartdata.cfm file in your editor.
2. Edit the GetSalaries query so that it appears as follows:

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

3. Add the following code before the `htmltag`:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<!--- Convert start date to start year. --->
<!--- Convert the date to a number for the query to work --->
<cfloop index="i" from="1" to="#GetSalaries.RecordCount#">
<cfset GetSalaries.StartDate[i]=
NumberFormat(DatePart("yyyy", GetSalaries.StartDate[i]) ,9999)>
</cfloop>

<!--- Query of Queries for average salary by start year. --->
<cfquery dbtype = "query" name = "HireSalaries">
 SELECT
  StartDate,
  AVG(Salary) AS AvgByStart
 FROM GetSalaries
 GROUP BY StartDate
</cfquery>

<!--- Round average salaries to thousands. --->
<cfloop index="i" from="1" to="#HireSalaries.RecordCount#">
 <cfset HireSalaries.AvgByStart[i]=
 Round(HireSalaries.AvgByStart[i]/1000)*1000>
</cfloop>
```

4. Add the following `cfchart` tag before the end of the `body`tag block:

```
<!--- Area-style Line chart, from HireSalaries Query of Queries. --->
<cfchart
  chartWidth=400
  BackgroundColor="##FFFF00"
  show3D="yes"
 >
 <cfchartseries
  type="area"
  query="HireSalaries"
  valueColumn="AvgByStart"
  itemColumn="StartDate"
  />
</cfchart>
<br>
```

5. Save the page.
6. View the chartdata.cfm page in your browser.

Reviewing the code

The following table describes the code and its function:

Code	Description

© 2014 Adobe Systems Incorporated. All rights reserved.

`Employee.StartDate,`	Add the employee start date to the data in the GetSalaries query.

`<cfloop index="i" from="1" to="#GetSalaries.RecordCount#">`
`<cfset GetSalaries.StartDate[i]=NumberFormat(DatePart("yyyy", GetSalaries.StartDate[i]) ,9999)>`
`</cfloop>`
	Use a `cfloop` tag to extract the year of hire from the hire data, and convert the result to a four-digit number.

`<cfquery dbtype = "query" name = "HireSalaries">`
`  SELECT`
`  StartDate,`
`  AVG(Salary) AS AvgByStart`
`  FROM GetSalaries`
`  GROUP BY StartDate`
`</cfquery>`
	Create a second query from the GetSalaries query. This query contains the average salary for each start year.

`<cfloop index="i" from="1" to="#HireSalaries.RecordCount#">`
`  <cfset HireSalaries.AvgByStart[i]=Round(HireSalaries.AvgByStart[i]/1000)*1000>`
`</cfloop>`
	Round the salaries to the nearest thousand.

`<cfchart`
`  chartWidth=400`
`  BackgroundColor="##FFFF00"`
`  show3D="yes"`
`  >`
`  <cfchartseries`
`  type="area"`
`  query="HireSalaries"`
`  valueColumn="AvgByStart"`
`  itemColumn="StartDate"`
`  />`
`</cfchart>`
	Create an area chart using the HireSalaries query. Chart the average salaries against the start date.Limit the chart width to 400 pixels, show the chart in three dimensions, and set the background color to white.

© 2014 Adobe Systems Incorporated. All rights reserved.

Setting curve chart characteristics

Curves charts use the attributes already discussed. However, curve charts require a large amount of processing to render. For fastest performance, create them offline, write them to a file or variable, and then reference them in your application pages. For information on creating offline charts, see Writing a chart to a variable.

© 2014 Adobe Systems Incorporated. All rights reserved.

Administering charts

Use the ColdFusion Administrator to administer charts. In the Administrator, you can choose to save cached charts in memory or to disk. You can also specify the number of charts to cache, the number of charting threads, and the disk file for caching images to disk.
ColdFusion caches charts as they are created. In that way, repeated requests of the same chart load the chart from the cache rather than having ColdFusion render the chart over and over again.

> **Note**
>
> You do not have to perform any special coding to reference a cached chart. Whenever you use the cfchart tag, ColdFusion inspects the cache to see if the chart has already been rendered. If so, ColdFusion loads the chart from the cache.

The following table describes the settings for the ColdFusion charting and graphing engine:

Option	Description
Cache Type	Sets the cache type. Charts can be cached in memory or to disk. Caching in memory is faster, but more memory intensive.
Maximum number of images in cache	Specifies the maximum number of charts to store in the cache. When the limit is reached, the oldest chart in the cache is deleted to make room for a new one. The maximum number of charts you can store in the cache is 250.
Max number of charting threads	Specifies the maximum number of chart requests that can be processed concurrently. The minimum number is 1 and the maximum is 5. Higher numbers are more memory-intensive.
Disk cache location	When caching to disk, specifies the directory in which to store the generated charts.

© 2014 Adobe Systems Incorporated. All rights reserved.

Writing a chart to a variable

In some cases, your application could have charts that are static or charts that, because of the nature of the data input, take a long time to render. In this scenario, you can create a chart and write it to a variable.
Once written to a variable, other ColdFusion pages can access the variable to display the chart, or you can write the variable to disk to save the chart to a file. Saving the variable on disk lets you create or update charts only as needed, rather than every time someone requests a page that contains a chart.
You use the name attribute of the `cfchart` tag to write a chart to a variable. If you specify the name attribute, the chart is not rendered in the browser but is written to the variable.
You can save the chart as an Adobe Flash SWF file, or as a JPEG or PNG image file. If you save the image as a SWF file, you can pass the variable back to a Flash client using ColdFusion Flash Remoting. For more information, see Using the Flash Remoting Service.

> **Note**
>
> If you write the chart to a JPEG or PNG file, mouseover tips and URLs embedded in the chart for data drill-down do not work when you redisplay the image from the file. However, if you save the image as a SWF file, both tips and drill-down URLs work. For more information on data drill-down, see Linking charts to URLs.

Write a chart to a variable and a file

1. Create a ColdFusion page with the following content:

```
<cfchart name="myChart" format="jpg">
 <cfchartseries type="pie">
  <cfchartdata item="New Vehicle Sales" value=500000>
  <cfchartdata item="Used Vehicle Sales" value=250000>
  <cfchartdata item="Leasing" value=300000>
  <cfchartdata item="Service" value=400000>
 </cfchartseries>
</cfchart>

<cffile
 action="WRITE"
 file="c:\inetpub\wwwroot\charts\vehicle.jpg"
 output="#myChart#">
<img src="/charts/vehicle.jpg" height=240 width=320>
```

2. Save the page as chartToFile.cfm in myapps under the web root directory.
3. View the chartToFile.cfm page in your browser.

Reviewing the code

The following table describes the highlighted code and its function:

Code	Description
`<cfchart name="myChart" format="jpg">`	Define a chart written to the `myChart` variable by using the JPEG format.

© 2014 Adobe Systems Incorporated. All rights reserved.

`<cffile action="WRITE" file="c:\inetpub\wwwroot\charts\vehicle.jpg" output="#myChart#">`	Use the cffile tag to write the chart to a file.
`<img src="/charts/vehicle.jpg" height=240 width=320>`	Use the HTML `img` tag to display the chart.

© 2014 Adobe Systems Incorporated. All rights reserved.

Linking charts to URLs

ColdFusion provides a data drill-down capability with charts, which lets you click the data and legend areas of a chart to request a URL. For example, if you have a pie chart and want a user to be able to select a pie wedge for more information, you can build that functionality into your chart.
You use the `url` attribute of the cfchart tag to specify the URL to open when a user clicks anywhere on the chart. For example, the following code defines a chart that opens the page moreinfo.cfm when a user clicks the chart:

```
<cfchart
 xAxisTitle="Department"
 yAxisTitle="Salary Average"
 url="moreinfo.cfm"
 >

  <cfchartseries
   seriesLabel="Department Salaries"
  ...
  />

</cfchart>
```

You can use the following variables in the `url` attribute to pass additional information to the target page:

- `$VALUE$`: The value of the selected item, or an empty string
- `$ITEMLABEL$`: The label of the selected item, or an empty string
- `$SERIESLABEL$`: The label of the selected series, or an empty string
  For example, to let users click the graph to open the page moreinfo.cfm, and pass all three values to the page, you use the following `URL`:

```
url="moreinfo.cfm?Series=$SERIESLABEL$&Item=$ITEMLABEL$&Value=$VALUE$"
```

The variables are not enclosed in number signs like ordinary ColdFusion variables. They are enclosed in dollar signs. If you click a chart that uses this `url` attribute value, it could generate a URL in the following form:

```
http://localhost:8500/tests/charts/moreinfo.cfm?
 Series=Department%20Salaries&Item=Training&Value=86000
```

You can also use JavaScript in the URL to execute client-side scripts. For an example, see *Linking to JavaScript from a pie chart* below.

Dynamically linking from a pie chart

In the following example, when you click a pie wedge, ColdFusion displays a table that contains the detailed salary information for the department represented by the wedge. The example is divided into two parts: creating the detail page and making the pie chart dynamic.

Part 1: Creating the detail page

© 2014 Adobe Systems Incorporated. All rights reserved.

This page displays salary information for the department you selected when you click a wedge of the pie chart. The department name is passed to this page using the `$ITEMLABEL$` variable.

1. Create an application page with the following content:

```
<cfquery name="GetSalaryDetails" datasource="cfdocexamples">
 SELECT Departmt.Dept_Name,
  Employee.FirstName,
  Employee.LastName,
  Employee.StartDate,
  Employee.Salary,
  Employee.Contract
 FROM Departmt, Employee
 WHERE Departmt.Dept_Name = '#URL.Item#'
 AND Departmt.Dept_ID = Employee.Dept_ID
 ORDER BY Employee.LastName, Employee.Firstname
</cfquery>

<html>
<head>
 <title>Employee Salary Details</title>
</head>

<body>

<h1><cfoutput>#GetSalaryDetails.Dept_Name[1]# Department
 Salary Details</cfoutput></h1>
<table border cellspacing=0 cellpadding=5>
<tr>
 <th>Employee Name</th>
 <th>StartDate</th>
 <th>Salary</th>
 <th>Contract?</th>
</tr>
<cfoutput query="GetSalaryDetails">
<tr>
 <td>#FirstName# #LastName#</td>
 <td>#dateFormat(StartDate, "mm/dd/yyyy")#</td>
 <td>#numberFormat(Salary, "$999,999")#</td>
 <td>#Contract#</td>
</tr>
</cfoutput>
</table>
</body>
</html>
```

2. Save the page as Salary_details.cfm in the myapps directory under the web root directory.

Reviewing the code

The following table describes the code and its function:

Code	Description

© 2014 Adobe Systems Incorporated. All rights reserved.

`<cfquery name="GetSalaryDetails" datasource="cfdocexamples">`
` SELECT Departmt.Dept_Name,`
`  Employee.FirstName,`
`  Employee.LastName,`
`  Employee.StartDate,`
`  Employee.Salary,`
`  Employee.Contract`
` FROM Departmt, Employee`
` WHERE Departmt.Dept_Name = '#URL.Item#'`
` AND Departmt.Dept_ID = Employee.Dept_ID`
` ORDER BY Employee.LastName, Employee.Firstname`
`</cfquery>`
	Get the salary data for the department whose name was passed in the URL parameter string. Sort the data by the last and first names of the employee.

`<table border cellspacing=0 cellpadding=5>`
`<tr>`
` <th>Employee Name</th>`
` <th>StartDate</th>`
` <th>Salary</th>`
` <th>Contract?</th>`
`</tr>`
`<cfoutput query="GetSalaryDetails">`
`<tr>`
` <td>#FirstName# #LastName#</td>`
` <td>#dateFormat(StartDate, "mm/dd/yyyy")#</td>`
` <td>#numberFormat(Salary, "$999,999")#</td>`
` <td>#Contract#</td>`
`</tr>`
`</cfoutput>`
`</table>`
	Display the data retrieved by the query as a table. Format the start date into standard month/date/year format, and format the salary with a leading dollar sign, comma separator, and no decimal places.

Part 2: Making the chart dynamic

1. Open chartdata.cfm in your editor.
2. Edit the `cfchart`tag for the pie chart so it appears as follows:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cfchart
  font="Times"
  fontBold="yes"
  backgroundColor="##CCFFFF"
  show3D="yes"
  url="Salary_Details.cfm?Item=$ITEMLABEL$"
  >

 <cfchartseries
  type="pie"
  query="DeptSalaries"
  valueColumn="AvgByDept"
  itemColumn="Dept_Name"
  colorlist="##990066,##660099,##006699,##069666"
 />
</cfchart>
```

3. Save the file as chartdetail.cfm.
4. View the chartdata.cfm page in your browser.
5. Click the slices of the pie chart to request the Salary_details.cfm page and pass in the department name of the wedge you clicked. The salary information for that department appears.

Reviewing the code

The following table describes the highlighted code and its function:

Code	Description
`url = "Salary_Details.cfm?Item=$ITEMLABEL$"`	When the user clicks a wedge of the pie chart, call the Salary_details.cfm page in the current directory, and pass it the parameter named Item that contains the department name of the selected wedge.

Linking to JavaScript from a pie chart

In the following example, when you click a pie wedge, ColdFusion uses JavaScript to display a pop-up window about the wedge.

Create a dynamic chart with JavaScript:

1. Create an application page with the following content:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<script>
function Chart_OnClick(theSeries, theItem, theValue){
alert("Series: " + theSeries + ", Item: " + theItem + ", Value: " + theValue);

 }
</script>

<cfchart
  xAxisTitle="Department"
  yAxisTitle="Salary Average"
  tipstyle=none
  url="javascript:Chart_OnClick('$SERIESLABEL$','$ITEMLABEL$','$VALUE$');"
 >
 <cfchartseries type="bar" seriesLabel="Average Salaries by Department">
  <cfchartData item="Finance" value="75000">
  <cfchartData item="Sales" value="120000">
  <cfchartData item="IT" value="83000">
  <cfchartData item="Facilities" value="45000">
 </cfchartseries>
</cfchart>
```

2. Save the page as chartdata_withJS.cfm in the myapps directory under the web root directory.
3. View the chartdata_withJS.cfm page in your browser:
4. Click the slices of the pie chart to display the pop-up window.

#back to top

© 2014 Adobe Systems Incorporated. All rights reserved.

Charting enhancements

The server-side charting introduced in ColdFusion 10 that allowed you to create highly interactive charts has been further enhanced to produce visually more appealing charts.

Note that the new look and feel will not be available for the following tag attributes:

cfchart

- labelFormat
- seriesPlacement with percent as the value
- sortXAxis
- tipStlye
- xAxisType
- yAxisType
- xOffset
- yOffset

cfchartseries

- markerStyle: letterx, mcross, snow, and rcross.
- paintStyle

ColdFusion 11 provides a utility to convert XML styles (used by the old charting system) to JSON styles (used by the new charting system).

To perform this conversion, you need to use cfchart_xmltojson.bat (or cfchart_xmltojson.sh) available in <cfusion_home>/bin folder:

```
cfchart_xmltojson.bat <xml_file_path>
```

The converted JSON style will be created in the same location as the XML file.

See this document for usage examples.

© 2014 Adobe Systems Incorporated. All rights reserved.

Creating Reports and Documents for Printing

You can use Adobe ColdFusion tags, functions, and tools to create pages and reports that are suitable for printing.

© 2014 Adobe Systems Incorporated. All rights reserved.

About printable output

Although all web browsers let you print HTML pages, HTML-format pages are not optimized for printed output. For example, lack of control over line breaks, page breaks, headers, footers, and page numbers are just a few of the problems that you encounter when designing reports and other pages meant to be printed.
In the context of ColdFusion, the term *printable output* refers to pages that include the following features:

- Page numbers
- Headers and footers
- Page breaks
- Clickable hypertext links when viewed online
  ColdFusion provides the following tags for generating printable output:

- `cfdocument`: Creates printable output and returns it to the browser or saves it in a file. For more information, see Creating PDF and FlashPaper output with the cfdocument tag.
- `cfreport`: Uses the specified report definition to create printable output and return it to the browser or save it in a file. ColdFusion supports report definitions from the following tools:
  - **ColdFusion Report Builder**: The ColdFusion Report Builder is a banded report writer that is integrated with ColdFusion. For more information, see About Report Builder.
  - **Crystal Reports**: Crystal Reports is a report writer whose report definitions you can use with the `cfreport` tag. For more information, see Creating reports with Crystal Reports (Windows only).
    ColdFusion printable reports are available in the following formats:

- **FlashPaper** ColdFusion creates a SWF file. Clients must have an up-to-date version of Adobe Flash Player installed.
- **Adobe Acrobat** ColdFusion creates a PDF file. Clients must have the Adobe Reader installed.
- **Microsoft Excel (ColdFusion reporting only)**ColdFusion creates an Excel spreadsheet.

> **Note**
>
> The Excel report output format type provides limited support for the formatting options available in ColdFusion reporting. Images and charts are not supported and numeric data containing formatting (such as commas, percents, currency) appears as plain text in Excel. The Excel output format supports simple reports only and it is recommended that you give careful design and layout consideration to reports designed for Excel output.

- **Crystal Reports (Windows only)** ColdFusion passes control to Crystal Reports, which creates HTML. This option is available with the `cfreport` tag only.

© 2014 Adobe Systems Incorporated. All rights reserved.

Creating PDF and FlashPaper output with the cfdocument tag

The `cfdocument` tag converts everything between its start and end tags into PDF or FlashPaper output format and returns it to the browser or saves it to a file. As a result you can easily convert HTML to printable output, as the following example shows:

```
<cfdocument format="FlashPaper">
<p>This is a document rendered by the cfdocument tag.</p>
</cfdocument>
```

The `cfdocument` tag supports all HTML and CFML tags, with the following exceptions:

- `cfchart`
- Tags that generate content displayed in Flash Player.
- Interactive tags, such as `form`, `cfform`, and `cfapplet`
- JavaScript that dynamically modifies elements or element positions
  Additionally, the HTML wrapped by the `cfdocument` tag must be well-formed, with end tags for every start tag and proper nesting of block-level elements.

> **Note**
>
> ColdFusion does not return HTML and CFML outside the `<cfdocument>` `</cfdocument>` pair.

Creating basic reports from HTML and CFML

You can convert HTML-based reports into PDF or FlashPaper output by wrapping the HTML in the `cfdocument` start and end tags, and specifying `cfdocument` attributes, as appropriate, to customize the following items:

- Page size
- Page orientation
- Margins
- Encryption (PDF only)
- User password and owner password (PDF only)
- Permissions (PDF only)
  For complete information on these options, see the cfdocument tag discussion in the *CFML Reference*.

> **Note**
>
> Embedding fonts in the report can help ensure consistent display across multiple browsers and platforms. For more information on the considerations related to embedding fonts, see Creating a simple report.

The following example displays a list of employees, using a cfoutput tag to loop through the query:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cfdocument format="flashpaper">
<h1>Employee List</h1>
<!--- Inline query used for example purposes only. --->
<cfquery name="EmpList" datasource="cfdocexamples">
 SELECT FirstName, LastName, Salary, Contract
 FROM Employee
</cfquery>
<cfoutput query="EmpList">
#EmpList.FirstName#, #EmpList.LastName#, #LSCurrencyFormat(EmpList.Salary)#,
 #EmpList.Contract#<br>
</cfoutput>
</cfdocument>
```

Creating sections, headers, and footers

You can use the cfdocument and cfdocumentsection tags to fine-tune your printable output, as follows:

- `cfdocumentitem`: Creates page breaks, headers, or footers.
- `cfdocumentsection`: Divides output into sections, optionally specifying custom margins. Within a section, use the `cfdocumentitem` tag to specify unique headers and footers for each section. Each document section starts on a new page.

The cfdocumentitem tag

You use one or more `cfdocumentitem` tags to specify headers and footers or to create a page break. You can use `cfdocumentitem` tags with or without the `cfdocumentsection` tag, as follows:

- With `cfdocumentsection`: The `cfdocumentitem` attribute applies only to the section, and overrides previously specified headers and footers.
- Without `cfdocumentsection`: The `cfdocumentitem`attribute applies to the entire document, as follows:
  - If the tag is at the top of the document, it applies to the entire document.
  - If the tag is in the middle of the document, it applies to the rest of the document.
  - If the tag is at the end of the document, it has no affect.
    You can use the `cfdocumentitem` tag to create a running header for an entire document, as the following example shows:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cfdocument format="PDF">
<!--- Running header --->
<cfdocumentitem type="header">
 <font size="-3"><i>Directory Report</i></font>
</cfdocumentitem>
<h3>cfdirectory Example</h3>
<!--- Use cfdirectory to display directories by name and size --->
<cfdirectory
 directory="#GetDirectoryFromPath(GetTemplatePath())#"
 name="myDirectory" recurse="yes"
 sort="directory ASC, name ASC, size DESC">
<!---- Output the contents of the cfdirectory as a cftable ----->
<cftable query="myDirectory"
 htmltable colheaders>
 <cfcol header="DIRECTORY:" text="#directory#">
 <cfcol header="NAME:" text="#Name#">
 <cfcol header="SIZE:" text="#Size#">
</cftable>
</cfdocument>
```

The cfdocumentsection tag

When using `cfdocumentsection`, all text in the document must be enclosed within `cfdocumentsection` tags. ColdFusion ignores HTML and CFML outside `cfdocumentsection` tags. The margin attributes override margins specified in previous sections or in the parent `cfdocument` tag. If you specify margin attributes, the unit attribute of the parent `cfdocument` tag control the units; the default for the unit attribute is inches.
Within a section, use the `cfdocumentitem` tag to specify unique headers and footers for each section and a page break before each section, as the following example shows:

© 2014 Adobe Systems Incorporated. All rights reserved.

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

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

Using the cfdocument scope

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

- **currentpagenumber** Displays the current page number.
- **totalpagecount** Displays the total page count.
- **currentsectionpagenumber** Displays the current section number.
- **totalsectionpagecount**Displays the total number of sections.

© 2014 Adobe Systems Incorporated. All rights reserved.

> **Note**
>
> You can use the `cfdocument` scope variables in expressions within the `cfdocumentitem` tag only.

You typically use these variables in a header or footer to display the current page number and total number or pages, as the following example shows:

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

Creating bookmarks in PDF files

You can use the `cfdocument bookmark` attribute to create bookmarks for each section within a PDF document, as the following example shows:

```
<cfdocument format="PDF" bookmark="yes">
 <cfdocumentitem type="header">
 <font size="-1" align="center"><i>Building Better Applications</i></font>
 </cfdocumentitem>
 <cfdocumentitem type="footer">
 <font size="-1"><i>Page <cfoutput>#cfdocument.currentpagenumber# of
  #cfdocument.totalpagecount#</cfoutput></i></font>
 </cfdocumentitem>
 <cfdocumentsection name="Introduction">
  <h3>Introduction</h3>
  <p>The introduction goes here.</p>
 </cfdocumentsection>
 <cfdocumentsection name="Chapter 1">
  <h3>Chapter 1: Getting Started</h3>
  <p>Chapter 1 goes here.</p>
 </cfdocumentsection>
 <cfdocumentsection name="Chapter 2">
  <h3>Chapter 2: Building Applications</h3>
  <p>Chapter 2 goes here.</p>
 </cfdocumentsection>
 <cfdocumentsection name="Conclusion">
  <h3>Conclusion</h3>
  <p>The conclusion goes here.</p>
 </cfdocumentsection>
</cfdocument>
```

The bookmarks appear in the bookmarks panel of the PDF document.

Using cfhttp to display web pages

You can use the cfhttp tag in combination with the cfdocument tag to display entire web pages in PDF or FlashPaper output format, as the following example shows:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<!--- You can pass a URL in the URL string --->
<cfparam name="url.target_url" default="http://www.boston.com">
<cfoutput>
<cfhttp url="#url.target_url#" resolveurl="yes">

<cfdocument format="FlashPaper">
<cfdocumentitem type="header">
 <cfoutput>#url.target_url#</cfoutput>
</cfdocumentitem>
<cfdocumentitem type="footer">
 <cfoutput>#cfdocument.currentpagenumber# / #cfdocument.totalpagecount#</cfoutput>
</cfdocumentitem>
<!--- Display the page --->
#cfhttp.filecontent#
</cfdocument>
</cfoutput>
```

Using advanced PDF options

The `cfdocument` tag supports the Acrobat security options, as the following table shows:

Security option	Description
Encryption	Use the `encryption` attribute to specify whether PDF output is encrypted. Specify one of the following: • 128-bit • 40-bit • none
User password	Use the `userpassword` attribute to specify a password that users must enter to view the document.
Owner password	Use the `ownerpassword` attribute to specify a password that users must enter to view and optionally modify the document.

Additionally, the `cfdocument` tag supports the following Acrobat security permissions through the `permissions` at tribute. Specify one or more of the following values; separate multiple permissions with a comma:

Permission	Description
Printing	Specify the `AllowPrinting` attribute to enable viewers to print the document.
Modification	Specify the `AllowModifyContents` attribute to let viewers modify the document, assuming they have the required software.

© 2014 Adobe Systems Incorporated. All rights reserved.

Copy	Specify the `AllowCopy` attribute to let viewers select and copy text from the document.
Annotation	Specify `AllowModifyAnnotations` to let viewers add comments to the document. If users add annotations, they must save the PDF after making changes.
Screen readers	Specify `AllowScreenReaders` to enable access to the document through a screen reader.
Fill in	Specify `AllowFillIn` to enable users to use form fields.
Assembly	Specify `AllowAssembly` to enable users to create bookmarks and thumbnails, as well as insert, delete, and rotate pages.
Degraded printing	Specify `AllowDegradedPrinting` to enable lower-resolution printing. This format prints each page as a bitmap, so printing can be slower.

Note

The defaults for these options vary, based on encryption level. These options apply to PDF only. For more information, see the cfdocument discussion in the CFML Reference.

The following example creates a PDF document that allows copying only:

```
<cfdocument format="PDF" encryption="40-bit"
 ownerpassword="us3rpa$$w0rd" userpassword="us3rpa$$w0rd"
 permissions="AllowCopy" >
<h1>Employee List</h1>
<cfquery name="EmpList" datasource="cfdocexamples">
 SELECT FirstName, LastName, Salary
 FROM Employee
</cfquery>
<cfoutput query="EmpList">
 #EmpList.FirstName#, #EmpList.LastName#, #LSCurrencyFormat(EmpList.Salary)#<br>
</cfoutput>
</cfdocument>
```

Saving printable reports in files

You can use the `cfdocument filename` attribute to save the generated PDF or SWF content to a file, as the following example shows:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<!--- The compasstravel database is part of the Getting Started
 tutorial application, found under the cfdocs directory. --->
<cfquery datasource="compasstravel" name="compasstrips">
SELECT tripName, tripDescription, tripLocation, price
FROM trips
ORDER BY price
</cfquery>
<cfdocument format="pdf"
  filename="#GetDirectoryFromPath(GetTemplatePath())#/compasstrips.pdf"
  overwrite="yes">
 <cfdocumentsection>
  <h1 align="center">Compass Travel</h1>
  <h2 align="center">Destination Guide</h2>
  <p align="center"><img src="cfdocs/getting_started/photos/somewhere.jpg"></p>
 </cfdocumentsection>
 <cfdocumentsection>
  <cfdocumentitem type="header">
   <font size="-3"> <i>Compass Travel Trip Descriptions</i></font>
  </cfdocumentitem>
  <cfdocumentitem type="footer">
   <font size="-3">
    <cfoutput>Page #cfdocument.currentpagenumber#</cfoutput>
   </font>
  </cfdocumentitem>
  <cfoutput query="compasstrips">
   <hr>
   <h2>#tripName#</h2>
   <p><b>#tripLocation#</b></p>
   <p>Price: #DollarFormat(price)#</p>
   <p>#tripDescription#</p>
  </cfoutput>
 </cfdocumentsection>
</cfdocument>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

Creating reports with Crystal Reports (Windows only)

When running on Windows, the cfreport tag also supports the execution of reports created using Crystal Reports version 9 or 10.

> **Note**
>
> When you install Crystal Reports, select the Enable export to HTML and Enable export to Disk options. These options are not enabled by default, so you must use the Custom Install option.

1. Create a report definition in Crystal Reports.
2. Create a CFM page and add a `cfreport` tag that invokes the Crystal Reports report definition. The following example shows the `cfreport`tag invoking a Crystal Reports report definition and passing a filter condition:

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

3. Open a browser and display the CFM page.

> ✅ ColdFusion uses COM to call Craxdrt9.dll for Crystal Reports version 9, and Craxdrt.dll for Crystal Reports version 10. If you have problems with the `cfreport_` tag, ensure that these DLLs are registered and, if not, use regsvr32 to register them (the default location for these DLLs is C:\Program Files\Crystal Decisions\Report Designer Component)._

For complete information on defining reports in Crystal Reports, see the Crystal Reports documentation.

#back to top

© 2014 Adobe Systems Incorporated. All rights reserved.

Creating Reports with Report Builder

Improve your access to important business data by creating integrated business reports with Adobe ColdFusion Report Builder and CFML.

© 2014 Adobe Systems Incorporated. All rights reserved.

About Report Builder

ColdFusion reporting adds integrated business reporting to ColdFusion, providing access to important business data. ColdFusion reporting consists of server-side run-time processing and a graphical user interface (GUI), called the Report Builder.
For information on installing the Report Builder, see Getting started.
The Report Builder is a Windows-only tool that lets you build banded reports. A *banded report* consists of multiple horizontal sections (bands), one band for each part of a printed report. For example, data and text in the report header band prints at the beginning of the report, data and text in the page header band prints at the beginning of each page, and data and text in the page footer band prints at the end of each page. In the middle of the report is the detail band, which, at run time, contains one row for each row in the report's result set or database query.

> The Report Builder contains an extensive online Help system, including quick-start tutorial topics and context-sensitive dialog box Help. Press F1 to consult the online Help.

Report Builder and CFR files

The Report Builder is a stand-alone application that creates report definitions, interacting with a ColdFusion server, as necessary. The Report Builder stores report definition information in a ColdFusion Report (CFR) file. This file contains field definitions, formatting, database SQL statements, CFML, and other information. You display a CFR file by using the `cfreport` tag and, if enabled for the report, display the report by running the CFR file in a browser.

> **Note**
>
> The Report Builder runs in the Windows platform only. However, the CFR files created by the Report Builder run on all platforms that ColdFusion runs on and that have ColdFusion Reporting enabled.

RDS

Remote Development Services (RDS) is a proprietary protocol that uses HTTP to enable the Query Builder and Chart Wizard to access database data through a ColdFusion data source. To enable this functionality in the Report Builder, you define settings for an *RDS server*. RDS server is another name for an associated ColdFusion server that has enabled RDS.
For more information, see *Using CFML in reports* in Common reporting tasks and techniques.

Run time

At run time, you display the CFR file by using a ColdFusion server that has ColdFusion Reporting enabled. You can display the CFR file directly or run it through the `cfreport` tag. Also, you can save the report to a file instead of returning output to the browser. If the report requires input parameters or a passed query, use the `cfreport` tag. If you pass a query attribute in the `cfreport` tag, it overrides any internal SQL statement in the report definition.

© 2014 Adobe Systems Incorporated. All rights reserved.

Getting started

For installation instructions, see *Installing ColdFusion*. When you install the Report Builder, it also registers Windows DLLs that RDS uses. If these DLLs fail to register properly, the Report Builder generates errors at startup and when using RDS.

Setup Wizard

The first time you start the Report Builder, it runs the Setup Wizard. The Setup Wizard prompts you to define default settings for an associated ColdFusion server. These settings include the following:

- Default unit of measurement: Inches, centimeters, or pixels.
- ColdFusion server. The RDS server that the Query Builder and Chart Wizard use to access database data (RDS must be enabled on this server). The Setup Wizard requires the following information:
  - Host name or IP address.
  - Web server port. Typically, the port is 80 if you are using a web server connector, 8500 if you are using the built-in web server in the server configuration, 8300 if you are using the built-in web server with an instance created with the ColdFusion Enterprise Instance Manager, or a J2EE-server-specific web server port number.
  - RDS password for the associated ColdFusion server.
- Directory path to the web root used by the associated ColdFusion server (for example, C:\Inetpub\wwwroot or C:_ColdFusion_\wwwroot).
- URL for the web root used by the associated ColdFusion server (for example, http://localhost or http://localhost:8500).
  After running the Setup Wizard, the Report Gallery dialog box appears. When you click the Using A Report Wizard radio button, the Report Builder runs the Report Creation Wizard, which prompts you for information and automatically generates a complete report definition.
  For more information on the Report Creation Wizard, see the Report Builder online Help.

Configuring RDS

Configure one RDS server for each ColdFusion server for which you define reports. After you configure an RDS server, you can use the Query Builder to access data sources that you defined in the associated ColdFusion server, and select database columns for use as query fields in a report.

Add an RDS server

1. Open the Preferences dialog box by selecting Edit > Preferences from the menu bar.
2. Click Server Connection.
3. Click the plus sign (+) next to the pop-up menu in the upper-left corner of the dialog box.
4. In the Configure RDS Server dialog box, specify the following information, and then click OK:

- **Description** A name for the server connection. This name appears in the pop-up menu on the left side of the Query Builder.
- **Host name** The host on which ColdFusion runs. Type **localhost** or an IP address.
- **Port** Web server port number. Accept the default port (80) or enter the port number of the ColdFusion server's built-in web server (8500 is the default port number).
- **Context Root (J2EE configuration only)** The context root (if any) for the ColdFusion web application.
- **Use Secure Sockets Layer** (Optional) Enables SSL security.
- **User Name** Not applicable to ColdFusion RDS.
- **Password**RDS password. You set this password in the ColdFusion Administrator.

> ✅ Do not confuse the RDS password with the ColdFusion Administrator password, which you also manage through the ColdFusion Administrator.

© 2014 Adobe Systems Incorporated. All rights reserved.

- **Prompt for password** Specifies whether to prompt for an RDS password each time you use the Query Builder. If you select this option, leave the User Name and Password fields blank.

Designate a default RDS server

1. Open the Preferences dialog box by selecting Edit > Preferences from the menu bar.
2. Click Server Connection.
3. Select an RDS server from the Preferred RDS Server pop-up menu, and click OK.
   The Report Builder automatically connects to the specified server when you display the Query Builder or Chart Wizard.

User interface usage, tips, and techniques

The Report Builder workspace includes the following areas:

- **Toolbox**: Contains nonvariable elements placed in a report, including text, shapes, images, subreports, and graphs. To use toolbox elements, click the element, and then drag in the report band to define the element's size. After you place an element on a report band, you can modify its appearance and behavior by using the Properties panel.
- **Alignment panel**: Use Control-click or Shift-click to select multiple elements in a report band, and then click the appropriate alignment icon. You can also use Control+A to select all elements in a report band.
- **Report bands**: Place toolbox elements, query fields, and calculated fields on report bands. The default report bands are report header, page header, column header, page footer, report footer, and watermark. Page header, page footer, and watermark are closed by default; to open them drag one of the adjacent splitter bars. To define additional bands for groups, select Report > Group Management.
  ColdFusion provides three panels that you use to place and format data elements in the workspace:

- **Properties panel**: Contains display and report characteristics for the selected field. To display the Properties panel, choose Window > Properties Inspector from the main menu. To change a property value, type or select a new value, and press Enter. For complete information on properties, see the Report Builder online Help.
- **Fields and parameters panel**: Contains items for query fields, input parameters, and calculated fields. To display the Fields and parameters panel, choose Window > Fields and Parameters from the main menu. Use the add, edit, and delete icons to manage these fields. After you define a field, drag the field name to add the field, its associated label, or both, to a report band.
- **Report styles panel**: Contains the styles that you define for a report. To display the Report styles panel, choose Window > Report Styles from the main menu. Use the add, edit, and delete icons to manage report styles. After you define styles, you apply them to elements on the report instead of specifying font, font size, and so on, for each individual element. If your report layout, platform, or font availability requirements change, you can modify the style to apply the changes throughout the report. Additionally, you can specify a style as the default for the report: if no other style is applied to an element in the report, Report Builder applies the default style to that element.
  The View menu lets you control whether toolboxes and panel windows appear. Also, you can click a window's title to undock it and drag it to another area of the screen. For example, you can drag all three panels and dock them in the same window. Report Builder lets you switch between them by clicking the tabs at the top of the window. To redock a tool window or panel, drag it to the side or corner until a rectangle appears, and then release the mouse button.
  For more information, see Common reporting tasks and techniques and the online Help.

Report definition guidelines

To ensure a successful report, plan the following before defining it in the ColdFusion Report Builder:

- Report design issues:
- **Audience** Why are you creating this report? Who is the audience?
- **Data** What data must be in the report? Where does it come from? Whether you use the Query Builder or pass

© 2014 Adobe Systems Incorporated. All rights reserved.

a query to the report, plan the data in advance.

- **Grouping** Are groups required? If so, ensure that the result set is returned in the correct order, and you define a group based on the sort column.
- **Calculated fields** Are there fields that must be totaled or calculated? For column totals, use calculated fields. For calculated totals on individual rows, use SQL. For more information, see Common reporting tasks and techniques.
- **Input parameters** Does the report require variable input? If so, define an input parameter and pass values to the report at run time by using the `cfreportparam` tag. For more information, see Common reporting tasks and techniques.
- Data retrieval strategy:
- *Query Builder
  and basic SQL* Use this option when your report has standard selection criteria (such as a WHERE clause with sorting and a fixed set of selection criteria) and when you have to develop a report quickly. This method also lets you specify `cfquery` options, such as caching.
- **Query Builder and advanced query mode** Use this option when you use a ColdFusion query encapsulated in the report definition. This option is also useful if the query comes from the `cfdirectory`, `cfldap`, or `cfp op` tags; query of queries; or is dynamically constructed with the `QueryNew` function.
- **The cfreport tag and a passed query** Use this option when you require more control over the result set used in the report. For example, use this option if your application has a form that your clients use to construct dynamic selection criteria.
- Related visual information:
- **Charts** For more information, see *Using charts*.
- **Subreports** For more information, see *Using subreports*.

Managing fonts with printable reports

Ideally, reports achieve a consistent look across all client platforms and all browsers. ColdFusion handles this consistency automatically for graphics and images, using the size specifications in the report definition. However, potential differences in font availability across browsers, browser versions, languages, and platforms can affect the font display for your report. A variety of factors ensure a consistent report display.

Embedded fonts

You can ensure consistent report display by embedding fonts. However, reports with embedded fonts have a larger file size.

Output format

The FlashPaper and PDF output formats handle embedded fonts differently.

- **FlashPaper** FlashPaper always embeds fonts, which ensures that reports always display appropriately.
- **PDF** PDF reports can optionally embed fonts, however, if your report doesn't use embedded fonts, ensure that the fonts are available on the client computers.

Font availability on the server computer and the client computer

ColdFusion has different requirements for rendering the fonts in a report, depending on where the fonts are located.

- **Server computer** For all formats, the fonts used in a report must reside on the computer that runs ColdFusion. ColdFusion requires these fonts to render the report accurately. ColdFusion automatically locates Acrobat built-in fonts and fonts stored in typical font locations (such as the Windows\fonts directory). However, if your server has additional fonts installed in nonstandard locations, register them with the ColdFusion Administrator so that the `cfdocument` and `cfreport` tags can locate and render PDF and FlashPaper reports.
- **Client computer** If your PDF report does not embed fonts, the fonts reside on the client computer to ensure

© 2014 Adobe Systems Incorporated. All rights reserved.

consistent report display.

Mapping logical fonts to physical fonts

If you are using Java logical fonts, such as serif, sans serif, or monospaced, ColdFusion maps these fonts to physical fonts by using specifications in the *cf_root*/cfusion/lib/cffont.properties file. (In the JEE configuration, this file is in the *cf_webapp_root*/WEB-INF/cfusion/lib directory). You can modify these mappings, if necessary. Also, if you are using an operating system whose locale is not English, you can create a locale-specific mapping file by appending *.java-locale-code* to the filename. If ColdFusion detects that it is running on a non-English locale, it first checks for a cffont.properties.*java-locale-code* file. For example, on a computer that uses the Chinese locale, name the file cffont.properties.cn. For more information on Java locale codes, see the Sun website.

> ✅ The ColdFusion install includes a cffont.properties.ja file for the Japanese locale.

This discussion applies to both the `cfdocument` and `cfreport` tags. For more information, see the Report Builder online Help.

© 2014 Adobe Systems Incorporated. All rights reserved.

Common reporting tasks and techniques

With Report Builder, you can include data in reports in a variety of formats, and perform calculations on the information. For more information, including troubleshooting tips, see Report Builder online Help.

Grouping and group breaks

You can add clarity to a report's organization by grouping the information. You can define separate headings for each new group and also display group-specific summary information, such as subtotals at the end of each group's area of the report. For example, you could create a report that displays departments, employees, and their salaries. Grouping the data by department lets users quickly understand department salary characteristics. When the department ID changes, the ColdFusion Report Builder triggers a group break. The group break completes the old group by displaying the group footer and starts the new group by displaying the group header.
The ColdFusion Report Builder does not group data itself. Ensure that the SQL used to retrieve the result set is already grouped in the appropriate order; typically you implement grouping by specifying an ORDER BY clause in the SQL SELECT statement used for the report. For example, you could use the following SQL SELECT statement:

```
SELECT EmployeeID, LastName, FirstName, Title, City, Region, Country
FROM Employees
ORDER BY Country, City
```

For this example, you can define two groups: one that corresponds to Country, and a second group that corresponds to City. When you define more than one group, the Group Management dialog box appears with Up Arrow and Down Arrow keys, which you can use to control group hierarchy. For example, country should precede city, because countries contain cities.

Define a group

1. Select Report > Group Management from the menu bar.
2. Click Add.
3. Specify a group name in the Name field.
4. Specify the value that controls grouping (also called a group expression) in the Group on field. At run time, ColdFusion triggers a group break when the result of this value changes. These values are often query field names. However, this value can also be a calculated field or other type of expression. Sample group expressions include the following:

- **Query field** Creates a group break when the associated column in the result set contains a different value. The field that you specify must be one of the sort criteria for the result set; for example, query.country.
- **Calculated field** Creates a group break when a calculated field returns a different value. For example, if the expression calc.FirstLetter returns the first letter of a query column, you can group a report in alphabetical order.
- **Boolean expression** Creates a group break when a Boolean expression returns a different value. For example, if your result set is sorted by the passpercentage column, you could use the Boolean expression query.passpercentage LT 50.

1. Specify group break options:

- *Start New
  Column* Forces a new column on a group break.
- **Start New Page** Forces a new page on a group break.
- **Reset Page Number** Resets the page number to 1 on a group break.

1. Specify band size and printing information:

- **Min. height for group** The minimum height that must remain on a page for ColdFusion to print the group

© 2014 Adobe Systems Incorporated. All rights reserved.

band on that page.
- **Reprint Header on Each Page** Displays the group header on each page.

1. Click OK. The Report Builder adds the group to the report and creates header and footer bands for the group.
2. Click OK again.
3. Add headings, text, query fields, calculated fields, and other information to the group's header and footer.

Create group subtotals

1. Create a calculated field to contain the group subtotal. Create the calculated field that uses the following criteria:
   - Specify a numeric data type.
   - Select Sum in the Calculation field.
   - Specify the field to sum on in the Perform Calculation On field. For example, a report on employees by department could sum on query.emp_salary.
   - Specify to reset the field when the group changes.
2. Place the calculated field on the report.
   For more information on calculated fields, see the Report Builder online Help.

Defining, modifying, and using fields and input parameters

The Report Builder supports variable data through query fields, input parameters, and calculated fields, as follows:

- **Query field** Maps to columns in the database result set associated with the report. You define one query field for each column in the associated database query.
- **Calculated field** Analyzes or sums multiple detail rows in a report. ColdFusion dynamically generates calculated field values at report-generation time, optionally recalculating the value with each new report, page, column, or group.
- **Input parameter** Specifies data fields that you pass to the report at run time through the `cfreportparam` tag or from a main report to a subreport. You can place input parameters directly on a report band or you can use them as input to a calculated field.

Define a query field

1. Choose Window > Fields and Parameters.
2. Click Query Fields.
3. Click the plus sign (+) at the upper edge of the tab.
4. Type a value for the name field. This value must match a column name in the corresponding cfquery statement and cannot contain a period.
5. Type a default label.
6. Specify the data type of the corresponding database column, as follows:

Object	Time	Long
Boolean	Double	Short
Byte	Float	Big Decimal
Date	Integer	String
Time Stamp	BLOB	CLOB

7. Click OK.

© 2014 Adobe Systems Incorporated. All rights reserved.

> **Note**
>
> The Query Builder defines query fields automatically for all database columns in the result set (this action does not apply to the Advanced Query Builder). Also, if you run the Query Builder as part of the Report Creation Wizard, the wizard places query fields on the report.

Define a calculated field

1. Choose Window > Fields and Parameters.
2. Click Calculated Fields.
3. Click the plus sign (+) at the upper edge of the tab.
4. Specify a name, default label text, and data type. Data type options are the same as for query fields.
5. Specify calculation options:

- **Calculation** Specifies the type of calculation that ColdFusion performs. Valid values are: Average, Count, DistinctCount, First, Highest, Lowest, Nothing, Standard Deviation, Sum, System, and Variance. If you specify Nothing, you typically use the Perform Calculation On field to specify a dynamic expression. Except for Nothing (for which you use the Perform Calculation On field) and System (for which you write a customized scriptlet class), you use these calculations for group, page, and report totals.
- **Perform Calculation On** Specifies a field or expression. Click the ... button to display the Expression Builder.
- **Initial Value** Specifies an initial value for the calculated field.

1. Specify the following reset options, and click OK:

- **Reset Field When** Specifies when to reset the calculated field value. Valid values are: None, Report, Page, Column Group.
- **Reset Group** If Reset Field When is set to Group, use this field to specify the group whose group break triggers the reset. For additional information on calculated fields, see the Report Builder online Help.

Define an input parameter

1. Choose Window > Fields and Parameters.
2. In the Fields and Parameters panel, click Input Parameters.
3. Click the plus sign at the upper edge of the tab.
4. In the Add Input Parameter dialog box, enter a value for the name field. This value must match an input parameter, such as the name attribute of a `cfreportparam` tag included in the `cfreport` tag that uses the report definition.
5. Enter the default label text.
6. Specify a data type and default value, and click OK. Data type options are the same as for query fields.
   For more information on using input parameters, see *Using input parameters to pass variables and other data at run time* in this page and *Using subreports* in this page.

Place a query field, calculated field, or input parameter on a report band

1. In the Fields and Parameters panel, use the radio buttons to specify whether to place the label, the field, or both.
2. Drag the query field, calculated field, or input parameter from the Fields and Parameters tab to the appropriate report band.
3. Drag the query field, calculated field, or input parameter to the desired band.
4. (Optional) Use the Properties panel to customize the field display.
   For example, you could have a query field named query.emp_salary and a calculated field that sums query.emp_salary, resetting it with each group. Place query.emp_salary in the detail band, and the associated calculated field in the group footer band.

Using toolbox elements on report bands

© 2014 Adobe Systems Incorporated. All rights reserved.

You use the toolbox to add graphic and textual elements, such as images, circles, squares, lines, dynamic fields, charts, and subreports, to report bands.
The basic technique for adding toolbox elements is to click in the toolbox element and then drag to define an area in the appropriate report band. For some toolbox elements, such as image and text box, a dialog box immediately appears, prompting for more information. For all toolbox elements, you customize the appearance of the element by using the Properties sheet.

> You can add toolbox elements from the Insert menu.

For information on charts, see Using charts. For information on subreports, see Using subreports.

Create a text box

1. Click the Label icon (abc) in the toolbox.
2. Define the area for the label by dragging on the desired band.
3. Enter the label text in the Edit Label Text dialog box. To add a line break, press Control+Enter.
4. Click OK, or press Enter.

> **Note**
>
> ColdFusion trims leading and trailing blanks from labels. To include leading and trailing blanks, define a dynamic field and include the blanks in the expression, for example, " My Title ".

Import image files

1. Click the Image icon in the toolbox.
2. Define the area for the image by dragging on the desired band.
3. In the Image File Name dialog box, navigate to the file that contains the image, select the file, and click OK.

Use a database BLOB column as an image source

1. Click the image icon in the toolbox (the icon has a tree on it).
2. Define the area for the image by dragging on the desired band.The Image File Name dialog box appears.

> You can also drag the BLOB field from the Fields and Parameters tab to a report band.

1. Click Cancel.The Expression Builder appears.
2. Click the Image Type pop-up menu and change File/URL to BLOB.
3. Select the query field or input parameter that contains the BLOB column.

> **Note**
>
> The BLOB column must contain a binary image in GIF, JPEG, or PNG format.

1. Click OK.

> **Note**
>
> These instructions assume that the contents of the BLOB column can be rendered as an image.

Add rectangles, ellipses, and lines

1. Click the rectangle, ellipses, or line icon in the toolbox.
2. Define the area or line by dragging on the desired band.
3. Resize the selected element by dragging the handles that surround it.

© 2014 Adobe Systems Incorporated. All rights reserved.

> ✅ Pressing the Control key while resizing a rectangle, ellipsis, or line, constrains the element to a square, circle, or angles that are multiples of 45 degrees.

Add dynamic fields

1. Click the Field icon in the toolbox.
2. Define the area for the dynamic field by dragging on the desired band.The Add Field dialog box appears (if you haven't defined any query fields, the Expression Builder appears).
3. Select the field to add. Selecting a query field, calculated field, or input parameter is the same as dragging from the Fields and Parameters tab.
4. (Optional) Select Manually Entered Expression.The Expression Builder appears. This option is useful for calculations that use variables in the same row. For example, to compute total price for an order detail line item, you could use the following expression:

```
LSNumberFormat((query.unitprice * query.quantity), ",_.__")
```

5. Click OK.

Aligning elements

Organized element layout is essential to a visually pleasing report. You achieve this organization by aligning, spacing, and centering visual elements on each band relative to each other, to the band itself, and to elements on other bands.
The Report Builder Align Palette includes the following options:

- Align left, center, and right
- Align top, horizontal, and bottom
- Same heights, widths, and both
- Space equally horizontally
- Space equally vertically
  You align, size, and space multiple report elements, as follows:
  Relative to the band they are in: You control relative alignment through the Align to Band icon, which is the bottom icon in the Align Palette. When it is enabled, the Align to Band icon has a rectangle surrounding it, and the Report Builder aligns and spaces one or more elements relative to the height and width of the band.
  Relative to each other: When Align to Band is disabled, Report Builder aligns and spaces two or more elements relative to each other.

Use the Align Palette

1. Select two or more elements by pressing Control-click, Shift-click, or using lasso select.
2. Click the alignment icon, or select Modify > Alignment > *alignment option* from the menu bar.

> ✅ The Align Palette options are also available from Modify > Alignment on the menu bar.

For complete information on fine-tuning element display, see the Report Builder online Help.

Using report styles

A report style is like a font style in Microsoft Word. Instead of explicitly associating an element with formatting specifications, you associate the element with a style. doing so provides you with report-wide control of the formatting characteristics of your report.
Additionally, you can specify style that is the default for the report. The ColdFusion Report Builder uses the default

© 2014 Adobe Systems Incorporated. All rights reserved.

style for all fields for which you have applied no other font specifications or styles. The default style, if defined, is displayed in bold in the Report Styles panel.
Report Builder also lets you import styles from a Cascading Style Sheet (CSS) file and export styles defined in Report Builder to a CSS file. This way you can enforce standard formatting across reports and override styles at run time from a CFM page. For more information, see Using Cascading Style Sheets and the *CFML Reference*.

> **Note**
>
> When choosing fonts for your report, ensure that the fonts are available on the server that runs ColdFusion and (if you don't embed fonts) on the client computer. For more information on fonts, see Creating a simple report.

Define a style

1. Choose Window > Report Styles.
2. Click the icon at the upper edge of the Report Styles tab.
3. Type a value for the Name field. Style names must be unique.
4. Add other style characteristics, and click OK.

Specify a style as the default

1. Edit a text style or create one.
2. Select the option with this label: This option is the default style if no other style is selected for an object.
3. Add or modify other text style characteristics, and click OK.

Apply a style to a report element

1. Select the element in the report band.
2. Choose Window > Properties Inspector.
3. Choose the style from the Style pop-up menu.
   For more information, see the Report Builder online Help.

Previewing reports

Report building is an iterative process and most developers periodically display the in-progress report to review their most recent changes. If your report uses an internal query and you established default web root settings, preview functionality is enabled automatically. If your report uses a passed query, define an associated CFM page and associate that page with the report. The Report Builder runs this page when you request Report Preview.

Preview a report that uses an internal query

1. (Optional) Define default server connection information using the Preferences dialog box, if you did not define these settings previously:
   - Default RDS server configuration (used for Query Builder and Chart Wizard only; not required for report preview).
   - Fully qualified path for the local web root directory; for example, C:_ColdFusion_\wwwroot or C:\Inetpub\wwwroot.
   - URL for the local web root, for example, http://localhost:8500 or http://localhost.
2. (Optional) Specify the output format in the Report Properties dialog box (the default format is FlashPaper).
3. (Optional) If a CFM page runs, specify the URL of the CFM page in the Report Properties dialog box.
4. Save your report.
5. Select File > Preview from the menu bar to display the report.

© 2014 Adobe Systems Incorporated. All rights reserved.

> **Note**
>
> If the Report Builder displays the Edit Preview Report URL dialog box instead of displaying the Preview window, select Edit > Preferences from the menu bar and insure that the web root file and URL settings are correct on the Server Connection pane.

1. Close the preview window by pressing F12.
   If your report is designed to accept a query object from a `cfreport` tag, associate a URL with the report. If necessary, the Report Builder prompts for this URL when you preview the report. Otherwise, you can open the Report Properties dialog box, and specify the URL of the CFM page in the Report Preview URL field.

> You can use the `cfreport_` tag to run a report, regardless of whether the report has an internal query or is passed a query._

Preview with an associated CFM file

1. Select Report > Report Properties from the menu bar.
2. Specify the URL of the associated CFM page in the Report Preview URL field. This CFM page must contain a `cfreport` tag whose `template` attribute specifies the current CFR file and, if necessary, passes a query in the `query` attribute.
3. Save your report.
4. Press F12. Depending on the output format that you have chosen, the Preview Report window displays your report in PDF, FlashPaper, RTF, XML, HTML, or Excel format.

Displaying page numbers

The Report Builder includes a built-in calculated field named `PAGE_NUMBER`, which displays the current page number when you place it on a report band.

Add a built-in calculated field

1. Click the Field tool in the toolbox.
2. Drag in the center of the header or footer band to define the size of the page number field.The Add Field dialog box appears, listing all fields defined for the report, including built-in calculated fields and input parameters.
3. Select `calc.PAGE_NUMBER`, and click OK.

> You can use the Field tool to add any type of field (query field, calculated field, input parameter) to a report.

For information on the other built-in calculated fields, see the Report Builder online Help.

Using layered controls

Layered controls are elements that you place at the same location of a report band, and then use `PrintWhen` expre ssions to conditionally display one or the other at run time. You can use layered elements to customize the circumstances under which the elements display and enhance a report's ability to communicate important information.

Place an element directly over another element

1. Place the elements on the band.
2. Choose Window > Properties to display the Properties panel.
3. Specify a `PrintWhen` expression, display properties, and placement properties for each element using the

© 2014 Adobe Systems Incorporated. All rights reserved.

Properties panel, as follows:

4. Specify a `PrintWhen` expression for each element. For example, you could specify the following expression to display one element when shippeddate is later than requireddate (that is, late) and another element when shippeddate is earlier than requireddate:

- **First element** query.shippeddate LTE query.requireddate
- **Second element** query.shippeddate GT query.requireddate

1. Specify different display characteristics for each element. For example, if an order is late, display it in red text.
2. Set the Top, Left, Height, and Width properties to the same values for each element.

> ✅ When you specify identical placement properties, you access the individual elements through the Layered Controls menu.

Use the Layered Controls menu

1. Right-click on the top element.
2. Select Layered Controls > *elementname* from the pop-up menu. The Report Builder identifies each layered element by displaying its `PrintWhen` expression.
3. Select the element and choose Window > Properties Inspector to view the element properties.

Using links

You can include hypertext links from query fields, calculated fields, input parameters, charts, and images to a variety of destinations:

- An anchor or page within the same report
- An anchor or page within another report
- An HTML page, optionally specifying an anchor and URL parameters
  One use for links is to create drill-down reports, in which you click an item to display detailed information. For example, clicking an employee line item passes the employee ID as a parameter to a page that displays complete information for the employee.
  For complete usage information on creating anchors and hypertext links, see the Report Builder online Help.

Defining properties for report elements

A set of properties defines every element on a report, including the report itself. These properties affect the look, feel, and behavior of each element.
For many properties, the Report Builder lets you define their values through user interface elements, such as dialog boxes, toolbar icons, and menu items. For example, you set a text label's font size using a toolbar icon. You can set values for all properties, however, through the Properties panel, which display all properties for the currently selected element.

> ✅ Sometimes a report contains multiple, closely spaced elements and it is difficult to select an individual element using the mouse. In this case, selecting the element from the Properties panel pop-up menu is an easy way to select an element.

The Properties panel has two views:

- **Sort alphabetically** All properties for the currently selected element display in alphabetical order.
- **Sort into groups**The Properties panel displays related properties in the following predefined groups:
  - Advanced
  - Columns
  - Page Layout
  - Printing

© 2014 Adobe Systems Incorporated. All rights reserved.

- Colors and Style
- Data
- Font
- Font Style
- Formatting
- Hyperlinks
- Layout
- Print Control

  The Report Builder displays only groups that relate to the currently selected element.

Set or modify a property for an element in the workspace

1. Select the element.
2. (Optional) If the Properties panel is not already displayed, choose Window > Properties Inspector.The Report Builder displays its properties in the Properties panel.
3. Modify the property. Depending on the property, you enter a value, select a value from a pop-up menu, or open the Expression Builder to use an expression.
4. Press Enter.

> ✅ When you select a color, double-click the color.

Choose a different element

Select the element from the pop-up menu. When you select a new element, the Report Builder selects the element and displays its properties.

> ✅ Although the Properties panel is a powerful way to set properties, you typically set properties through dialog boxes and toolbar icons. For example, you use the Report Properties dialog box to set report-wide settings. For complete information on setting properties, see "Property reference" in the Report Builder online Help.

Displaying reports

Your application can run a report by displaying the CFR file in a browser or by displaying a CFM page whose `cfreport` tag runs the report.
You can optionally use the `cfreport` tag to save the report to a file.
The `cfreport` tag supports advanced PDF encryption options. For more information, see cfreport in the *CFML Reference*.
For information on report preview, see *Previewing reports*.

Display a report by using the cfreport tag

1. Create a report, with or without an internal query.
2. Create a CFM page and add a `cfreport` tag that runs the report. If the report does not use an internal query, also populate a query and pass it using the `query` attribute. If the report uses an internal query and you use the `query`attribute, the passed query overrides the internal query.

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cfquery name="northwindemployees" datasource="localnorthwind">
 SELECT EmployeeID, LastName, FirstName, Title, City, Region, Country
 FROM Employees
 ORDER BY Country, City
</cfquery>

<CFREPORT format="PDF" template="EmpReport.cfr"
 query="#northwindemployees#"/>
```

Note

ColdFusion does not render text that occurs before or after the `cfreport_` tag._

1. Open a browser and display the CFM page.ColdFusion generates the report.

If you display an HTML report, ColdFusion generates temporary files for images in the report. You can specify how long the temporary files are saved on the server by using the `resourceTimespan` attribute of the cfreport tag. For more information, see the *CFML Reference.*

Display a CFR file in a browser

1. Create a report that uses an internal query and does not use input parameters.
2. Open a browser and display the CFR file.

Save a report to a file

1. Create a report, with or without an internal query.
2. Create a CFM page and add a `cfreport` tag that runs the report. Optionally pass a `query` attribute, as described in the previous procedure. Include a `filename`attribute that specifies the fully qualified name of the file being created, as the following example shows:

```
<CFREPORT format="PDF" template="emppicture.cfr"
 filename="#GetDirectoryFromPath(GetTemplatePath())#/emppicture.pdf"
 overwrite="yes"/>
```

If you write the report output to an HTML file, ColdFusion creates a directory located relative to the HTML file, generates files for the images (including charts) in the report, and stores the image files in the directory. For more information, see *Exporting the report in HTML*.

Use the .pdf extension for PDF output format, the .swf extension for FlashPaper output format, .xml extension for an XML file, .rtf extension for an RTF file, .html extension for HTML files, and the .xls extension for Excel format.

1. Open a browser and display the CFM page. ColdFusion generates the report, saves the file, and displays an empty page in the browser.

Disable browser display of the CFR file

1. Open the Report Properties dialog box by selecting Report > Report Properties from the menu bar.

© 2014 Adobe Systems Incorporated. All rights reserved.

2. Clear the Allow Direct .CFR Browser Invocation option, and click OK.

Using input parameters to pass variables and other data at run time

Input parameters are data fields that you pass to the report at run time. You can place input parameters directly on a report band or you can use them as input to a calculated field.
Define input parameters in the same manner as query fields. You can specify a default value that ColdFusion uses when no corresponding parameter exists. For more information on defining input parameters, see *Defining, modifying, and using fields and input parameters*.
You use input parameters in the following ways:

- **Through the cfreportparam tag**: Input parameters must correspond, by name, to `cfreportparam` tags embedded in the CFM page invocation. For example, if you define an input parameter named ReportTime, you pass a `cfreportparam`tag with a name attribute set to ReportTime, as the following example shows:

```
<cfreport format="PDF" template="FourthReport.cfr" query="#coursedept#">
 <cfreportparam name="ReportTime" value="#DateFormat(Now())#,
#TimeFormat(Now())#">
</cfreport>
```

- **Subreport parameters**: When a subreport requires information from a main report, you define subreport parameters in the main report and corresponding input parameters in the subreport. For more information, see Using subreports.
  For information on dynamically populating input parameters at run time, see Advanced query mode.

Using CFML in reports

CFML is the scripting language for the Report Builder. By leveraging CFML, you can create reports that select and format data to meet your needs. You use CFML in the following areas of the Report Builder:

- Advanced query mode
- Report functions
- Expressions

Advanced query mode

In some cases, you create a complex query, reuse an existing query, or encapsulate additional CFML processing as part of query creation for the report. To use a query in these ways, you use advanced query mode to create CFML that returns a query. When you click the Advanced button at the top of the Query Builder, the Report Builder displays a text entry area in which you can enter CFML that generates a query. ColdFusion executes this tag at report execution time and passes the query result set to the report.

> **Note**
>
> When you use advanced query mode, the Query Builder does not create query fields automatically. create the associated query fields manually.

The CFML used in advanced query mode must include a query object whose name matches that in the Variable that contains the query object field. You can use any CFML tag that returns a query object or the `QueryNew` function. The CFML can use multiple query objects, but can only return one.

© 2014 Adobe Systems Incorporated. All rights reserved.

> **Note**
>
> If you set an empty variable (for example, <cfset name=" ">), the Report Builder throws a Report data binding error.

This example CFML uses the `cfhttp` tag to retrieve a query:

```
<cfhttp
url="http://quote.yahoo.com/download/quotes.csv?Symbols=csco,jnpr&format=sc1l1&ext=.
csv"
method="GET"
name="qStockItems"
columns="Symbol,Change,LastTradedPrice"
textqualifier="""
delimiter=","
firstrowasheaders="no">
```

Another possible use of advanced query mode is to test for passed parameters in the URL or FORM scopes and use those parameters to retrieve data, as the following example shows:

```
<!--- First look for URL param. URL overrides cfreportparam. --->
<cfif isDefined("url.deptidin")>
 <cfset param.deptidin = url.deptidin>
</cfif>

<!-- Then look for FORM param. Overrides URL param. --->
<cfif isDefined("form.deptidin")>
 <cfset param.deptidin = form.deptidin>
</cfif>

<cfquery name="CFReportDataQuery" datasource="cfdocexamples">
SELECTLastName, FirstName, Dept_ID
FROMEmployee
WHERE (Dept_ID = #param.deptidin#)
</cfquery>
```

Using report functions

Report functions are user-defined CFML functions that you code using the Report Function Editor and run in report fields. You can use them to format data (such as concatenating and formatting all the field that make up an address), to retrieve data, and for many other purposes.
Three built-in functions are unique to Report Builder: `InitializeReport`, `BeforeExport`, and `FinalizeReport`. For more information, see the Report Builder online Help.

Report Builder built-in functions

1. Select Report > Report Functions from the menu bar.The Report Function Editor displays.
2. Click the Add Default Functions icon (the first on the left).The built-in functions are added to the left pane.
3. Select a function from the left pane.Commented code associated with the function appears in the right pane.
4. Modify the code and click OK.

© 2014 Adobe Systems Incorporated. All rights reserved.

Create a report function

1. Select Report > Report Functions from the menu bar.The Report Function Editor displays.
2. Click the plus sign to add a new report function.The Add Report Function dialog box displays.
3. Specify a name and click OK.
4. The Report Function Editor places a `cfreturn` tag in the text entry area.
5. Code the function, and click OK. The function is a ColdFusion user-defined function so all UDF rules and features are available for use. The following example shows a report function that concatenates address fields:

```
<cfargument name="Name" required="yes"/>
<cfargument name="Address1" required="yes"/>
<cfargument name="Address2" required="yes"/>
<cfargument name="City" required="yes"/>
<cfargument name="State" required="yes"/>
<cfargument name="Zip" required="yes"/>

<cfset variables.CRLF = Chr(13) & Chr(10)>
<cfset variables.ResultVar="">

<cfif Trim(arguments.Name) NEQ "">
 <cfset variables.ResultVar='#arguments.Name#'>
</cfif>
<cfif Trim(arguments.Address1) NEQ "">
 <cfif variables.ResultVar NEQ "">
  <cfset variables.ResultVar='#variables.ResultVar & variables.CRLF#'>
 </cfif>
 <cfset variables.ResultVar='#variables.ResultVar & arguments.Address1#'>
</cfif>
<cfif Trim(arguments.Address2) NEQ "">
 <cfif variables.ResultVar NEQ "">
  <cfset variables.ResultVar='#variables.ResultVar & variables.CRLF#'>
 </cfif>
 <cfset variables.ResultVar='#variables.ResultVar & arguments.Address2#'>
</cfif>
<cfif variables.ResultVar NEQ "">
 <cfset variables.ResultVar='#variables.ResultVar & variables.CRLF#'>
</cfif>

<cfset variables.ResultVar='#variables.ResultVar & arguments.City & ", " &
arguments.State & " " & arguments.Zip#'>

<cfreturn variables.ResultVar>
```

Use a report function

1. Place a dynamic field on the appropriate report band.The Add Field dialog box displays.
2. Specify Manually Entered Expression, and click OK.The Expression Builder displays.
3. Specify "report.*functionname*", and click OK.

Using expressions

Many elements of the Report Builder (including query fields, calculated fields, input parameters, images, and report object attributes) are single operand ColdFusion expressions. Because these elements are expressions, you can

© 2014 Adobe Systems Incorporated. All rights reserved.

manipulate them with CFML functions.
The Expression Builder is a graphical interface that lets you quickly apply CFML functions to Report Builder elements. Uses for the Expression Builder include the following:

- Many of the report object attributes (such as PrintWhen) accept expressions, which you can associate with query parameters, input parameters, or ColdFusion page variables. You can tie report attributes and columns to display based on run-time data or user preference.
- Concatenating fields
- Formatting fields
- Calculated fields
- Accessing and displaying ColdFusion page variables and scopes
  For information on using the Expression Builder, see Report Builder online Help.
  For more information on expressions, see Using Expressions and Number Signs.

Using charts

Charts can help clarify large or complex data sets. The Report Builder lets you place a chart in any report band and supports many types of charts.
To add a chart to a report, you use the Chart Wizard, which steps you through the chart building process. The Chart Wizard, which is fully integrated with the Query Wizard to facilitate database-driven charts, helps you define the chart type, the data used for the report and other formatting options.

> ✅ As you use the Chart Wizard to choose and define the various aspects of a given chart, the Report Builder uses RDS to generate chart images in real time. However, the data in these chart images is not real.

The Chart Wizard includes the following panels:

- **Chart Types**: Select the chart type (for example, bar) and subtype (for example, 3D-stacked).
- **Chart Series**: Select the data for the series. When you add a series, the Report Builder lets you hard-code series data or open the Query Builder to populate the series using a database query.
- **Chart Formatting**: Specifies title and series, general appearance, 3D appearance, lines and markers, and font.

> ✅ The data you specify through the Chart Wizard corresponds to the attributes specified in the cfchart, cfchartseries, and cfchartdata tags. For more information on these tags, see the *CFML Reference.*

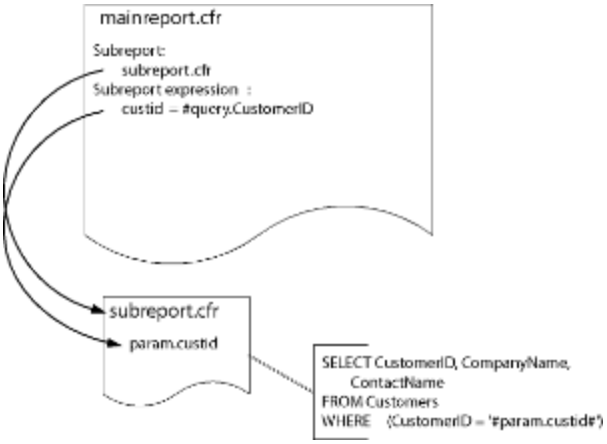
For complete information on ColdFusion charting capabilities, see Creating Charts and Graphs. For more information on charting using the Report Builder, see Report Builder online Help.

Using subreports

Subreports let you nest a report within your report. The data that you display in a subreport is typically related to the data in the main report. You enable this display by passing one or more subreport parameters to the subreport. However, the data displayed in a subreport can also be unrelated to the data in the main report.
Reasons to use subreports including the following:

- You prefer to avoid complex SQL, such as a RIGHT OUTER JOIN.
- Your report requires data from multiple databases.
  The following example shows the use of subreport parameters and the relationship between a report and a subreport:

© 2014 Adobe Systems Incorporated. All rights reserved.



Note

Although the Report Builder supports multiple levels of nesting, it displays one level of nesting only.

For additional information on subreports, see the Report Builder online Help.

Defining a subreport

You can define a subreport and include it in a report, or you can define it as part of inserting the subreport in the main report.
A subreport has the following characteristics:

- Data displayed in the detail band only. A subreport uses no header or footer bands.
- If the subreport is related to the main report, it must include an internal query that uses a SELECT statement with a WHERE clause specifying the name of the input parameter used in the main report's Subreport Expression property.
  If you have already defined a subreport, you add it to the main report and define subreport parameters, as necessary.

Add an existing subreport

1. Define or open your main report.
2. Click the Subreport icon in the toolbox.
3. Drag an area for the subreport in the desired report band.
4. Select From An Existing Report, specify the subreport, and click Next.
5. Select the fields in the main report that correspond to fields in the subreport and click Next.
6. Click Finish.The Report Builder adds the subreport to the main report, saving the report to subreport mappings as subreport parameters.
7. To modify subreport parameter settings, select the subreport and click Subreport Parameters in the Properties panel.
   If you are certain about the data required for a subreport, you can define a new subreport while adding it to the main report.

Add a new subreport

1. Define or open your main report.
2. Click the Subreport icon in the toolbox.
3. Drag an area for the subreport in the report band.
4. Select As A New Report and click Next.
5. Click Query Builder.
6. Select the tables and columns for the subreport.

© 2014 Adobe Systems Incorporated. All rights reserved.

7. Specify a WHERE clause for the report by using the Condition and Criteria columns for the key columns. Specify a WHERE for Condition and either `='#CFVariable#'` (string column) or `=#CFVariable#` (numeric column) for Criteria, and then overtype `CFVariable` with the name of the input parameter for the subreport (you define the input parameter name later in the procedure.)
8. Click Save, and then click Next.
9. Specify grouping fields, if appropriate for your subreport, and click Next.
10. Specify Free Form or Grid, and click Next.
11. Specify Only Detail Band, and click Next.
12. Specify a color scheme, and click Next.
13. Specify headings, as appropriate, and click Next.
14. For each parameter required by the subreport, specify the following:
    - Parameter name.
    - Associated value from the main report (select from the pop-up menu).
    - Data type.
15. Click Next.
16. Specify a fully qualified filename for the subreport, and then click Next.
17. Click Finish.Report Builder adds the subreport to the main report. Report Builder lets you change subreport name and modify subreport parameters in a main report.

Modify subreport settings

1. Click the subreport element in the main report.
2. To change the subreport, modify Subreport Expression.
3. To modify subreport parameters:
    a. Click the Subreport Parameters property.
    b. Click the ... button.
    c. Add, modify, or delete subreport parameters, and click OK.

© 2014 Adobe Systems Incorporated. All rights reserved.

Creating a simple report

The following example shows how to create a simple report by using the Report Wizard and then modifying it. The example uses the cfartgallery database, which is installed with ColdFusion.
The example shows how to perform the following tasks:

- Create a base report by using the Report Wizard and the Query Builder.
- Use the Expression Builder to modify the data presentation in the report.
- Modify the display text for column data.
- Add a text field to the report and format text and data elements by using report styles.
- Add an image file and images from a database.
- Create and add a calculated field to display the total sales by artist.
- Add group-level and report-level pie charts that show the ratio of sold and unsold art for each artist and for all the artists in the database.
- Export report styles to a Cascading Style Sheet (CSS) file.

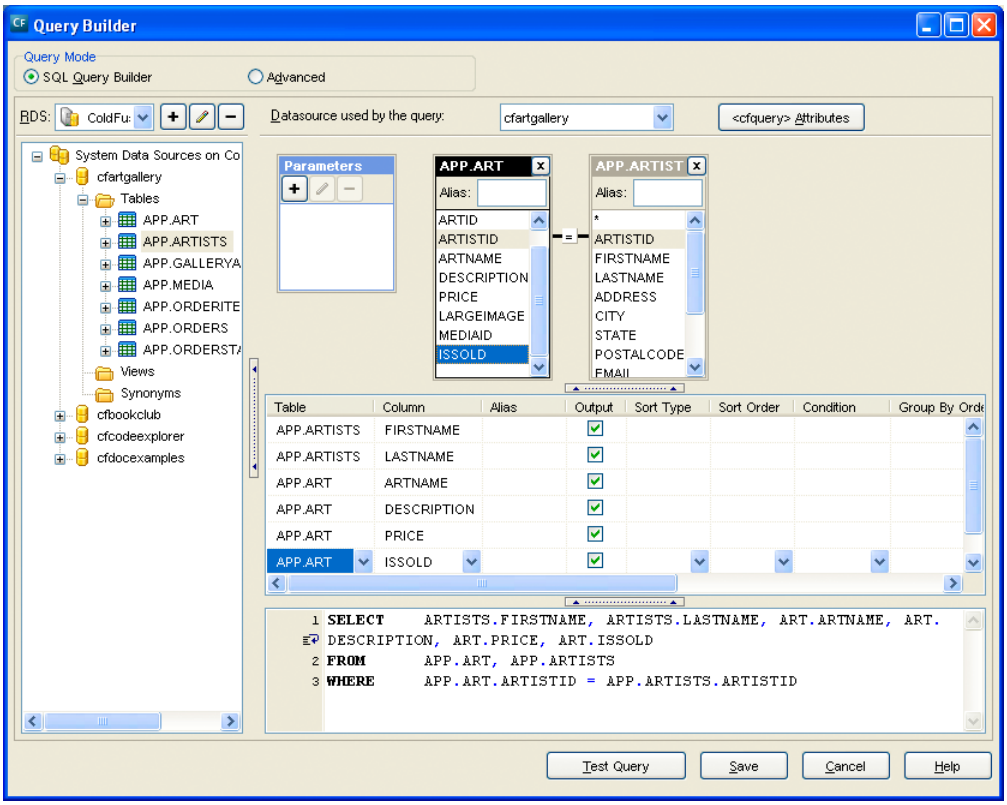
Create a report by using the Report Wizard

1. Start Report Builder.
2. Select Report Creation Wizard and click OK.
3. Click the Query Builder button:

> **Note**
>
> If you have RDS configured, provide the RDS password.

   a. From the list of data sources in the database pane, expand the cfartgallery database.
   b. Expand the Tables folder.
   c. Double-click the APP.ART table in the database pane. Report Builder adds the APP.ART table to the table pane.
   d. Double-click the APP.ARTISTS table in the database pane. Report Builder adds the APP.ARTISTS table to the table pane. Notice that it automatically creates the join between the two tables based on the ARTISTID column.
   e. In the APP.ARTISTS table, double-click the FIRSTNAME and LASTNAME columns. The Query Builder adds the fields to the select statement in the SQL pane.
   f. In the ART table, double-click the ARTNAME, DESCRIPTION, PRICE, and ISSOLD columns. The following example shows the completed query in the Query Builder:

© 2014 Adobe Systems Incorporated. All rights reserved.



g. Click the Test Query button to preview the results.
h. Close the test query window and click the Save button in the Query Builder window.

2. Double-click the FIRSTNAME column to add it to the Non-printed Fields pop-up menu and click the Next button.
3. In the Available Fields list, double-click LASTNAME to group the records by the artists' last names.
4. Click the Next button three times to accept the default values.
5. Choose Silver and click the Next button.
6. Change the title of the report to Sales Report and click the Finish button. The Report Creation Wizard generates the report and displays it in the Report Builder workspace.
7. Choose File > Save As and save the report as ArtSalesReport1 in the default directory. Report Builder automatically adds the CFR extension.
8. Press F12 to preview the report. Report Builder displays the records grouped by the artists' last names.
9. Click the close box to close the Preview Report window and return to the Report Builder workspace.

Changing the column heading labels

By default, the Report Wizard uses the column name for the column headers in the report, but you can change the label text for column headings.

Edit the heading label text

1. Double-click the LASTNAME field in the Column Header band.
2. Replace the column name with Artist Name, and click OK.
3. Replace the remaining column labels as follows:
   - ARTNAME > Title
   - DESCRIPTION > Description
   - PRICE > Price
   - ISSOLD > Sold?

Using expressions to format data

Use the Expression Builder to perform the following tasks:

© 2014 Adobe Systems Incorporated. All rights reserved.

- Change the display of the ISSOLD value to a yes/no expression. By default, Report Builder displays 0 (not sold) or 1 (sold) for the ISSOLD column based on how the data is stored in the database. You can use a function to change the display to yes or no.
- Change the value of the PRICE column to a dollar format.
- Concatenate the artists' first and last names. Even though the FirstName field is a nonprinted field in the report, you can add it to an expression because it is part of the SQL query that you created.

Change a Boolean value to yes/no

1. Double-click the query.ISSOLD element in the detail band. Report Builder displays the Expression Builder for that element.
2. In the Expression Builder, expand the Functions folder.
3. Choose Display and Formatting from the Functions list. Report Builder displays the list of functions in the right pane of the Expression Builder.
4. Double-click YesNoFormat from the list of functions. Report Builder automatically completes the following expression in the expression pane:

```
YesNoFormat(query.ISSOLD)
```

5. Click OK to close the Expression Builder and return to the report.
6. Choose File > Save to save your changes to the report.
7. Press F12 to preview the report. Yes or no appears in the Sold? column based on whether the artwork sold.

Display numbers in dollar format

1. Double-click the field in the PRICE column of the detail band.
2. In the expression pane, change the expression to the following text:

```
DollarFormat(query.PRICE)
```

3. Click OK to close the Expression Builder and return to the report.

Concatenate the FIRSTNAME and LASTNAME fields

1. Double-click the query.LASTNAME field in the LASTNAME group header.
2. In the Expression Builder, type the following expression:

```
query.FIRSTNAME &" "& query.LASTNAME
```

   Notice that the Expression Builder prompts you with the available field names as you type.

3. Click the OK button in the Expression Builder.
4. Choose File > Save from the Report Builder menu bar to save your changes to the report.
5. Press F12 to preview the report.Report Builder displays the first and last name for each of the artists. Notice that the report still is grouped alphabetically by last name.
6. Close the preview window.

Adding page breaks before group changes

Create a page break so that each artist name starts on at the top of a page in the report output.

© 2014 Adobe Systems Incorporated. All rights reserved.

Add page breaks between artist names

1. Choose Report > Group Management from the main menu bar. The Group Management dialog box appears with LASTNAME selected.
2. Click the Edit button.
3. Select the Start New Page option and click OK.

Adding a calculated field

Calculate the sum of the artwork sold by artist

1. Choose Window > Fields and Parameters.
2. Report Builder displays the Fields and Parameters panel.
3. Expand the list of calculated fields.
4. With Calculated Fields selected, click the button at the upper edge of the Fields and Parameters panel.
5. Make the following changes in the Add Calculated Field dialog box:
   a. Change the name of the calculated field to **Sold**.
   b. Change the label text to **Sold**.
   c. Change the Data Type to Float.
   d. Change the Calculation to Sum.
   e. In the Perform Calculation On field, enter the following expression:

```
Iif(IsBoolean(query.ISSOLD) and query.ISSOLD, query.Price,0)
```

   This expression multiplies the total price of the artwork per artist by the number of items sold to calculate the total sales per artist. If the ISSOLD value for a record is 1 (sold), the value is multiplied by 1 and added to the total; if the ISSOLD value for a record is 0 (unsold), the value is multiplied by 0.

   f. Change the Reset Field When value to **Group**.
   g. Change the Reset Group value to **LASTNAME**, and click OK. Report Builder adds the calculated field definition in the Fields and Parameters panel.

Add the calculated field to your report

1. Insert a field in the LASTNAME Footer band.
2. In the Add Field dialog box, select calc.Sold from the pop-up menu.
3. In the Expression Builder, type the following code:

```
DollarFormat(calc.Sold)
```

4. Press F12 to preview the report. Report Builder displays the sum of the artwork sold for each artist.

Adding and formatting fields

You can add a text field to your report and define a style for it. When you define a style, you can reuse it throughout your report or export the style so that you can use it in other reports. Also, you can override report styles at run time by using the `cfreport` and the `cfreportparam` tags. For more information, see Overriding report styles.

Add a text field

1. In the Controls toolbox on the left side of the Report Builder window, click the text icon (the button with abc on it) and place the text field to the left of the calculated field in the LASTNAME footer.
2. In the Edit Label dialog box, type **Total Sales**, and click OK.

© 2014 Adobe Systems Incorporated. All rights reserved.

Create a style

1. Choose Window > Report Styles from the main menu.
2. Click the button.
3. In the Name field, enter GroupFooter.
4. Click the Color and Style tab and change the color to **#9999CC**.
5. Click the Font tab and change the Font to **Tahoma** and click the bold option. Then click OK. Report Builder adds GroupFooter style to the pop-up menu of available styles in the report.
6. Choose File > Save from the menu bar to save your changes to the report.

Apply the style to text and data elements in the report

1. Select the Total Sales text box in the LASTNAME Footer band.
2. Choose Window > Properties Inspector.
3. Choose GroupFooter from the Style pop-up menu.
4. Select the calculated field element and apply the GroupFooter Style to it.
5. Press F12 to preview your report:

Adding images

When you add images with Report Builder, you can perform the following types of tasks:

- Replace the company name text box with a company logo in the report header.
- Use the Query Builder to add images from a database.
- Display the report in RTF format for faster display.

Add a logo to the report header

1. Select the Company Name text box located in the header band preceding Sales Report.
2. Choose Edit > Cut to remove the text box from the report.
3. Click the Add Image icon in the Controls toolbox. (The icon has a picture of a tree on it.)
4. Drag the mouse in the header band preceding the Sales Report text box. When you release the mouse, the Image File Name dialog box appears.
5. Navigate to the Art World logo file:C:\ColdFusion9\wwwroot\cfdocs\getting_started\photos\somewhere.jpg
6. Click Open. Report Builder displays the Art World logo in the area that you selected.
7. With the image selected in the workspace, choose Windows > Properties Inspector. The Properties Inspector for the image appears:
    a. Under Colors and Style, change the Transparency to Transparent.
    b. Under Formatting, change Scale Image to Retain Shape.
8. In the Header band, control-click the logo image and the Sales Report text box in the workspace to select them.
9. Click the Align Left Sides icon in the Controls toolbox.
10. Choose File > Save to save your changes.
11. Press F12 to preview the report.
12. Close the preview window and readjust the image size and location as needed.

Add images from a database

1. From the menu bar, choose Report > Report Query.
2. In the Art table, double-click LARGEIMAGE. The Query Builder adds the LARGEIMAGE column to the select statement.
3. Click the Test Query button. A list of image filenames appears to the right of the ISSOLD column.
4. Close the Test Query window and click the Save button in the Query Builder.
5. In the Report window, expand the Detail band by clicking the lower splitter bar and dragging down.
6. Click the Add Image icon in the Controls toolbox and drag the mouse in Detail band of the report to the left of the query.ARTNAME field. When you release the mouse, the Image File Name dialog box appears.
7. Navigate to the cfartgallery images directory:C:\ColdFusion9\wwwroot\cfdocs\images\artgallery
8. In the File Name field, type **#query.largeimage#**.

© 2014 Adobe Systems Incorporated. All rights reserved.

9. Click the Open button. Report Builder adds the column to the Detail band of the report.
10. Align the image column with the top of the Detail band.
11. With the image element selected in the detail band, choose Window > Properties Inspector.
12. Change the following properties:
    a. Transparency: Transparent.
    b. Scale Image: Retain Shape. This option scales the images proportionately within the bounding box.
    c. Error Control: No Image. This option ensures that Report Builder displays blank images rather than generates an error for images missing from the database.
    d. Using Cache: False. This option refreshes the display each time you preview the report output in the browser.
13. Choose File > Save to save your changes.

Change the report output format

1. Choose Report > Report Properties from the menu bar.
2. From the Default Output Format pop-up menu, choose RTF. Use this format for faster display in a web browser.
3. Click OK to close the Report Properties dialog box and return to the report.
4. Choose File > Save to save your changes.
5. Press F12 to preview the report. The images are displayed beneath Artist name and to the left of the art title.
6. Change the Default Output Format to HTML and preview the results.

Adding charts

You can use the Chart Builder to add two pie charts to your report: the first pie chart shows the total dollar amount of the art sold versus the total dollar amount unsold art for each artist; the second pie chart shows the sum of artwork sold versus unsold for all of the artists.
The two pie charts are the same except for the scope. To apply a pie chart to a group (the ratio of sold to unsold art for each artist), add the pie chart to the group footer band. To apply the pie chart to the report (the ratio of sold to unsold art for all artists), add the pie chart to the report footer band.
In Adding a calculated field, you added a calculated field for the total dollar amount of artwork sold. Before you can create the pie chart for this example, create a second calculated field for the total dollar amount of unsold art.

Add a calculated field for the sum of unsold art

1. Choose Window > Fields and Parameters.
2. Select the Calculated Fields heading in the Fields and Parameters panel.
3. Click the icon at the upper edge of the panel:
    a. In the Name field, type **Unsold**.
    b. In the Default Label Text field, type **Unsold**.
    c. In the Data Type field, choose Big Decimal from the pop-up menu.
    d. In the Calculation field, choose Sum from the pop-up menu.
    e. In the Perform Calculation On field, enter the following expression to calculate the dollar amount of unsold art:

```
Iif(IsBoolean(query.ISSOLD) and not(query.ISSOLD), query.Price,0)
```

    f. In the Reset Field When field, choose Group from the pop-up menu.
    g. In the Reset Group field, choose LASTNAME.
    h. Click OK to close the Add Calculated Field dialog box and return to the report.
4. Choose File > Save from the menu bar to save your changes to the report.

Add a pie chart to the group footer

1. Expand the LASTNAME Footer band.

© 2014 Adobe Systems Incorporated. All rights reserved.

2. Choose Insert > Chart from the Report Builder menu bar:
    a. Choose Pie from the Base Chart Type list. The Chart Sub-Type appears to the right of the Base Chart Type.
    b. Choose the 3D chart.
3. Click the Next button. Then click the Add button:
    a. In the Series Label field, type **Total Sales**.
    b. In the Paint Style field, choose Light.
    c. In the Data Label field, choose Value.
    d. In the Color List, type **Teal,Gray**.
    e. In the Chart Data Source area, ensure that the Data From A Fixed List of Values option is selected.
4. Click the Add button:
    a. In the Label field, type **Sold**.
    b. In the Value field, choose #calc.Sold# from pop-up menu.
    c. Click OK.
5. Click the Add button again:
    a. In the Label field, type **Unsold**.
    b. In the Value field, choose #calc.Unsold# from the pop-up menu.
    c. Click OK twice to return to the Chart Series dialog box.
6. Click the Next button. In the Chart Formatting dialog box, click the Titles & Series tab and make the following changes:
    a. In the Chart Title field, type **Total Sales** for #query.LASTNAME#.
    b. In the X Axis Title field, type **Sold**.
    c. In the Y Axis Title field, type **Unsold**.
    d. In the Label Format field, choose Currency from the pop-up menu.
    e. Click the 3-D Appearance tab and ensure that Show 3-D is selected.
7. Click the Font tab and make the following changes:
    a. Change the Font Name to Arial.
    b. Change the Font Size to 9.
8. Click the Finish button. Report Builder adds a place holder for the pie chart in the report.
9. Resize and move the chart to the desired location within the LASTNAME Footer band.
10. Choose File > Save to save your changes to the report.
11. Press F12 to preview the report.

Add a pie chart to the report footer

1. Create two calculated fields to use in the report footer pie chart with the following parameters:

Name	**TotalSold**	**TotalUnsold**
Default Label Text:	Total Sold	Total Unsold
Data Type:	Big Decimal	Big Decimal
Calculation:	Sum	Sum
Perform Calculation On:	`Iif(IsBoolean(query.ISSOLD) and query.ISSOLD, query.Price,0)`	`Iif(IsBoolean(query.ISSOLD) and not(query.ISSOLD), query.Price,0)`
Initial Value:	0	0
Reset Field When:	Report (Changes)	Report (Changes)

© 2014 Adobe Systems Incorporated. All rights reserved.

Reset Group:	LASTNAME	LASTNAME

1. Expand the Report Footer band, which is located directly below the Page Footer band.
2. Copy the pie chart from the Group Footer and paste it in the Report Footer.
3. Double-click the pie chart and click the Next button.
4. Double-click Total Sales to display the Edit Chart Series dialog box.
5. Change the Series Label to Total Sales for Artists.
6. Change the chart series values:

Label	Value
Sold	`#calc.TotalSold#`
Unsold	`#calc.TotalUnsold#`

1. Click the Next button, and then Click the Title & Series tab.
2. Change the Chart Title to **Total Sales for Artists**, and click Finish.
3. Choose File > Save from the menu bar to save your changes to the report.
4. Press F12 to preview the report.
   The Total Sales for Artists pie chart appears only on the last page of the report. Verify that the calculations are correct.

Using Cascading Style Sheets

The Report Creation Wizard automatically creates and applies the following styles to your report:

- ReportTitle
- CompanyName
- PageTitle
- ReportDate
- SubTitle
- DetailData (default style)
- DetailLabel
- PageFooter
- RectangleStyle
- LineStyle
  The instructions on Adding and formatting fields show how to add a field called GroupFooter and apply it to a text field and a data field in the GroupFooter band. You can export the styles in a report to a CSS file. Report Builder automatically generates the CSS code for the styles. This technique is an efficient way to maintain a single set of styles to use with multiple reports. You can modify the styles in the CSS file by using any text editor and either import the CSS file in Report Builder or override the styles in the report at run time.

Export report styles to a CSS file

1. Choose Window > Report Styles.
2. Click the export icon (the icon with the orange arrow).
3. In the File Name field, type **artstyles**. Report Builder automatically adds the CSS extension.
4. Navigate the artStyles.css file and double-click it to open it. The following example shows the generated CSS code:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
ReportTitle
{
  color:Black;
  font-size:24pt;
}
CompanyName
{
  color:#6188A5;
  font-weight:bold;
}
PageTitle
{
  color:#333333;
  font-size:14pt;
  font-weight:bold;
}
ReportDate
{
  color:#333333
}
SubTitle
{
  color:#6089A5;
  font-size:12pt;
  font-weight:bold;
}
DetailLabel
{
  color:Black;
  background-color:#E3EDEF;
  font-weight:bold;
}
DetailData
{
  default-style:true;
  color:Black;
  line-size:thin;
}
PageFooter
{
  color:#2F2F2F;
  font-size:8pt;
}
RectangleStyle
{
  color:#E3EDEF;
  background-color:#E3EDEF;
}
LineStyle
{
  color:#CCCCCC;
  background-color:#CCCCCC;
}
GroupFooter
{
  color:Blue;
  font-weight:bold;
  font-family:Tahoma;
```

© 2014 Adobe Systems Incorporated. All rights reserved.

5. Change the ReportTitle style `color` attribute to **Red** and add the `font-weight`attribute, as the following code shows:

```
ReportTitle
{
  color:Red;
  font-size:24pt;
  font-weight: bold;
}
```

6. Save the CSS file.
   Also, you can override report styles from ColdFusion. Form more information, see Overriding report styles.

> **Note**
>
> If you add a style to the CSS file, add a style with the same name to the report in Report Builder. Also, Report Builder does not support all CSS styles. For more information, see the cfreport tag in the CFML Reference.

Import the CSS file

1. Choose Window > Report Styles.
2. Click the import styles icon (the one with the blue arrow).
3. Navigate to the location of the artStyles.css file, and click OK. Report Builder automatically updates the report style definition and applies the updated style to report title.
4. Press F12 to preview the report.

Overriding report settings at run time

You can use the `cfreport` tag in ColdFusion to override report settings in a Report Builder report at run time. The examples use the CFR file that you created in Creating a simple report.

Overriding the report query

This example filters the data in the report based on the login ID of the artist. When the artist logs on, the report displays the data and pie chart for that artist. The report also includes the pie chart with data from all the artists. The following code creates a simple login page in ColdFusion. The form uses artist's last name as the user ID. (The code does not include password verification):

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<h3>Artist Login Form</h3>
<p>Please enter your last name and password.</p>
<cfform name="loginform" action="artSalesReport.cfm" method="post">
<table>
 <tr>
  <td>Last Name:</td>
  <td><cfinput type="text" name="username" required="yes" message="A username is
   required."></td>
 </tr>
 <tr>
  <td>Password:</td>
  <td><cfinput type="password" name="password" required="yes" message="A password is

   required."></td>
 </tr>
</table>
 <br />
 <cfinput type="submit" name="submit" value="Submit">
</cfform>
```

On the processing page, add a query like the one you created in the Report Builder report. The ColdFusion query must contain at least all of the columns included in the Report Builder query; however, the ColdFusion query can contain additional data.
The query in the following example selects all of the data from the ART and ARTISTS tables based on the artist's last name. The `cfreport` tag uses the pathname of the CFR file as the report template.

```
<cfquery name="artsales" datasource="cfartgallery">
SELECT *
FROMAPP.ART, APP.ARTISTS
WHERE APP.ART.ARTISTID = APP.ARTISTS.ARTISTID
   AND APP.ARTISTS.LASTNAME= <cfqueryparam value="#FORM.username#">
ORDER BY ARTISTS.LASTNAME
</cfquery>

<cfreport query="#artsales#" template="ArtSalesReport1.cfr" format="RTF"/>
```

ColdFusion displays the report for the artist in RTF format. Notice that the value of the `format` attribute overrides the Default Output format defined in the CFR file.

Exporting the report in HTML

To generate a report in HTML and display it directly in the browser, change the format attribute to HTML:

```
<cfreport template="ArtSalesReport1.cfr" format="HTML"/>
```

ColdFusion automatically generates a temporary directory where it stores all of the image files in the report (charts are saved as PNG files). The location of the temporary directory is:
C:\ColdFusion9\tmpCache\CFFileServlet_cfreport_report_unique_identifier_

© 2014 Adobe Systems Incorporated. All rights reserved.

You can specify when the temporary directory is removed from the server by using the `CreateTimeSpan` function as a value for the `resourceTimespan` attribute:

```
<cfreport query="#artsales#" template="ArtSalesReport1.cfr" format="HTML"
resourceTimespan="#CreateTimeSpan(0,1,0,0)#"/>
```

You can specify the time span in days, hours, minutes, and seconds. In this example, the temporary directory is deleted after one hour. For more information, see the *CFML Reference*.
To export the report output to an HTML file, specify the `filename` attribute. The following code writes the report output to an HTML file called artSales.html:

```
<cfreport template="ArtSalesReport1.cfr" format="HTML" filename="artSales.html"
overwrite="yes"/>
```

ColdFusion creates an image directory relative to the HTML output file in the format *filenamefiles. In this example, ColdFusion automatically generates PNG files for the charts in the report and saves them to a directory called artSales_files. Also, it generates copies of all of the JPG images extracted from the cfartgallery database and stores them in the artSales_files directory. For more information, see the _CFML Reference*.

Overriding report styles

To override the report styles in a report, specify the `style` attribute of the `cfreport` tag. The value must contain valid CSS syntax, the pathname to a CSS file, or a variable that points to valid CSS code. The CSS style names must match the report style names defined in Report Builder.
The following code shows how to override the styles in the ArtSalesReport1.cfr report with the styles defined in the artStyles.css file:

```
<cfreport template="ArtSalesReport1.cfr" style="artStyles.css" format="PDF"/>
```

The following code shows how to apply a CSS style as a value of the `style` attribute:

```
<cfreport template="ArtSalesReport1.cfr" style='ReportTitle {defaultStyle: false;
 font-family:"Tahoma"; color: "lime";}' format="FlashPaper">
</cfreport>
```

The following code shows how to create a variable called *myStyle* and use it as a value of the `style` attribute:

```
<cfset mystyle='DetailData { defaultStyle: true; font-family: "Tahoma"; color:
##00FFF0;}'>
<cfreport template="ArtSalesReport1.cfr" style="#mystyle#" format="HTML">
</cfreport>
```

For more information, see the cfreport tag in the *CFML Reference*.

© 2014 Adobe Systems Incorporated. All rights reserved.

#back to top

© 2014 Adobe Systems Incorporated. All rights reserved.

Creating Slide Presentations

You can use Adobe ColdFusion to create slide presentations.

© 2014 Adobe Systems Incorporated. All rights reserved.

About ColdFusion presentations

ColdFusion lets you create dynamic slide presentations from source files and from CFML and HTML code on a ColdFusion page. You can use data extracted from a database to populate the slide content, including graphs and charts. Also, you can add images, audio tracks, and video clips to each slide in the presentation. ColdFusion provides three tags for creating slide presentations:

Tag	Description
cfpresentation	Defines the look of the presentation and determines whether the presentation is saved to files or run directly in the client browser.
cfpresentationslide	Defines the content of the slide from one of the following: • A SWF file • An HTML file • A URL that returns HTML content • HTML and CFML code in the `cfpresentationslide` start and end tags
cfpresenter	Provides information about the person presenting a slide. You can assign a presenter to one or more slides. Presenter information is displayed in the control panel for the duration of the slide.

You specify at least one slide for the presentation and can assign each presenter to one or more slides. The following example shows a slide presentation with content from four different sources and two presenters:

```
<cfpresentation title="myPresentation">
 <cfpresenter name="Tuckerman" title="V.P. of Marketing"
   email="tuckerman@company.com">
 <cfpresenter name="Anne" title="V.P. of Sales" email="anne@company.com">
 <cfpresentationslide src="slide1.swf" title="Overview" duration="10"
   presenter="Anne"/>
 <cfpresentationslide src="slide2.htm" title="Q1 Sales" duration="30"
   presenter="Anne"/>
 <cfpresentationslide src="http://www.markettrends.com/index.htm"
   title="Market Trends" duration="30" presenter="Tuckerman"/>
 <cfpresentationslide title="Summary" duration="10">
  <h3>Summary</h3>
   <ul>
    <li>Projected Sales</li>
    <li>Challenges Ahead</li>
    <li>Long Term Goals</li>
   </ul>
 </cfpresentationslide>
</cfpresentation>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

Note

The `cfpresentationslide` tag requires an end tag. If you specify a source file as the slide content, use the end slash as a shortcut for the end tag.

When the presentation runs, the slides appear in the order they are listed on the ColdFusion page for the duration specified in each slide. The presenter information is displayed in a control panel next to the slide to which it is assigned.

© 2014 Adobe Systems Incorporated. All rights reserved.

Creating a slide presentation

Use the `cfpresentation` tag to customize the look of the slide presentation. Customizations can include the control locations and the colors used in the presentation interface, as the following example shows:

```
<cfpresentation title="Sales Presentation" controlLocation="left"
primaryColor="##0000FF" shadowColor="###000033" textColor="##FFFF00"
showNotes="yes">
```

The title appears at the top of the control panel. The color settings affect the presentation interface, but not the format of the slides within the presentation. Set the `showNotes` attribute to `yes` to display text notes that are defined for individual slides.
If you do not specify a directory, as in the previous example, ColdFusion runs the presentation directly in the client browser. The presentation uses files written to a temp directory on the server. To save the presentation, specify an absolute path or a directory relative to the CFM page. (ColdFusion does not create the directory; it must exist already.) In the following example, the presentation files are stored in the salesPresentation directory on the local drive:

```
<cfpresentation title="Sales Presentation" directory="c:\salesPresenation">
```

ColdFusion automatically generates the following files necessary to run the presentation and saves them in the specified directory:

- components.swf
- index.htm
- loadflash.js
- viewer.swf
  Also, ColdFusion creates a subdirectory called data where it stores the following files:
- srchdata.xml (which creates the search interface)
- vconfig.xml
- viewer.xml
- A SWF file generated for each slide in the presentation
- Copies of the media files referenced in the presentation slidesMedia files can include JPEG files, FLV and SWF video files, and mp3 audio files. To run the presentation that you saved to files, double-click the index.htm file.

Note

ColdFusion does not overwrite the files referenced by the slides in the presentation; changes to the generated presentation files do not affect the source files.

© 2014 Adobe Systems Incorporated. All rights reserved.

Adding presenters

Optionally, you can add one or more presenters under the `cfpresentation` tag. ColdFusion displays the presenter information in the control panel for the current slide to which it is assigned. A slide does not require a presenter.
Use the `cfpresenter` tag to specify personal information. This information can include a title, an e-mail address, a logo and an image of the person, as the following code shows:

```
<cfpresentation title="Sales Presentation">
<cfpresenter name="Anne" title="V.P. of Sales" biography="Anne Taylor has been a top
seller at Widgets R Us for five years." logo="images/logo.jpg"
image="images/ataylor_empPhoto.jpg" email="ataylor@widgetsrus.com">
```

The `name` attribute is required. You use this value to assign the presenter to one or more slides. To assign a presenter to a slide, use the `cfpresenter` tag `name` attribute value as the `cfpresentationslide` tag `presenter` attribute. The following example creates a presenter named Tuckerman and assigns him to a slide called Overview:

```
<cfpresentation title="Sales Presentation">
<cfpresenter name="Tuckerman" title="V.P. of Marketing">
<cfpresentationslide title="Overview" src="overview.swf" presenter="Tuckerman"
duration="10"/>
...
</cfpresentation>
```

Note

Assign presenters explicitly to slides. To assign a presenter to more than one slide, use the presenter name in each of the `cfpresentationslide` tags.

When you assign a presenter to a slide, the presenter information is displayed in the control panel for the duration of the slide. Images must be in JPEG format and the files must be located in a path relative to the ColdFusion page. ColdFusion maps the `email` attribute value to the contact link in the control panel. This link opens an e-mail message in the local e-mail application when you click it.
The following code creates three presenters for a presentation and assigns two of the presenters to slides:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cfpresentation title="Sales Presentation">
 <cfpresenter name="Hannah" title="V.P. of Marketing" image="hannah.jpg">
 <cfpresenter name="Anne" title="V.P. of Sales" image="Anne.jpg">
 <cfpresenter name="Wilson" title="V.P. of Engineering"
   image="Wilson.jpg">
 <cfpresentationslide title="Overview" presenter="Hannah" duration="30"
   src="slide1.htm"/>
 <cfpresentationslide title="Q1 Sales" presenter="Anne" duration="15"
   src="slide2.htm"/>
 <cfpresentationslide title="Projected Sales" presenter="Anne"
   duration="15" src="slide3.htm" video="promo.flv"/>
 <cfpresentationslide title="Conclusion" src="slide4.htm"/>
</cfpresentation>
```

The presenter Hannah is assigned to one slide and Anne is assigned to two slides. The last slide in the presentation has no presenter assigned to it. Because Wilson is not assigned to a slide, his information does not appear in the presentation. In the second slide, Anne's photo is displayed in the control panel. In the third slide, however, the video called promo.flv runs in place of Anne's photo in the control panel for the duration of the slide. The video does not display in the slide.

Note

Videos must be in SWF or FLV format. You cannot specify audio and video for the same slide.

© 2014 Adobe Systems Incorporated. All rights reserved.

Adding slides

Use one `cfpresentationslide` tag for each slide in the presentation. The presentation runs the slides in the order they are listed beneath the `cfpresentation` tag. You can create content for a slide in one of the following ways:

Source	Description	Example
A SWF or HTML file	The file must be located on the system running ColdFusion. You can specify an absolute path or a path relative to the ColdFusion page.	`<cfpresentationslide title="slide 1" src="presentation/slide1.swf"/><cfpresentationslide title="slide 2" src="c:/presentation/slide2.htm"/>`
A URL	The URL must return HTML content.	`<cfpresentationslide title="slide 3" src="http://www.worldview.com/index.htm"/>`
HTML and CFML code on the ColdFusion page	Enclose the HTML and CFML code within the `cfpresentationslide` start and end tags.	`<cfpresentationslide><h3>Total Sales</h3><cfchart format="jpg" chartwidth="500" show3d="yes"><cfchartseries type="pie" query="artwork" itemcolumn="issold" valuecolumn="price"/></cfchart></cfpresentationslide>`

Creating content from source files

The following code creates a presentation with three slides from source files in different locations:

```
<cfpresentation title="Garden Mania" directory="gardenPresentation">
 <cfpresentationslide title="Seeds of Change" src="c:\gardening\seeds.html"
audio="media\hendrix.mp3" duration="30"/>
 <cfpresentationslide title="Flower Power" src="shockwave\flowerPower.swf"
duration="40"/>
 <cfpresentationslide title="Dig Deep" src="http://www.smartgarden.com/index.htm"
duration="15"/>
</cfpresentation>
```

In this example, ColdFusion generates the files required to run the presentation in the gardenPresentation directory. It generates a new SWF file in the data subdirectory from each of the slides. ColdFusion also copies the hendrix.mp3 file and saves it in the data subdirectory.

© 2014 Adobe Systems Incorporated. All rights reserved.

> **Note**
>
> Links within slides created from HTML files are not active.

Creating content from HTML and CFML code

If you do not specify a source file for a slide, create the content by using HTML or CFML in the `cfpresentations lide` tag body. The following presentation contains one slide with each with the following types of content:

- Generated from HTML
- Generated from HTML and CFML
- Extracted from an HTML file on an external website

```
<cfpresentation title="The Road Ahead">
<cfpresentationslide title="Yellow Bricks" audio="myaudio1.mp3" duration="10">
 <h3>Yellow Bricks</h3>
 <table cellpadding=10>
  <tr>
   <td>
    <ul>
     <li>Way to go Dorothy</li>
     <li>Making tracks</li>
     <li>No place like home</li>
    </ul>
   </td>
   <td><img src="../cfdocs/images/artgallery/maxwell01.jpg"/>
   </td>
  </tr>
 </table>
 </cfpresentationslide>
<cfpresentationslide title="Wild Ride" duration="5">
 <h3>Wild Ride</h3>
 <cfchart format="jpg" title="Who's Ahead" show3D="yes" chartHeight=500
chartWidth=500>
  <cfchartseries type="pyramid">
   <cfchartdata item="Dorothy" value=10>
   <cfchartdata item="Tin Man" value=30>
   <cfchartdata item="Scarecrow" value=15>
   <cfchartdata item="Lion" value=50>
   <cfchartdata item="Toto" value=5>
  </cfchartseries>
 </cfchart>
</cfpresentationslide>
<cfpresentationslide title="The Golden Age of Ballooning" duration="10"
src="http://www.balloning.com/index.htm"/>
</cfpresentation>
```

> **Note**
>
> The value for the format attribute of the `cfchart` tag must be JPG or PNG.

The content for slides is not limited to static data: you can generate content from information extracted from a database or a query of queries.

© 2014 Adobe Systems Incorporated. All rights reserved.

© 2014 Adobe Systems Incorporated. All rights reserved.

Sample presentations

This section provides two sample presentations.

Example 1

The following example creates a simple presentation that incorporates data retrieved from the cfdocexamples database. It shows how to perform the following tasks:

- Create slides generated from HTML and CFML.
- Add images to slides.
- Add charts and tables with data extracted from a database.
- Add audio tracks to individual slides.

```
<!--- The following query extracts employee data from the cfdocexamples
   database. --->
<cfquery name="GetSalaryDetails" datasource="cfdocexamples">
 SELECT Departmt.Dept_Name,
  Employee.FirstName,
  Employee.LastName,
  Employee.StartDate,
  Employee.Salary,
  Employee.Contract
 From Departmt, Employee
 Where Departmt.Dept_ID = Employee.Dept_ID
 ORDER BY Employee.LastName, Employee.Firstname
</cfquery>

<!--- The following code creates a presentation with three presenters. --->
<cfpresentation title="Employee Satisfaction" primaryColor="##0000FF"
glowColor="##FF00FF" lightColor="##FFFF00" showoutline="no">
 <cfpresenter name="Jeff" title="CFO" email="jeff@company.com"
   logo="../cfdocs/getting_started/photos/somewhere.jpg"
   image="../cfdocs/images/artgallery/jeff01.jpg">
 <cfpresenter name="Lori" title="VP Marketing" email="lori@company.com"
   logo="../cfdocs/getting_started/photos/somewhere.jpg"
   image="../cfdocs/images/artgallery/lori01.jpg">
 <cfpresenter name="Paul" title="VP Sales" email="paul@company.com"
   logo="../cfdocs/getting_started/photos/somewhere.jpg"
   image="../cfdocs/images/artgallery/paul01.jpg">

<!--- The following code creates the first slide in the presentation
   from HTML. --->
 <cfpresentationslide title="Introduction" presenter="Jeff"
   audio="myAudio1.mp3" duration="5">
 <h3>Introduction</h3>
 <table>
  <tr><td>
   <ul>
    <li>Company Overview</li>
    <li>Salary by Department</li>
    <li>Employee Salary Details</li>
   </ul>
  </td></tr>
 </table>
 </cfpresentationslide>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<!--- The following code creates the second slide in the presentation.
   The chart is populated with data from the database query. --->
 <cfpresentationslide title="Salary by Department" presenter="Lori"
   duration="5" audio="myAudio3.mp3">
  <h3>Salary by Department</h3>
  <cfchart format="jpg" xaxistitle="Department" yaxistitle="Salary">
   <cfchartseries type="bar" query="GetSalaryDetails"
     itemcolumn="Dept_Name" valuecolumn="salary">
   </cfchartseries>
  </cfchart>
 </cfpresentationslide>

<!--- The following code creates the third slide in the presentation. The table is
populated with data from the query. The table also contains an image located
relative to the CFM page on the server. --->
 <cfpresentationslide title="Salary Details" presenter="Paul"
   duration="10" audio="myAudio1.mp3">
  <h3>Employee Salary Details</h3>
  <table border cellspacing=0 cellpadding=5 valign="top">
  <tr>
  <td>
  <table border cellspacing=0 cellpadding=5 valign="top">
  <tr>
   <th>Employee Name</th>
   <th>Start Date</th>
   <th>Salary</th>
   <th>Department</th>
   <th>Contract?</th>
  </tr>
  <cfoutput query="GetSalaryDetails">
  <tr>
   <td>#FirstName# #LastName#</td>
   <td>#dateFormat(StartDate, "mm/dd/yyyy")#</td>
   <td>#numberFormat(Salary, "$9999,9999")#</td>
   <td>#dept_name#</td>
   <td>#Contract#</td>
   </tr></cfoutput>
   </table>
  </td>
  <td width="200" >
   <img src="images/raquel02.jpg"/>
  </td>
 </table>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

```
 </cfpresentationslide>
</cfpresentation>
```

Example 2

The following example shows how to create a simple sales presentation with data from the cfartgallery database. Specifically, it shows how to perform the following tasks:

- Create slides generated from HTML and CFML.
- Create a slide from a URL that returns HTML content.
- Add charts with data extracted from a database and a query of queries.
- Add video and audio tracks to individual slides.

```
<!--- The following query extracts data from the cfartgallery database. --->
<cfquery name="artwork" datasource="cfartgallery">
SELECT FIRSTNAME || ' '|| LASTNAME AS FULLNAME, ARTISTS.ARTISTID, ARTNAME,
PRICE, ISSOLD
FROM ARTISTS, ART
WHERE ARTISTS.ARTISTID = ART.ARTISTID
ORDER BY LASTNAME
</cfquery>

<!--- The following query of queries determines the total dollar amount of
   sales per artist. --->
<cfquery dbtype="query" name="artistname">
SELECT FULLNAME,
SUM(PRICE) AS totalSale
FROM ARTWORK
WHERE ISSOLD = 1
GROUP BY FULLNAME
ORDER BY totalSale
</cfquery>

<!--- The following code determines the look of the slide presentation.
ColdFusion displays the slide presentation directly in the browser because no
destination is specified. The title appears above the presenter information.
--->
<cfpresentation title="Art Sales Presentation" primaryColor="##0000FF"
glowColor="##FF00FF" lightColor="##FFFF00" showOutline="yes" showNotes="yes">

<!--- The following code defines the presenter information. You can assign
each presenter to one or more slides. --->
 <cfpresenter name="Aiden" title="Artist" email="Aiden@artgallery.com"
image="../cfdocs/images/artgallery/aiden01.jpg">
 <cfpresenter name="Raquel" title="Artist" email="raquel@artgallery.com"
image="../cfdocs/images/artgallery/raquel05.jpg">
 <cfpresenter name="Paul" title="Artist" email="paul@artgallery.com"
image="../cfdocs/images/artgallery/paul01.jpg">

<!--- The following code defines the content for the first slide in the
presentation. The duration of the slide determines how long the slide plays
before proceeding to the next slide. The audio plays for the duration of the
slide. --->
 <cfpresentationslide title="Introduction" presenter="Aiden" duration="5"
audio="myAudio1.mp3">
 <h3>Introduction</h3>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

```
 <table>
  <tr><td>
   <ul>
    <li>Art Sales Overview</li>
    <li>Total Sales</li>
    <li>Total Sales by Artist</li>
    <li>Conclusion</li>
   </ul>
  </td>
  <td><img src="../cfdocs/images/artgallery/maxwell01.jpg"/></td></tr>
 </table>
 </cfpresentationslide>

<!--- The following code generates the second slide in the presentation from
an HTML file located on an external website. --->
 <cfpresentationslide title="Artwork Sales Overview" presenter="Raquel"
audio="myAudio2.mp3" duration="5" src="http://www.louvre.com/index.html"/>

<!--- The following code generates the third slide in the presentation, which
contains a pie chart with data extracted from the initial database query.
ColdFusion runs the video defined in the cfpresentationslide tag in place of
the presenter image defined in the cfpresenter tag. --->
 <cfpresentationslide title="Total Artwork Sold" presenter="Aiden"
   duration="5" video="video1.flv">
  <h3>Total Sales</h3>
  <cfchart format="jpg" chartwidth="500" show3d="yes">
   <cfchartseries type="pie" query="artwork"
     colorlist="##00FFFF,##FF00FF" itemcolumn="issold"
     valuecolumn="price"/>
   </cfchart>
  </cfpresentationslide>

<!--- The following code generates the fourth slide in the presentation with
    data extracted from the query of queries. --->
 <cfpresentationslide title="Sales by Artist" presenter="Paul"
   duration="5" audio="myAudio3.mp3">
  <h3>Total Sales by Artist</h3>
  <table border cellspacing=10 cellpadding=0>
  <TR>
  <TD>
   <table border cellspacing=0 cellpadding=5>
    <tr>
     <th>Artist Name</th>
     <th>Total Sales</th>
    </tr>
    <tr>
    <cfoutput query="artistname">
    <td>#FULLNAME#</td>
    <td>#dollarFormat(totalSale)#</td>
    </tr>
    </cfoutput>
   </table>
   </td>
   <td>
    <cfchart format="jpg" xaxistitle="Artist" yaxistitle="Total Sales"
      chartwidth="400">
     <cfchartseries type="bar" query="artistname"
       itemcolumn="fullname" valuecolumn="totalSale"/>
    </cfchart>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

```
    </td>
    </tr>
   </table>
  </cfpresentationslide>

<!--- The following code defines the final slide in the presentation. This
slide does not have a presenter assigned to it. --->
 <cfpresentationslide title="Conclusion" duration="1" notes="Special thanks to
Lori and Jeff for contributing their art and expertise.">
 <h1>Great Job Team!</h1>
 <p><img src="../cfdocs/images/artgallery/paul05.jpg"></p>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

```
 </cfpresentationslide>
</cfpresentation>
```

- Example 1
- Example 2

© 2014 Adobe Systems Incorporated. All rights reserved.

PDF Generation in ColdFusion 11

ColdFusion has introduced the **high quality PDF generation capability** from HTML using WebKit through 2 new tags, **<cfhtmltopdf>** and **<cfhtmltopdfitem>** . (**<cfdocument>** remains as another solution for PDF generation, for backward compatibility.) This document describes the usage and functionality of these new tags.

Terminology

The following terms are the commonly used in <cfhtmltopdf> (WebKit):

- **Service** - Refers to the process which carries out series of activities to convert HTML documents to PDFs.
- **Service Manager** - Manages the PDF generation service processes, monitors the service process states, spawns new processes for conversions, and facilitates recycling of processes.
- **WebKit** - Is a layout engine software designed to allow web browsers to render web pages.

The new <cfhtmltopdf> tag

<cfhtmltopdf> creates high quality PDF output from a text block containing CFML and HTML using the PDF Service Manager.

The WebKit implementation for <cfhtmltopdf> is referred as PDFG (acronym of PDF Generator) which is shipped as part of the Jetty installer. The component running inside the Jetty server is referred as the service manager, which takes request for PDF conversion from one or many ColdFusion server/s.

The WebKit implementation for <cfhtmltopdf>:

- Encapsulates the PDF conversion library.
- Performs HTML to PDF conversion in process.

See **<cfhtmltopdf>** and **<cfhtmltopdfitem>.**

Using the remote PDF service manager

You need to use a remote PDF service manager instead of the local PDF service manager due to the following reasons:

- When the ColdFusion Server is running on an OS not supported by the local PDF service manager
- When performance of the local ColdFusion Server is critical as PDF generation is a CPU intensive operation.
- When you want to distribute the load of generating the PDFs for maximum throughput.

Adding and editing the PDF service manager

From the Administrator console, click **Data & Services > PDF Service**.

You can add or edit a PDF service manager by providing the name, host name, port, and other configuration details.

© 2014 Adobe Systems Incorporated. All rights reserved.

To add or edit a remote service manager, you need to provide the details of Jetty service running on local or remote machine over which the service manager is deployed. Follow the instructions below to complete this process:

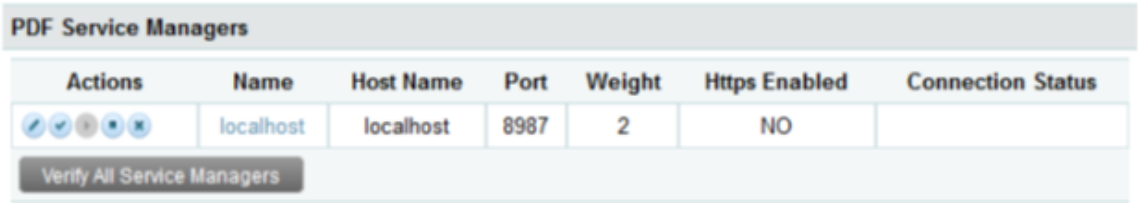
1. Enter the Name. A unique name representing the PDF service manager.
2. Enter the Host Name. This is the host name of the remote PDF service manager.
3. Enter the Port number. This is the port number of the PDF service manager. By default, the port is set to 8987.
4. Enter the **Weight** detail of the PDF service manager. Weight is the load balancing factor and determines the amount of PDF conversions that the service manager can handle. Set this based on the configuration of your system.
5. If the PDF service manager is running on HTTPS protocol, select the HTTPS Enabled check box.
6. Click **Add PDF Service Manager** button to complete the process.

	Local Service Manager: You can edit, enable, disable, and verify the service manager. Note that a local service manager cannot be added or deleted and it's host name cannot be changed.

Using the remote service manager, you can add, edit, delete, enable, disable, and verify the service manager.

Working with the PDF service manager

PDF Service Managers section lists all the available PDF service managers.



> Note: The hostname for the local PDF service manager must be one of the server's local addresses.

The description for each Action button (in the above image) is as follows:

Button	Button Description	Action

© 2014 Adobe Systems Incorporated. All rights reserved.

	Edit	Enables users to edit the settings of the existing service manager.
	Verify	Enables users to verify the current connection status of service manager.
	Delete	Enables users to delete the PDF service.

Note: Verify All Service Manager button is used to verify all the listed PDF Service Managers.

Configuring the PDF service manager

To configure the PDF service manager, use the configuration file (web.xml) located at <cf_install_root>/<CF Instance>/jetty/webapps/PDFgServlet/WEB-INF/. The available configuration settings are:

- **minService** = Minimum number of services running in process pool. Default value is 1.
- **maxService** = Maximum number of services that can run in the process pool. Default value is 2. As PDF Conversion is a CPU intense operation, it should be appropriately configured based on the number of processors in the system. Ideally, keep this value lesser than or equal to the number of CPUs in the machine.
- **idlePDFGServiceTimeout** = Idle timeout for services in the process pool. After timeout, the services are removed. Default value is 2 minutes. Creation and initialization of service is a costly operation. Also the service consumes memory for conversion. Based on the conversion frequency and load, this parameter must be tuned.
- **waitingQueueSize** = The number of PDF conversion requests which can wait in the service manager queue. Default value is 10. Once the available services are engaged in the conversion process, upcoming conversion requests are queued up. Based on the load on the service manager at the given point of time, the queue size must be configured.
- **waitingQueueTimeout**= Timeout (in seconds) for queued PDF task in the service manager queue. Default value is 60 seconds. This value needs to be tuned considering the following parameters:
    - The average PDF generation time.
    - Total number of service managers configured with the Server. If more than one service manager is configured with the Server, having a high value for waitingQTimeout may result in unnecessary wait for conversion task, even though the other service managers have idle service which can take up the task.
    - **maxPDFConversionForService** = Maximum PDF conversion allowed per service. Default value is 1000.Once service does the specified number of PDF conversions, the service will be restarted. **Note:** Based on how large or small PDF your application generates, this value needs to be tuned. For large PDFs, the memory usage of the service increases at faster rate than the smaller PDFs. Hence the maxPDFConversionForService value needs to be lower.
    - **startScanPort** = Start scanning port used by the services. Default value is 1600.
    - **endScanPort** = End scanning port used by services. Default value is 1800.
    - **enableLogging** =By default logging for PDF service manager is disabled. To enable logging, set the flag to true. Use this option only for debugging purpose.

About logging audit log details

PDF service manager logging is used to track service manager related issues and generates verbose log that contains complete details about the service and conversion request lifecycle. Use it to track the PDF conversion request.

© 2014 Adobe Systems Incorporated. All rights reserved.

Note: Logging should not be used for an extended period of time in a production environment. Extended use will produce large logs and impact performance.

The following sections describe the enhancements made to the PDF generation process.

The PDFG service logs will be available at <cf_install_root>\cfusion\jetty\webapps\PDFgServlet\.

Support for restricted DDX elements

In ColdFusion 11, the restricted DDX elements are supported for the Enterprise Edition. See this document for the list.

© 2014 Adobe Systems Incorporated. All rights reserved.

Using Web Elements and External Objects

This section helps you understand about using ColdFusion Web Services as well as integrating with JEE and Java elements; using Microsoft .Net assemblies; and integrating with COM and CORBA objects.

- Using XML and WDDX
- Using Web Services
- Using ColdFusion Web Services
- Integrating JEE and Java Elements in CFML Applications
- Using Microsoft .NET Assemblies
- Integrating COM and CORBA Objects in CFML Applications

#back to top

© 2014 Adobe Systems Incorporated. All rights reserved.

Using XML and WDDX

You can use Adobe ColdFusion to create, use, and manipulate XML documents. You can also use Web Distributed Data Exchange (WDDX), an XML dialect, for transmitting structured data, including transferring data between applications and between CFML and JavaScript.

© 2014 Adobe Systems Incorporated. All rights reserved.

About XML and ColdFusion

XML has rapidly become the universal language for representing documents and data on the web. These documents can extend beyond the traditional concept of a paper document or its equivalent. For example, XML is often used to represent database or directory information. XML is also commonly used to represent transaction information, such as product orders or receipts, and for information such as inventory records and employee data. Because XML represents data in a tagged, textual format it is an excellent tool for representing information that must be shared between otherwise-independent applications such as order entry and inventory management. No application must know anything about the other. Each application only must be prepared to get data in a format that is structured according to the XML DTD or Schema. For example, in a distributed order processing application, the order placement component, order fulfillment component, inventory management component, and billing component can all share information with each other in XML format. They could use a common XML DTD, of different components could communicate with each other using different DTDs.

After an application parses the XML document, it can then manipulate the information in any way that is appropriate. For example, you can convert tabular XML data into a ColdFusion recordset, perform queries on the data and then export the data an XML document. For example, the code in using XML in a ColdFusion application takes a customer order in XML, converts the data to a recordset, and uses a query to determine the order cost. It then prepares a receipt as an XML document.

ColdFusion provides a comprehensive and easy-to-use set of tools for creating and using XML documents. ColdFusion lets you do the following with XML documents:

- Convert XML text into ColdFusion XML document objects.
- Create new ColdFusion XML document objects.
- Modify ColdFusion XML document objects.
- Validate XML against a DTD or Schema
- Transform XML using XSLT (Extensible Stylesheet Language Transformation).
- Extract data from XML documents using XPath expressions.
- Convert ColdFusion XML document objects to text and save them in files.

  ColdFusion can also represent forms that you create using the `cfform` tag as XML. You can have ColdFusion generate the XML and process it using an XSLT skin to generate output for display, or ColdFusion can generate XML text and place it in a variable for further processing. For more information on XML Forms, see Creating Skinnable XML Forms

© 2014 Adobe Systems Incorporated. All rights reserved.

The XML document object

ColdFusion represents an XML document as an object, called an *XML document object*, that is much like a standard ColdFusion structure. In fact, most ColdFusion structure functions, such as `StructInsert`, work with XML document objects. For a full list of ColdFusion functions that work on XML document objects, see *Functions for XML object management* in Modifying a ColdFusion XML object.
You can look at the overall structure of an XML document in two ways: a basic view and a DOM (Document Object Model)-based node view. The basic view presents all the information in the document, but does not separate the data into as fine-grained units as the node view. ColdFusion can access XML document contents using either view.

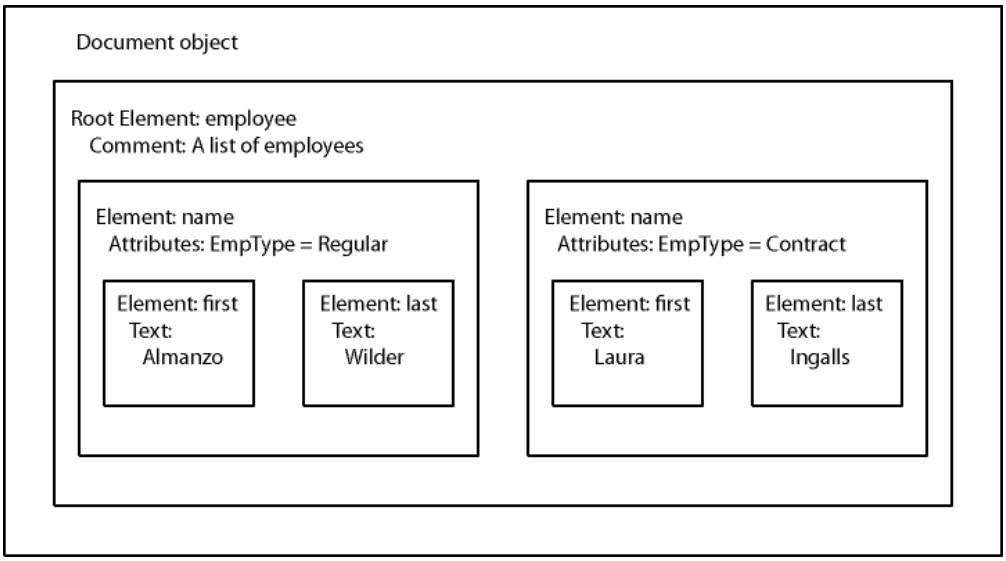
A simple XML document

The descriptions of the basic and node views use the following simple XML document. This document is used in many of the examples in the ColdFusion XML documentation.

```
<?xml version="1.0" encoding="UTF-8"?>
<employee>
<!-- A list of employees -->
 <name EmpType="Regular">
  <first>Almanzo</first>
  <last>Wilder</last>
 </name>
 <name EmpType="Contract">
  <first>Laura</first>
  <last>Ingalls</last>
 </name>
</employee>
```

Basic view

The basic view of an XML document object presents the object as a container that holds one root element structure. The root element can have any number of nested element structures. Each element structure represents an XML tag (start tag/end tag set) and all its contents; it can contain additional element structures. A basic view of the simple XML document looks like the following:



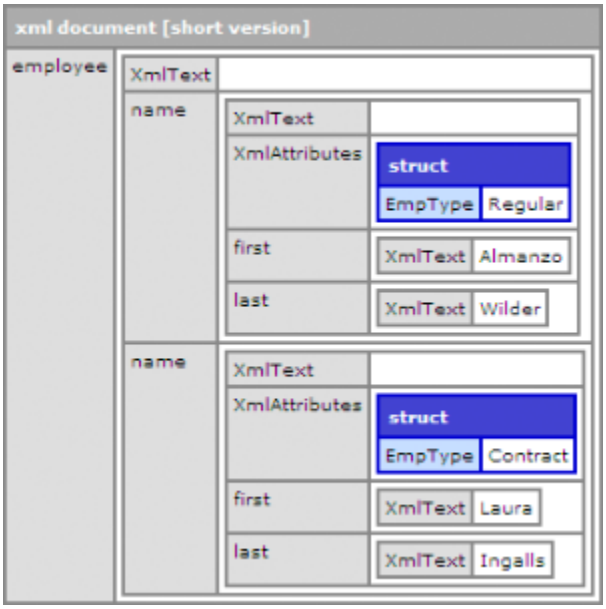
DOM node view

© 2014 Adobe Systems Incorporated. All rights reserved.

The DOM node view presents the XML document object using the same format as the document's XML *Document Object Model (DOM)*. In fact, an XML document object is a representation of a DOM object.
The DOM is a World Wide Web Consortium (W3C) recommendation (specification) for a platform- and language-neutral interface to dynamically access and update the content, structure, and style of documents. ColdFusion conforms to the DOM Level 2 Core specification, available at www.w3.org/TR/DOM-Level-2-Core.
In the DOM node view, the document consists of a hierarchical tree of nodes. Each node has a DOM node type, a node name, and a node value. Node types include Element, Comment, Text, and so on. The DOM structures the document object and each of the elements it contains into multiple nodes of different types, providing a finer-grained view of the document structure than the basic view. For example, if an XML comment is in the middle of a block of text, the DOM node view represents its position in the text while the basic view does not.
ColdFusion also lets you use the DOM objects, methods, and properties defined in the W3C DOM Level 2 Core specification to manipulate the XML document object.
For more information on referencing DOM nodes, see *XML DOM node structure* below. This document does not cover the node view and using DOM methods and properties in detail.

XML document structures

An XML document object is a structure that contains a set of nested XML element structures. The following image shows a section of the `cfdump` tag output for the document object for the XML in A simple XML document. This image shows the long version of the dump, which provides complete details about the document object. Initially, ColdFusion displays a short version, with basic information. Click the dump header to change between short, long, and collapsed versions of the dump.



The following code displays this output. It assumes that you save the code in a file under your web root, such as C:\Inetpub\wwwroot\testdocs\employeesimple.xml

```
<cffile action="read" file="C:\Inetpub\wwwroot\testdocs\employeesimple.xml"
 variable="xmldoc">
<cfset mydoc = XmlParse(xmldoc)>
<cfdump var="#mydoc#">
```

The document object structure

At the top level, the XML document object has the following three entries:

© 2014 Adobe Systems Incorporated. All rights reserved.

Entry name	Type	Description
XmlRoot	Element	The root element of the document.
XmlComment	String	A string made of the concatenation of all comments on the document, that is, comments in the document prologue and epilog. This string does not include comments inside document elements.
XmlDocType	XmlNode	The DocType attribute of the document. This entry only exists if the document specifies a DocType. This value is read only; you cannot set it after the document object has been createdThis entry does not appear when the `cfdump` tag displays an XML element structure.

The element structure

Each XML element has the following entries:

Entry name	Type	Description
XmlName	String	The name of the element; includes the namesapce prefix.
XmlNsPrefix	String	The prefix of the namespace.
XmlNsURI	String	The URI of the namespace.
XmlText orXmlCdata	String	A string made of the concatenation of all text and CData text in the element, but not inside any child elements. When you assign a value to the XmlCdata element, ColdFusion places the text inside a CDATA information item. When you retrieve information from document object, these element names return identical values.
XmlComment	String	A string made of the concatenation of all comments inside the XML element, but not inside any child elements.

© 2014 Adobe Systems Incorporated. All rights reserved.

XmlAttributes	Structure	All of this element's attributes, as name-value pairs.
XmlChildren	Array	All this element's children elements.
XmlParent	XmlNode	The parent DOM node of this element. This entry does not appear when the `cfdump` tag displays an XML element structure.
XmlNodes	Array	An array of all the XmlNode DOM nodes contained in this element.This entry does not appear the `cfdump` tag when displays an XML element structure.

XML DOM node structure

The following table lists the contents of an XML DOM node structure:

Entry name	Type	Description
XmlName	String	The node name. For nodes such as Element or Attribute, the node name is the element or attribute name.
XmlType	String	The node XML DOM type, such as Element or Text.
XmlValue	String	The node value. This entry is used only for Attribute, CDATA, Comment, and Text type nodes.

Note

The cfdump tag does not display XmlNode structures. If you try to dump an XmlNode structure, the `cfdump` tag displays "Empty Structure."

The following table lists the contents of the XmlName and XmlValue fields for each node type that is valid in the XmlType entry. The node types correspond to the objects types in the XML DOM hierarchy.

Node type	XmlName	xmlValue
`CDATA`	`#cdata-section`	Content of the CDATA section
`COMMENT`	`#comment`	Content of the comment

© 2014 Adobe Systems Incorporated. All rights reserved.

ELEMENT	Tag name	Empty string
ENTITYREF	Name of entity referenced	Empty string
PI (processing instruction)	Target entire content excluding the target	Empty string
TEXT	#text	Content of the text node
ENTITY	Entity name	Empty string
NOTATION	Notation name	Empty string
DOCUMENT	#document	Empty string
FRAGMENT	#document-fragment	Empty string
DOCTYPE	Document type name	Empty string

Note

Although XML attributes are nodes on the DOM tree, ColdFusion does not expose them as XML DOM node data structures. To view an element's attributes, use the element structure's XMLAttributes structure.

The XML document object and all its elements are exposed as DOM node structures. For example, you can use the following variable names to reference nodes in the DOM tree that you created from the XML example in A simple XML document:

```
mydoc.XmlName
mydoc.XmlValue
mydoc.XmlRoot.XmlName
mydoc.employee.XmlType
mydoc.employee.XmlNodes[1].XmlType
```

© 2014 Adobe Systems Incorporated. All rights reserved.

ColdFusion XML tag and functions

The following table lists the ColdFusion tags and functions that create and manipulate XML documents:

Tag or function	Description
`<cfxml variable="objectName"[caseSensitive="Boolean"]>`	Creates a ColdFusion XML document object consisting of the markup in the tag body. The tag can include XML and CFML tags. ColdFusion processes all CFML in the tag body before converting the resulting text to an XML document object.If you specify the `CaseSensitive="True"` attribute, the case of names of elements and attributes in the document is meaningful. The default value is `False`.For more information on using the `cfxml` tag, see *Creating an XML document object using the cfxml tag* in Creating and saving an XML document object.
`XmlParse (XMLText[[, caseSensitive],validator])`	Converts an XML document in a file or a string variable into an XML document object, and optionally validates the document against a DTD or schema. If you specify the optional second argument as `True`, the case of names of elements and attributes in the document is meaningful. The default value is `False`.For more information on using the XmlParse function, see *Creating an XML document object using the cfxml tag* in Creating and saving an XML document object.
`XmlNew([caseSensitive])`	Returns an empty XML document object. If you specify the optional argument as `True`, the case of names of elements and attributes in the document is meaningful. The default value is `False`.For more information on using the XmlNew function, see *Creating an XML document object using the XmlNew function* in *Creating an XML document object using the cfxml tag in Creating and saving an XML document object*..
`XmlElemNew(objectName {,namespaceURI],elementName)`	Returns an XML document object element with the specified name, optionally belonging to the specified namespace. You can omit the namespaceURI parameter and use only a namespace prefix if the prefix is defined elsewhere in the object.For more information on using theXmlElemNew function, see *Adding an element*.

© 2014 Adobe Systems Incorporated. All rights reserved.

`XmlTransform(XMLVar,XSLTStringVar[,parameters])`	Applies an Extensible Stylesheet Language Transformation (XSLT) to an XML document. The document can be represented as a string variable or as an XML document object. The function returns the resulting XML document as a string.For more information on using theXmlTransform function, see Transforming documents with XSLT.
`XmlSearch(objectName,XPathExpression)`	Uses an XPath expression to search an XML document object and returns an array of XML elements that match the search criteria.For more information on using the XmlSearch function, see Extracting data with XPath .
`XmlValidate(xmlDoc[,validator])`	Uses a Document Type Definition (DTD) or XML Schema to validate an XML text document (in a string or file) or an XML document object. The validator can be a DTD or Schema. If you omit the `validator` parameter, the document must specify a DTD or schema. For more information on using the XmlValidate function, see Validating XML documents
`XmlChildPos(element,elementName, position)`	Returns the position (index) in an XmlChildren array of the Nth child with the specified element name. For example, `XmlChildPos(mydoc.employee, "name", 2)` returns the position in mydoc.employee.XmlChildren of the mydoc.employee.name2 element. This index can be used in the `ArrayInsertAt` and `ArrayDeleteAt` functions. For more information on using theXmlChildPos function, see *Determining the position of a child element with a common name*, *Adding an element*, and *Deleting elements* in in Modifying a ColdFusion XML object.
`XmlGetNodeType(xmlNode)`	Returns a string identifying the type of an XML document object node returned by the function or in an element's XmlNodes array.
`IsWDDX(String)`	Determines whether a string is a well-formed WDDX packet.
`IsXML(String)`	Determines whether a string is well-formed XML text.
`IsXmlAttribute(variable)`	Determines whether the function parameter is an XML Document Object Model (DOM) attribute node.
`IsXmlDoc(objectName)`	Returns `True` if the function argument is an XML document object.

© 2014 Adobe Systems Incorporated. All rights reserved.

`IsXmlElem(elementName)`	Returns `True` if the function argument is an XML document object element.
`IsXmlNode(variable)`	Determines whether the function parameter is an XML document object node.
`IsXmlRoot(elementName)`	Returns `True` if the function argument is the root element of an XML document object.
`ToString(objectName)`	Converts an XML document object to a string representation.
`XmlFormat(string)`	Escapes special XML characters in a string so that the string can be used as text in XML.

About case sensitivity and XML document objects

The tags and functions that create XML document objects let you specify whether ColdFusion treats the object in a case-sensitive manner. If you do not specify case sensitivity, ColdFusion ignores the case of XML document object component identifiers, such as element and attribute names. If you do specify case sensitivity, names with different cases specify different components. For example, if you do not specify case sensitivity, the names mydoc.employee.name1 and mydoc.employee.NAME1 always specify the same element. If you specify case sensitivity, these names specify two separate elements. You cannot use dot notation references for element or attribute names in a case-sensitive XML document; for more information see *Referencing the contents of an XML object* in Using an XML object.

© 2014 Adobe Systems Incorporated. All rights reserved.

Using an XML object

Because an XML document object is represented as a structure, you can access XML document contents using either, or a combination of both, of the following ways:

- Using the element names, such as mydoc.employee.name1
- Using the corresponding structure entry names (that is, `XmlChildren` array entries), such as mydoc.employee.XmlChildren1
  Similarly, you can use either, or a combination of both, of the following notation methods:

- Structure (dot) notation, such as mydoc.employee
- Associative array (bracket) notation, such as mydoc"employee"

Referencing the contents of an XML object

Use the following rules when you reference the contents of an XML document object *on the right side* of an assignment or as a function argument:

- By default, ColdFusion ignores element name case. As a result, it considers the element name MyElement and the element name myELement to be equivalent. To make element name matching case-sensitive, specify `CaseSensitive="True"` in the `cfxml` tag, or specify `True` as a second argument in the `XmlParse` or `XmlNew` function that creates the document object.
- If your XML object is case sensitive, do not use dot notation to reference an element or attribute name. Use the name in associative array (bracket) notation, or a reference that does not use the case-sensitive name. For example, do not use names such as the following:

```
MyDoc.employee.name[1]

MyDoc.employee.XmlAttributes.Version
```

  Instead, use names such as the following:

```
MyDoc.xmlRoot.XmlChildren[1]
MyDoc.xmlRoot["name"][1]
MyDoc.["employee"]["name"][1]

MyDoc.xmlRoot.XmlAttributes["Version"]
MyDoc["employee"].XmlAttributes["Version"]
```

> Because ColdFusion_ always treats variable names as case-insensitive, using dot notation for element and attribute names in a case-sensitive XML document can generate unexpected results (such as all-uppercase variable names), exceptions, or both._

- If your XML object is case sensitive, you cannot use dot notation to reference an element or attribute name. Use the name in associative array (bracket) notation, or a reference that does not use the case-sensitive name (such as XmlChildren1) instead.
- Use an array index to specify one of multiple elements with the same name; for example, `#mydoc.employee.name1 and #mydoc.employee.name2`.If you omit the array index on the last component of an element identifier, ColdFusion treats the reference as the array of all elements with the specified name. For example, `mydoc.employee.name` refers to an array of two name elements.

© 2014 Adobe Systems Incorporated. All rights reserved.

- Use an array index into the `XmlChildren` array to specify an element without using its name; for example, `mydoc.XmlRoot.XmlChildren1`.
- Use associative array (bracket) notation to specify an element name that contains a period or colon; for example, `myotherdoc.XmlRoot["Type1.Case1"]`.
- You can use DOM methods in place of structure entry names.
  For example, the following variables all reference the `XmlText` value "Almanzo" in the XML document created in A simple XML document:

```
mydoc.XmlRoot.XmlChildren[1].XmlChildren[1].XmlText
mydoc.employee.name[1].first.XmlText
mydoc.employee.name[1]["first"].XmlText
mydoc["employee"].name[1]["first"].XmlText
mydoc.XmlRoot.name[1].XmlChildren[1]["XmlText"]
```

The following variables all reference the EmpType attribute of the first name element in the XML document created in A simple XML document:

```
mydoc.employee.name[1].XmlAttributes.EmpType
mydoc.employee.name[1].XmlAttributes["EmpType"]
mydoc.employee.XmlChildren[1].XmlAttributes.EmpType
mydoc.XmlRoot.name[1].XmlAttributes["EmpType"]
mydoc.XmlRoot.XmlChildren[1].XmlAttributes.EmpType
```

Neither of these lists contains a complete set of the possible combinations that can make up a reference to the value or attribute.

Assigning data to an XML object

When you use an XML object reference on the left side of an expression, most of the preceding rules apply to the reference *up to the last element* in the reference string.
For example, the rules in Referencing the contents of an XML object apply to mydoc.employee.name1.first in the following expression:

```
mydoc.employee.name[1].first.MyNewElement = XmlElemNew(mydoc, NewElement);
```

The rule for naming in case correct document objects, however, *applies to the full reference string*, as indicated by the following caution:

> Because ColdFusion_ always treats variable names as case-insensitive, using dot notation for element and attribute names in a case-sensitive XML document can generate unexpected results (such as all-uppercase variable names), exceptions, or both. In case-sensitive XML documents, use associative array notation or DOM notation names (such as XmlRoot or XmlChldren2)._

Referencing the last element on the left side of an expression

The following rules apply to the meaning of the last component on the left side of an expression:

© 2014 Adobe Systems Incorporated. All rights reserved.

1. The component name is an element structure key name (XML property name), such as `XmlComment`, ColdFusion sets the value of the specified element structure entry to the value of the right side of the expression. For example, the following line sets the XML comment in the mydoc.employee.name1.first element to "This is a comment":

```
mydoc.employee.name[1].first.XmlComment = "This is a comment";
```

2. If the component name specifies an element name and does not end with a numeric index, for example `mydoc.employee.name`, ColdFusion assigns the value on the right of the expression to the first matching element. For example, if both `mydoc.employee.name1` and `mydoc.employee.name2` exist, the following expression replaces `mydoc.employee.name1`with a new element named address, not an element named name:

```
mydoc.employee.name = XmlElemNew(mydoc, "address");
```

   After executing this line, if there had been both `mydoc.employee.name1` and `mydoc.employee.name2`, now only one `mydoc.employee.name` element exists with the contents of the original `mydoc.employee.name2`.

3. If the component name does not match an existing element, the element names on the left and right sides of the expression must match. ColdFusion creates an element with the name of the element on the left of the expression. If the element names do not match, it generates an error.For example if a, `mydoc.employee.name.phoneNumber` element does not exist, the following expression creates an `mydoc.employee.name.phoneNumber`element:

```
mydoc.employee.name.phoneNumber = XmlElemNew(mydoc, "phoneNumber");
```

   The following expression causes an error:

```
mydoc.employee.name.phoneNumber = XmlElemNew(mydoc, "address");
```

4. If the component name does not match an existing element and the component's parent or parents also do not exist, ColdFusion creates any parent nodes as specified on the left side and use the previous rule for the last element. For example, no `mydoc.employee.phoneNumber`element exists, the following expression creates a phoneNumber element containing an AreaCode element:

```
mydoc.employee.name.phoneNumber.AreaCode = XmlElemNew(mydoc, "AreaCode");
```

Assigning and retrieving CDATA values

To identify that element text is CDATA by placing it inside CDATA start and end marker information items, assign the text to the XmlCdata element, not the XmlText element. Specify CDATA because ColdFusion escapes the < and

© 2014 Adobe Systems Incorporated. All rights reserved.

> symbols in the element text when you assign it to an XmlText entry. You can assign a value to an element's XmlText entry *or* its XmlCdata entry, but not to both, as each assignment overwrites the other.
When you retrieve data from the document object, references to XmlCdata and XmlText return the same string.
The following example shows how ColdFusion handles CDATA text:

```
<cfscript>
 myCDATA = "This is CDATA text";
 MyDoc = XmlNew();
 MyDoc.xmlRoot = XmlElemNew(MyDoc,"myRoot");
 MyDoc.myRoot.XmlChildren[1] = XmlElemNew(MyDoc,"myChildNodeCDATA");
 MyDoc.myRoot.XmlChildren[1].XmlCData = "#myCDATA#";
</cfscript>

<h3>Assigning a value to MyDoc.myRoot.XmlChildren[1].XmlCdata.</h3>
<cfoutput>
 The type of element MyDoc.myRoot.XmlChildren[1] is:
#MyDoc.myRoot.XmlChildren[1].XmlType#<br>
 The value when output using XmlCdata is: #MyDoc.myRoot.XmlChildren[1].XmlCData#<br>

 The value when output using XmlText is: #MyDoc.myRoot.XmlChildren[1].XmlText#<br>
</cfoutput>
<br>
The XML text representation of Mydoc is:
<cfoutput><XMP>#tostring(MyDoc)#</XMP></cfoutput>

<h3>Assigning a value to MyDoc.myRoot.XmlChildren[1].XmlText.</h3>
<cfset MyDoc.myRoot.XmlChildren[1].XmlText = "This is XML plain text">
<cfoutput>
 The value when output using XmlCdata is: #MyDoc.myRoot.XmlChildren[1].XmlCData#<br>

 The value when output using XmlText is: #MyDoc.myRoot.XmlChildren[1].XmlText#<br>
</cfoutput>
<br>
The XML text representation of Mydoc is:
<cfoutput><XMP>#tostring(MyDoc)#</XMP></cfoutput>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

Creating and saving an XML document object

You can use several methods to create and save an XML document object. The specific technique that you use depends on the application and your coding style.

Creating an XML document object using the cfxml tag

The `cfxml` tag creates an XML document object that consists of the XML markup in the tag body. The tag body can include CFML code. ColdFusion processes the CFML code and includes the resulting output in the XML. The following example shows a simple `cfxml` tag:

```
<cfset testVar = True>
<cfxml variable="MyDoc">
 <MyDoc>
  <cfif testVar IS True>
   <cfoutput>The value of testVar is True.</cfoutput>
  <cfelse>
   <cfoutput>The value of testVar is False.</cfoutput>
  </cfif>
  <cfloop index = "LoopCount" from = "1" to = "4">
   <childNode>
    This is Child node <cfoutput>#LoopCount#.</cfoutput>
   </childNode>
  </cfloop>
 </MyDoc>
</cfxml>
<cfdump var=#MyDoc#>
```

This example creates a document object with a root element MyDoc, which includes text that displays the value of the ColdFusion variable testVar. MyDoc has four nested child elements, which are generated by an indexed cfloop tag. The `cfdump` tag displays the resulting XML document object.

> **Note**
>
> When you use the `cfxml` tag, do not include an <?xml ?> processing directive in the tag body. This directive is not required, and causes an error. To process XML text that includes the <?xml ?> directive, use the `XmlParse` function.

Creating an XML document object using the XmlNew function

The `XmlNew` function creates an XML document object, which you must then populate. For information on how to populate a new XML document, see *Adding, deleting, and modifying XML elements* in Modifying a ColdFusion XML object.

> **Note**
>
> You cannot set the `XmlDocType` property for an XML document object that you create with the `XmlNew` function.

The following example creates and displays the same ColdFusion document object as in *Creating an XML document object using the cfxml tag* above.

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<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#>
```

Creating an XML document object from existing XML

The `XmlParse` function converts an XML document or document fragment represented as text into a ColdFusion document object. You can use a string variable containing the XML or the name or URL of a file that contains the text. For example, if your application uses `cfhttp action="get"` to get the XML document, use the following line to create the XML document object:

```
<cfset myXMLDocument = XmlParse(cfhttp.fileContent)>
```

The following example converts an XML text document in a file to an XML document object:

```
<cfset myXMLDocument=XmlParse("C:\temp\myxmldoc.xml" variable="XMLFileText")>
```

The `XmlParse` function takes a second, optional, attribute that specifies whether to maintain the case of the elements and attributes in the document object. The default is to have the document object be case-insensitive. For more information on case sensitivity, see *Referencing the contents of an XML object*.
The `XmlParse` function also lets you specify a DTD or Schema to validate the XML text; if the XML is not valid, ColdFusion generates an error. You can specify the filename or URL of the validator, or the DTD or Schema can be in a CFML variable. You can also tell ColdFusion to use a DTD or Schema that is identified in the XML text. If you specify validation, also specify whether the document is case sensitive. The following example validates an XML document on file using a DTD that it specifies using a URL:

```
myDoc=XMLParse("C:\CFusion\wwwroot\examples\custorder.xml", false,
  "http://localhost:8500/examples/custorder.dtd")>
```

Saving and exporting an XML document object

The `ToString` function converts an XML document object to a text string. You can then use the string variable in any ColdFusion tag or function.

© 2014 Adobe Systems Incorporated. All rights reserved.

To save the XML document in a file, use the `ToString` function to convert the document object to a string variable, then use the `cffile` tag to save the string as a file. For example, use the following code to save the XML document myXMLDocument in the file C:\temp\myxmldoc.xml:

```
<cfset XMLText=ToString(myXMLDocument)>
<cffile action="write" file="C:\temp\myxmldoc.xml" output="#XMLText#">
```

© 2014 Adobe Systems Incorporated. All rights reserved.

Modifying a ColdFusion XML object

As with all ColdFusion structured objects, you can often use several methods to change the contents of an XML document object. For example, you often have the choice of using an assignment statement or a function to update the contents of a structure or an array. You can use array and structure functions that you can use to modify an XML document object. The section *XML document object management reference* provides a quick reference to modifying XML document object contents. Later sections describe these methods for changing document content in detail.

Functions for XML object management

The following table lists the ColdFusion array and structure functions that you can use to manage XML document objects and their functions, and describes their common uses. In several cases you can use either an array function or a structure function for a purpose, such as for deleting all of an element's attributes or children.

Function	Use
arrayLen	Determines the number of child elements in an element, that is, the number of elements in an element's `XmlChildren` array.
ArrayIsEmpty	Determines whether an element has any elements in its `XmlChildren` array.
StructCount	Determines the number of attributes in an element's `XmlAttributes` structure.
StructIsEmpty	Determines whether an element has any attributes in its `XmlAttributes` structure.Returns `True` if the specified structure, including the XML document object or an element, exists and is empty.
StructKeyArrayStructKeyList	Gets an array or list with the names of all of the attributes in an element's `XmlAttributes` structure. Returns the names of the children of an XML element.
ArrayInsertAt	Adds a new element at a specific location in an element's `XmlChildren` array.
ArrayAppendArrayPrepend	Adds a new element at the end or beginning of an element's `XmlChildren` array.
ArraySwap	Swaps the children in the `XmlChildren` array at the specified position.
ArraySet	Sets a range of entries in an `XmlChildren` array to equal the contents of a specified element structure. Each entry in the array range is a copy of the structure. Can be used to set a single element by specifying the same index as the beginning and end of the range.

© 2014 Adobe Systems Incorporated. All rights reserved.

ArrayDeleteAt	Deletes a specific element from an element's `XmlChildren` array.
ArrayClear	Deletes all child elements from an element's `XmlChildren` array.
StructDelete	Deletes a selected attribute from an element's `XMLAttributes` structure.Deletes all children with a specific element name from an element's `XmlChildren` array. Deletes all attributes of an element.Deletes all children of an element.Deletes a selected property value.
StructClear	Deletes all attributes from an element's `XMLAttributes` structure.
Duplicate	Copies an XML document object, element, or node structure.
IsArray	Returns `True` for the `XmlChildren` array. Returns false if you specify an element name, such as mydoc.XmlRoot.name, even if multiple name elements exist in XmlRoot.
IsStruct	Returns False for XML document objects, elements, and nodes. Returns True for `XmlAttributes` structures.
StructGet	Returns the specified structure, including XML document objects, elements, nodes, and `XmlAttributes` structures.
StructAppend	Appends a document fragment XML document object to another XML document object.
StructInsert	Adds a new entry to an `XmlAttributes` structure.
StructUpdate	Sets or replaces the value of a document object property such as `XmlName`, or of a specified attribute in an `XmlAttributes` structure.

Note

Array and structure functions not in the preceding or table or the table in the next section, **do not** work with XML document objects, XML elements, or XML node structures.

Treating elements with the same name as an array

In many cases, an XML element has multiple children with the same name. For example, the example document used in many XML examples has multiple name elements in the employee elements. In many cases, you can treat

© 2014 Adobe Systems Incorporated. All rights reserved.

the child elements with identical names as an array. For example, to reference the second name element in mydoc.employee, you can specify mydoc.employee.name2. However, you can only use a limited set of Array functions when you use this notation. The following table lists the array functions that are valid for such references:

Array function	Result
`IsArray(elemPath.elemName)`	Always returns False.
`ArrayClear(elemPath.elemName)`	Removes all the elements with name elemName from the elemPath element.
`ArrayLen(elemPath.elemName)`	Returns the number of elements named elemName in the elemPath element.
`ArrayDeleteAt(elemPath.elemName, n)`	Deletes the nth child named elemName from the elemPath element.
`ArrayIsEmpty(elemPath.elemName)`	Always returns `False`.
`ArrayToList(elemPath.elemName, n)`	Returns a comma-separated list of all the `XmlText` properties of all the children of elemPath named elemName.

XML document object management reference

The following tables provide a quick reference to the ways you can modify the contents of an XML document object. The sections that follow describe in detail how to modify XML contents.

> **Note**
>
> If your XML object is case sensitive, you cannot use dot notation to reference an element or attribute name. Use the name in associative array (bracket) notation, or a reference that does not use the case-sensitive name (such as xmlChildren1) instead.

Adding information to an element

Use the following techniques to add new information to an element:

Type	Using a function	Using an assignment statement

© 2014 Adobe Systems Incorporated. All rights reserved.

Attribute	`StructInsert(xmlElemPath.XmlAttributes, "key", "value")`	`xmlElemPath.XmlAttributes.key="value"` `xmlElemPath.XmlAttributes["key"]="value"`

Child element	To append:
	`ArrayAppend(xmlElempath.XmlChildren, newElem)`
	To insert:
	`ArrayInsertAt(xmlElempath.XmlChildren, position, newElem)`
		To append:
		`xmlElemPath.XmlChildren[i] =newElem`
		`xmlElemPath.newChildName =newElem`
		(where `newChildName` must be the same as `newElem.XmlName` and cannot be an indexed name such as `name[3]`)

Deleting information from an element

Use the following techniques to delete information from an element:

Type	Using a function	Using an assignment statement
Property	`StructDelete(xmlElemPath, propertyName)`	`xmlElemPath.propertyName=""`

© 2014 Adobe Systems Incorporated. All rights reserved.

Attribute	All attributes:
	`StructDelete(xmlElemPath, XmlAttributes)`
	A specific attribute:
	`StructDelete(xmlElemPath.XmlAttributes,"attributeName")`
		Not available

Child element	All children of an element:
	`StructDelete(xmlElemPath, "XmlChildren")`
	or
	`ArrayClear(xmlElemPath.XmlChildren)`
	All children with a specific name:
	`StructDelete(xmlElementpath,"elemName")`
	`ArrayClear(xmlElemPath.elemName)`
	A specific child:`ArrayDeleteAt(xmlElemPath.XmlChildren,position)`
	`ArrayDeleteAt(xmlElemPath.elemName,position)`
		Not available

Changing contents of an element

Use the following techniques to change the contents of an element:

© 2014 Adobe Systems Incorporated. All rights reserved.

Type	Using a function	Using an assignment statement
Property	`StructUpdate(xmlElemPath,"propertyName", "value")`	`xmlElemPath.propertyName ="value"` `xmlElemPath["propertyName"] ="value"`
Attribute	`StructUpdate(xmlElemPath.XmlAttributes,"attributeName", "value")`	`xmlElemPath.XmlAttributes.attributeName="value"` `xmlElemPath.XmlAttributes["attributeName"] = "value"`

© 2014 Adobe Systems Incorporated. All rights reserved.

Child element(replace)	`ArraySet(xmlElemPath.XmlChildren, index,index, newElement)`
	
	(use the same value for both index entries to change one element)
		Replace first or only child named *elementName*:
		`parentElemPath.elementName =newElement`
		`parentElemPath["elementName"]= newElement`
		
		Replace a specific child named *elementName*:
		`parentElemPath.elementName[index] = newElement`
		
		or
		`parentElemPath["elementName"][index] = newElement`

Adding, deleting, and modifying XML elements

Several basic techniques exist for adding, deleting, and modifying XML elements. The example code in the technique description uses the XML document described in A simple XML document.

Counting and finding the position of child elements

Often, an XML element has several children with the same name. For example, in the XML document defined in the simple XML document, the employee root element has multiple name elements.
To manipulate such an object, you often must know the number of children of the same name, and you could have to know the position in the `XmlChildren` array of a specific child name that is used for multiple children.

Counting child elements

The following user-defined function determines the number of child elements with a specific name in an element:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cfscript>
function NodeCount (xmlElement, nodeName)
{
 nodesFound = 0;
 for (i = 1; i LTE ArrayLen(xmlElement.XmlChildren); i = i+1)
 {
  if (xmlElement.XmlChildren[i].XmlName IS nodeName)
   nodesFound = nodesFound + 1;
 }
 return nodesFound;
}
</cfscript>
```

The following lines use this function to display the number of nodes named "name" in the mydoc.employee element:

```
<cfoutput>
Nodes Found: #NodeCount(mydoc.employee, "name")#
</cfoutput>
```

Determining the position of a child element with a common name

The `XmlChildPos` function determines the location in the `XmlChildren` array of a specific element with a common name. You use this index when ColdFusion must know where to insert or delete child elements. For example, if several name elements exist in mydoc.employee, use the following code to locate name2 in the `XmlChildren` array:

```
<cfset nameIndex = XmlChildPos(mydoc.employee, "name", 2)>
```

Adding an element

You can add an element by creating an element or by using an existing element.
Use the `XmlElemNew` function to create a new, empty element. This function has the following form:

```
XmlElemNew(docObject, elementName)
```

where *docObject* is the name of the XML document object in which you are creating the element, and *elementName* is the name you are giving the new element.
Use an assignment statement with an existing element on the right side to create an element using an existing element. See *Copying an existing element* for more information on adding elements using existing elements.

Adding an element using a function

You can use the `ArrayInsertAt` or the `ArrayAppend` function to add an element to an XML document object. For example, the following line adds a phoneNumber element after the last element for employee.name2:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cfset ArrayAppend(mydoc.employee.name[2].XmlChildren, XmlElemNew(mydoc,
   "phoneNumber"))>
```

The following line adds a new department element as the first element in employee. The name elements become the second and third elements.

```
<cfset ArrayInsertAt(mydoc.employee.XmlChildren, 1, XmlElemNew(mydoc,
   "department"))>
```

Use the format *parentElement*.`XmlChildren` to specify the array of elements to which you are adding the new element. For example, the following line causes an error:

```
<cfset ArrayInsertAt(mydoc.employee.name, 2, XmlElemNew(mydoc, "PhoneNumber")>
```

If you have multiple child elements with the same name, and you want to insert a new element in a specific position, use the function `XmlChildPos` to determine the location in the `XmlChildren` array where you want to insert the new element. For example, the following code determines the location of mydoc.employee.name1 and inserts a new name element as the second name element:

```
<cfscript>
nameIndex = XmlChildPos(mydoc.employee, "name", 1);
ArrayInsertAt(mydoc.employee.XmlChildren, nameIndex + 1, XmlElemNew(mydoc,
     "name"));
</cfscript>
```

Using a namespace: When you use a function to add an element, you can assign the element to a namespace by including the namespace prefix in the element name. If you have not yet associated the prefix with a namespace URI, also include a parameter with the namespace URI in the `XmlElemNew` function. This parameter must be the *second* parameter in the method, and the element name must be the third parameter. ColdFusion then associates the namespace prefix with the URI, and you can omit the URI parameter in further `xmlElemNew` functions.
The following example adds two to the supplies document root two elements in the Prod namespace. The first `XmlElemNew` function use sets the association between the Prod namespace prefix and the URI; the second use only requires the prefix on the element name.

```
<cfscript>
 mydoc.supplies.XmlChildren[1] = XmlElemNew(mydoc,
  "http://www.foo.com/Products", "Prod:soap");
 mydoc.supplies.XmlChildren[2] = XmlElemNew(mydoc, "Prod:shampoo");
</cfscript>
```

Adding an element using direct assignment

© 2014 Adobe Systems Incorporated. All rights reserved.

You can use direct assignment to append a new element to an array of elements. You cannot use direct assignment to insert an element into an array of elements.
When you use direct assignment, you can specify on the left side an index into the `XmlChildren` array greater than the last child in the array. For example, if two elements exist in mydoc.employee, you can specify any number greater than two, such as mydoc.employee.XmlChildren6. The element is always added as the last (in this case, third) child.
For example, the following line appends a name element to the end of the child elements of mydoc.employee:

```
<cfset mydoc.employee.XmlChildren[9] = XmlElemNew(mydoc, "name")>
```

If the parent element does not have any children with the same name as the new child, you can specify the name of the new node or the left side of the assignment. For example, the following line appends a phoneNumber element to the children of the first name element in mydoc.employee:

```
<cfset mydoc.employee.name[1].phoneNumber = XmlElemNew(mydoc, "phoneNumber")>
```

You cannot use the node name on the left to add an element with the same name as an existing element in the parent. For example, if mydoc.employee has two name nodes, the following line causes an error:

```
<cfset mydoc.employee.name[3] = XmlElemNew(mydoc, "name")>
```

However, the following line does work:

```
<cfset mydoc.employee.XmlChilren[3] = XmlElemNew(mydoc, "name")>
```

Copying an existing element

You can add a copy of an existing element elsewhere in the document. For example, if a mydoc.employee.name1.phoneNumber element exists, but no mydoc.employee. name2.phoneNumber, the following line creates an mydoc.employee. name2. phoneNumber element with the same value as the original element. This assignment copies the original element. Unlike with standard ColdFusion structures, you get a true copy, not a reference to the original structure. You can change the copy without changing the original.

```
<cfset mydoc.employee.name[2].phoneNumber = mydoc.employee.name[1].phoneNumber>
```

When you copy an element, the new element must have the same name as the existing element. If you specify the new element by name on the left side of an assignment, the element name must be the same as the name on the right side. For example, the following expression causes an error:

```
<cfset mydoc.employee.name[2].telephone = mydoc.employee.name[1].phoneNumber>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

Deleting elements

You can use many methods to delete individual or multiple elements.

Deleting individual elements

Use the `ArrayDeleteAt` function to delete a specific element from an XML document object. For example, the following line deletes the second child element in the mydoc.employee element:

```
<cfset ArrayDeleteAt(mydoc.employee.XmlChildren, 2)>
```

If an element has only one child element with a specific name, you can also use the `StructDelete` function to delete the child element. For example, the following line deletes the phoneNumber element named in the second employee.name element:

```
<cfset StructDelete(mydoc.employee.name[2], "phoneNumber")>
```

When multiple child elements have the same name, specify the element position, either among the elements of the same name, or among all child elements. For example, you can use the following line to delete the second name element in mydoc.employee:

```
<cfset ArrayDeleteAt(mydoc.employee.name, 2)>
```

You can also determine the position in the `XmlChildren` array of the element you want to delete and use that position. To do so, use the `XmlChildPos` function. For example, the following lines determine the location of mydoc.employee.name2 and delete the element:

```
<cfset idx = XmlChildPos(mydoc.employee, "name", 2)>
<cfset ArrayDeleteAt(mydoc.employee.XmlChildren, idx)>
```

Deleting multiple elements

If an element has multiple children with the same name, use the `StructDelete` function or `ArrayClear` function with an element name to delete all of an element's child elements with that name. For example, both of the following lines delete all name elements from the employee structure:

```
<cfset StructDelete(mydoc.employee, "name")>
<cfset ArrayClear(mydoc.employee.name)>
```

Use the `StructDelete` or `ArrayClear` function with `XmlChildren` to delete all of an element's child elements. For example, each of the following lines deletes all child elements of the mydoc.employee.name2 element:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cfset StructDelete(mydoc.employee.name[2], "XmlChildren")>
<cfset ArrayClear(mydoc.employee.name[2].XmlChildren)>
```

Adding, changing, and deleting element attributes

You modify an element's attributes the same way you change the contents of any structure. For example, each of the following lines adds a Status attribute the second mydoc.employee.name element:

```
<cfset mydoc.employee.name[2].XmlAttributes.Status="Inactive">
<cfset StructInsert(mydoc.employee.name[2].XmlAttributes, "Status", "Inactive")>
```

To change an attribute, use a standard assignment statement; for example:

```
<cfset mydoc.employee.name[2].XmlAttributes.Status="Active">
```

To delete an attribute, use StructDelete; for example:

```
<cfset StructDelete(mydoc.employee.name[1].XmlAttributes, "Status")>
```

Changing element properties

To change an element's properties, including its text and comment, use a standard assignment expression. For example, use the following line to add "in the MyCompany Documentation Department" to the mydoc.employee XML comment:

```
<cfset mydoc.employee.XmlComment = mydoc.employee.XmlComment & "in the
MyCompany Documentation Department">
```

Changing an element name

The XML DOM does not support changing an element name directly. To change the name of an element, create an element with the new name, insert it into the XML document object before or after the original element, copy all the original element's contents to the new element, and then delete the original element.

Clearing an element property value

To clear an element property value, either assign the empty string to the property or use the `StructDelete` functio n. For example, each of the following lines clears the comment string from mydoc.employee:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cfset mydoc.employee.XmlComment = "">
<cfset StructDelete(mydoc.employee, "XmlComment")>
```

Replacing or moving an element

To replace an element with a new element, use a standard replacement expression. For example, to replace the mydoc.employee.department element with a new element named organization, use either of the following lines:

```
<cfset mydoc.employee.department = XmlElemNew(mydoc, "Organization")>
<cfset mydoc.employee.XmlChildren[1] = XmlElemNew(mydoc, "Organization")>
```

To replace an element with a copy of an existing element, use the existing element on the right side of an expression. For example, the following line replaces the phoneNumber element for mydoc.employee.name2 with the phoneNumber element from mydoc.employee.name1:

```
<cfset mydoc.employee.name[2].phoneNumber=mydoc.employee.name[1].phoneNumber>
```

This code creates a true copy of the name1.phoneNumber element as name2.phoneNumber.
To move an element, assign it to its new location, then delete it from its old location. For example, the following lines move the phoneNumber element from mydoc.employee.name1 to mydoc.employee.name2:

```
<cfset mydoc.employee.name[2].phoneNumber=mydoc.employee.name[1].phoneNumber>
<cfset StructDelete(mydoc.employee.name[1], "phoneNumber")>
```

Note

You cannot copy or move an element from one document object to another document object.

Using XML and ColdFusion queries

You can convert XML documents into ColdFusion query objects and manipulate them using queries of queries. This technique does not require the use of XPath and provides a method of searching XML documents and extracting data that is natural to ColdFusion programmers.

Converting XML to a ColdFusion query

The following example reads an XML document, converts it to a query object, and then performs a query of queries on the object to extract selected data:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<!--- Read the file and convert it to an XML document object --->
<cffile action="read" file="C:\CFusion\wwwroot\myexamples\employees.xml"
variable="myxml">
<cfset mydoc = XmlParse(myxml)>

<!--- get an array of employees --->
<cfset emp = mydoc.employee.XmlChildren>
<cfset size = ArrayLen(emp)>

<cfoutput>
Number of employees = #size#
<br>
</cfoutput>
<br>
<!--- create a query object with the employee data --->
<cfset myquery = QueryNew("fname, lname") >
<cfset temp = QueryAddRow(myquery, #size#)>
<cfloop index="i" from = "1" to = #size#>
 <cfset temp = QuerySetCell(myquery, "fname",
  #mydoc.employee.name[i].first.XmlText#, #i#)>
 <cfset temp = QuerySetCell(myquery, "lname",
  #mydoc.employee.name[i].last.XmlText#, #i#)>
</cfloop>

<!--- Dump the query object --->
Contents of the myquery Query object: <br>
<cfdump var=#myquery#>
<br><br>

<!--- Select entries with the last name starting with A and dump the result --->
<cfquery name="ImqTest" dbType="query">
 SELECT lname, fname
 FROM myquery
 WHERE lname LIKE 'A%'
</cfquery>
<cfdump var=#imqtest#>
```

Converting a query object to XML

The following example shows how to convert a query object to XML. It uses `cfquery` to get a list of employees from the cfdocexamples database and saves the information as an XML document.

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<!--- Query the database and get the names in the employee table --->
<cfquery name="myQuery" datasource="cfdocexamples">
 SELECT FirstName, LastName
 FROM employee
</cfquery>

<!--- Create an XML document object containing the data --->
<cfxml variable="mydoc">
 <employee>
  <cfoutput query="myQuery">
    <name>
    <first>#FirstName#</first>
    <last>#LastName#</last>
   </name>
  </cfoutput>
 </employee>
</cfxml>

<!--- dump the resulting XML document object --->
<cfdump var=#mydoc#>
<!--- Write the XML to a file --->
\\x
  output=#toString(mydoc)#>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

Validating XML documents

ColdFusion provides the following methods for validating a document against a DTD or an XML Schema:

- The `XmlParse` function can validate XML text that it is parsing against a DTD or Schema. It the function encounters a validation error, ColdFusion generates an error and stops parsing the text. If the validator generates warnings, but no errors, ColdFusion parses the document and returns the result.
- The `XmlValidate` function can validate an XML text document or XML document object. against a DTD or Schema. The function returns a data structure with detailed information from the validator, including arrays of warning, error, and fatal error messages, and a Boolean status variable indicating whether the document is valid. Your application can examine the status information and determine how to handle it further.
  For examples of XML validation, see `XmlParse` and `XmlValidate` in the *CFML Reference*. The `XmlParse` example validates a document using a DTD. The `XmlValidate` example validates the document using an XML Schema that represents the same document structure as the DTD.

© 2014 Adobe Systems Incorporated. All rights reserved.

Transforming documents with XSLT

The Extensible Stylesheet Language Transformation (XSLT) technology transforms an XML document into another format or representation. For example, one common use of XSLT is to convert XML documents into HTML for display in a browser. XSLT has many other uses, including converting XML data to another format, such as converting XML in a vocabulary used by an order entry application into a vocabulary used by an order fulfillment application.
XSLT transforms an XML document by applying an Extensible Stylesheet Language (XSL) style sheet. (When stored in a file, XSL style sheets typically have the .xsl extension.) ColdFusion provides the `XmlTransform` function to apply an XSL transformation to an XML document. The function takes an XML document in string format or as an XML document object, and an XSL style sheet in string format, and returns the transformed document as a string. The following code:

1. Reads the simpletransform.xsl style sheet file into a string variable.
2. Uses the style sheet to transform the mydoc XML document object.
3. Saves the resulting transformed document in a second file.

```
<cffile action="read" file="C:\CFusion\wwwroot\testdocs\simpletransform.xsl"
 variable="xslDoc">
<cfset transformedXML = XmlTransform(mydoc, xslDoc)>
<cffile action="write" file="C:\CFusion\wwwroot\testdocs\transformeddoc.xml"
 output=transformedXML>
```

XSL and XSLT are specified by the World Wide Web Consortium (W3C). For detailed information on XSL, XSLT, and XSL style sheets, see the W3C website at www.w3.org/Style/XSL/. Several books are available on using XSL and XSLT.

© 2014 Adobe Systems Incorporated. All rights reserved.

Extracting data with XPath

XPath is a language for addressing parts of an XML document. Like XSL, XPath is a W3C specification. One of the major uses of XPath is in XSL transformations. However, XPath has more general uses. In particular, it can extract data from XML documents, such as complex data set representations. Thus, XPath is another data querying tool. XPath uses a pattern called an XPath expression to specify the information to extract from an XML document. For example, the simple XPath expression /employee/name selects the name elements in the employee root element. The `XmlSearch` function uses XPath expressions to extract data from XML document objects. The function takes an XML document object and an XPath expression in string format, and returns the results of matching the XPath expression with the XML. The returned results can be any XPath return type that ColdFusion can represent, such as an array of XML object nodes or a Boolean value. For more information, see XmlSearch in the *CFML Reference*. The following example extracts all the elements named last, which contain the employee's last names, from the employeesimple.xml file, and displays the names:

```
<cffile action="read"
 file="C:\inetpub\wwwroot\examples\employeesimple.xml"
 variable="myxml">
<cfscript>
 myxmldoc = XmlParse(myxml);
 selectedElements = XmlSearch(myxmldoc, "/employee/name/last");
 for (i = 1; i LTE ArrayLen(selectedElements); i = i + 1)
  writeoutput(selectedElements[i].XmlText & "<br>");
</cfscript>
```

XPath is specified by the World Wide Web Consortium. For detailed information on XPath, see the W3C website at www.w3.org/TR/xpath. Most books that cover XSLT also discuss XPath.

© 2014 Adobe Systems Incorporated. All rights reserved.

Example: using XML in a ColdFusion application

The following shows how you can use XML to represent data, and how ColdFusion can use XML data in an application. Although the example is too simple to be used in an application without substantial changes, it presents some of the common uses of XML with ColdFusion.
The example receives an order in the form of an XML document, processes it, and generates an XML receipt document. In this case, the order document is in a file, but it could be received as the result of an HTTP request, or retrieved using `cfpop`, `cfftp`, or other methods. The ColdFusion page does the following with the order:

1. Generates a query object from an XML document.
2. Queries a database table to determine the order discount percentage to use.
3. Uses a query of queries to calculate the total price, then calculates the discounted price.
4. Generates the receipt as an XML document.
   This example displays the results of the processing steps to show you what has been done.

The XML document

The order.xml document has the following structure:

- The root element is named order and has one attribute, id.
- One customer element exists with firstname, lastname, and accountnum attributes. The customer element does not have a body
- One items element exists that contains multiple item elements
- Each item element has an id attribute and contains a name, quantity, and unitprice element. The name, quantity, and unitprice elements contain their value as body text.
  The following order.xml document works correctly with the information in the cfdocexamples database:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<order id="4323251">
 <customer firstname="Philip" lastname="Cramer" accountNum="21"/>
 <items>
  <item id="43">
   <name>
    Large Hammer
   </name>
   <quantity>
    1
   </quantity>
   <unitprice>
    15.95
   </unitprice>
  </item>
  <item id="54">
   <name>
    Ladder
   </name>
   <quantity>
    2
   </quantity>
   <unitprice>
    40.95
   </unitprice>
  </item>
  <item id="68">
   <name>
    Paint
   </name>
   <quantity>
    10
   </quantity>
   <unitprice>
    18.95
   </unitprice>
  </item>
 </items>
</order>
```

The ColdFusion page

The ColdFusion page looks like the following:

```
<!--- Convert file to XML document object --->
<cffile action="read" file="C:\CFusion\wwwroot\examples\order.xml" variable="myxml">

<cfset mydoc = XmlParse(myxml)>

<!--- Extract account number --->
 <cfset accountNum=#mydoc.order.customer.XmlAttributes.accountNum#>
<!--- Display Order Information --->
<cfoutput>
 <b>Name=</b>#mydoc.order.customer.XmlAttributes.firstname#
```

© 2014 Adobe Systems Incorporated. All rights reserved.

```
  #mydoc.order.customer.XmlAttributes.lastname#
 <br>
 <b>Account=</b>#accountNum#
 <br>
 <cfset numItems = ArrayLen(mydoc.order.items.XmlChildren)>
 <b>Number of items ordered=</b> #numItems#
</cfoutput>
<br><br>

<!--- Process the order into a query object --->
<cfset orderquery = QueryNew("item_Id, name, qty, unitPrice") >
<cfset temp = QueryAddRow(orderquery, #numItems#)>
<cfloop index="i" from = "1" to = #numItems#>
 <cfset temp = QuerySetCell(orderquery, "item_Id",
  #mydoc.order.items.item[i].XmlAttributes.id#,#i#)>
 <cfset temp = QuerySetCell(orderquery, "name",
  #mydoc.order.items.item[i].name.XmlText#, #i#)>
 <cfset temp = QuerySetCell(orderquery, "qty",
  #mydoc.order.items.item[i].quantity.XmlText#, #i#)>
 <cfset temp = QuerySetCell(orderquery, "unitPrice",
  #mydoc.order.items.item[i].unitprice.XmlText#, #i#)>
</cfloop>

<!--- Display the order query --->
<cfdump var=#orderquery#>
<br><br>

<!--- Determine the discount --->
<cfquery name="discountQuery" datasource="cfdocexamples">
 SELECT *
 FROM employee
 WHERE Emp_Id = #accountNum#
</cfquery>
<cfset drate = 0>
<cfif #discountQuery.RecordCount# is 1>
 <cfset drate = 10>
</cfif>

<!--- Display the discount rate --->
<cfoutput>
 <b>Discount Rate =</b> #drate#%
</cfoutput>
<br><br>

<!--- Compute the total cost and discount price--->
<cfquery name="priceQuery" dbType="query">
 SELECT SUM(qty*unitPrice)
 AS totalPrice
 FROM orderquery
</cfquery>
<cfset discountPrice = priceQuery.totalPrice * (1 - drate/100)>

<!--- Display the full price and discounted price --->
<cfoutput>
 <b>Full Price=</b> #priceQuery.totalPrice#<br>
 <b>Discount Price=</b> #discountPrice#
</cfoutput>
<br><br>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<!---Generate an XML Receipt --->
<cfxml variable="receiptxml">
<receipt num = "34">
 <cfoutput>
  <price>#discountPrice#</price>
  <cfif drate GT 0 >
    <discountRate>#drate#</discountRate>
  </cfif>
 </cfoutput>
 <itemsFilled>
  <cfoutput query="orderQuery">
    <name> #name# </name>
    <qty> #qty# </qty>
    <price> #qty*unitPrice# </price>
  </cfoutput>
 </itemsFilled>
</receipt>
</cfxml>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<!--- Display the resulting receipt --->
<cfdump var=#receiptxml#>
```

Reviewing the code

The following table describes the CFML code and its function. For the sake of brevity, it does not include code that displays the processing results.

Code	Description

`<cffile action="read" file="C:\CFusion\wwwroot\examples\order.xml" variable="myxml">`
`<cfset mydoc = XmlParse(myxml)>`
`<!--- Extract account number --->`
`<cfset accountNum=#mydoc.order.customer.XmlAttributes.accountNum#>`
	Reads the XML from a file and convert it to an XML document object. Sets the accountNum variable from the customer entry's accountnum attribute.

© 2014 Adobe Systems Incorporated. All rights reserved.

<cfset orderquery = QueryNew("item_Id, name, qty, unitPrice") > <cfset temp = QueryAddRow(orderquery, #numItems#)> <cfloop index="i" from = "1" to = #numItems#> <cfset temp = QuerySetCell(orderquery, "item_Id", #mydoc.order.items.item[i].XmlAttributes.id#,#i#)> <cfset temp = QuerySetCell(orderquery, "name", #mydoc.order.items.item[i].name.XmlText#, #i#)> <cfset temp = QuerySetCell(orderquery, "qty", #mydoc.order.items.item[i].quantity.XmlText#, #i#)> <cfset temp = QuerySetCell(orderquery, "unitPrice", #mydoc.order.items.item[i].unitprice.XmlText#, #i#)> </cfloop>	Converts the XML document object into a query object. Creates a query with columns for the item_id, name, qty, and unitPrice values for each item.For each XML item entry in the mydoc.order.items entry, fills one row of the query with the item's id attribute and the text in the name, quantity, and unitprice entries that it contains.
<cfquery name="discountQuery" datasource="cfdocexamples">\ SELECT * FROM employee WHERE Emp_Id = #accountNum# </cfquery> <cfset drate = 0> <cfif #discountQuery.RecordCount# is 1> <cfset drate = 10> </cfif>	If the account number is the same as an employee ID in the cfdocexamples database Employee table, the query returns one record. and RecordCount equals 1. In this case, sets a discount rate of 10%. Otherwise, sets a discount rate of 0%.

© 2014 Adobe Systems Incorporated. All rights reserved.

`<cfquery name="priceQuery" dbType="query">`
`  SELECT SUM(qty*unitPrice)`
`  AS totalPrice`
`  FROM orderquery`
`  </cfquery>`
`  <cfset discountPrice =`
`priceQuery.totalPrice * (1 -`
`drate/100)>`
	Uses a query of queries with the SUM operator to calculate the total cost before discount of the ordered items, then applies the discount to the price. The result of the query is a single value, the total price.

`<cfxml variable="receiptxml">`
`  <receipt num = "34">`
`  <cfoutput>`
`  <price>#discountPrice#</price>`
`  <cfif drate GT 0 >`
 	
`<discountRate>#drate#</discountRat`
`e>`
`  </cfif>`
`  </cfoutput>`
`  <itemsFilled>`
`  <cfoutput query="orderQuery">`
`  <name> #name# </name>`
`  <qty> #qty# </qty>`
`  <price> #qty*unitPrice#`
`</price>`
`  </cfoutput>`
`  </itemsFilled>`
`  </receipt>`
`  </cfxml>`
	Creates an XML document object as a receipt. The receipt has a root element named receipt, which has the receipt number as an attribute. The receipt element contains a price element with the order cost and an itemsFilled element with one item element for each item.

© 2014 Adobe Systems Incorporated. All rights reserved.

Moving complex data across the web with WDDX

WDDX is an XML vocabulary for describing a complex data structure, such as an array, associative array (such as a ColdFusion structure), or a recordset, in a generic fashion. It lets you use HTTP to move the data between different application server platforms and between application servers and browsers. Target platforms for WDDX include ColdFusion, Active Server Pages (ASP), JavaScript, Perl, Java, Python, COM, Flash, and PHP.
The WDDX XML vocabulary consists of a document type definition (DTD) that describes the structure of standard data types and a set of components for each of the target platforms to do the following:

- **Serialize:** The data from its native representation into a WDDX XML document or document fragment.
- **Deserialize:** A WDDX XML document or document fragment into the native data representation, such as a CFML structure.
  This vocabulary creates a way to move data, its associated data types, and descriptors that allow the data to be manipulated on a target system, between arbitrary application servers.

> **Note**
>
> The WDDX DTD, which includes documentation, is located at www.openwddx.org/downloads/dtd/wddx_dtd_10.txt.

WDDX is a valuable tool for ColdFusion developers, however, its usefulness is not limited to CFML. If you serialize a common programming data structure (such as an array, recordset, or structure) into WDDX format, you can use HTTP to transfer the data across a range of languages and platforms. Also, you can use WDDX to store complex data in a database, file, or even a client variable.
WDDX has two features that make it useful for transferring data in a web environment:

- It is lightweight. The JavaScript used to serialize and deserialize data, including a debugging function to dump WDDX data, occupies less than 22 K.
- Unlike traditional client-server approaches, the source and target system can have minimal-to-no prior knowledge of each other. They only must know the structure of the data that is being transferred.
  WDDX was created in 1998, and many applications now expose WDDX capabilities. The best source of information about WDDX is www.openwddx.org. This site offers free downloads of the WDDX DTD and SDK and additional resources, including a WDDX FAQ, a developer forum, and links to additional sites that provide WDDX resources.

Uses of WDDX

WDDX is useful for transferring complex data between applications. For example, you can use it to exchange data between a CFML application and a CGI or PHP application. WDDX is also useful for transferring data between the server and client-side JavaScript.

Exchanging data across application servers

WDDX is useful for the transfer of complex, structured data seamlessly between different application server platforms. For example, an application based on ColdFusion at one business could `cfwddx` use to convert a purchase order structure to WDDX. It could then use `cfhttp` to send the WDDX to a supplier running a CGI-based system.
The supplier could then deserialize the WDDX to its native data form, the extract information from the order, and pass it to a shipping company running an application based on ASP.

Transferring data between the server and browser

You can use WDDX for server-to-browser and browser-to-server data exchanges. You can transfer server data to the browser in WDDX format and convert it to JavaScript objects on the browser. Similarly, your application pages can serialize JavaScript data generated on the browser into WDDX format and transfer the data to the application

© 2014 Adobe Systems Incorporated. All rights reserved.

server. You then deserialize the WDDX XML into CFML data on the server.
On the server, you use the `cfwddx` tag to serialize and deserialize WDDX data. On the browser, you use WddxSerializer and WddxRecordset JavaScript utility classes to serialize the JavaScript data to WDDX. (ColdFusion installs these utility classes on your server as webroot/CFIDE/scripts/wddx.js.)

WDDX and web services

WDDX does not compete with web services. It is a complementary technology focused on solving simple problems of application integration by sharing data on the web in a pragmatic, productive manner at low cost.
WDDX offers the following advantages:

- It can be used by lightweight clients, such as browsers or Flash Player.
- It can be used to store complex data structures in files and databases.
  Applications that take advantage of WDDX can continue to do so if they start to use web services. These applications could also be converted to use web services standards exclusively; only the service and data interchange formats: not the application model, must change.

How WDDX works

The following example shows how WDDX works. A simple structure with two string variables could have the following form after it is serialized into a WDDX XML representation:

```
<var name='x'>
 <struct>

  <var name='a'>
   <string>Property a</string>
  </var>
  <var name='b'>
   <string>Property b</string>
  </var>
 </struct>
</var>
```

When you deserialize this XML into CFML or JavaScript, the result is a structure that is created by either of the following scripts:

JavaScript	CFScript
x = new Object();x.a = "Property a";x.b = "Property b";	x = structNew();x.a = "Property a";x.b = "Property b";

Conversely, when you serialize the variable x produced by either of these scripts into WDDX, you generate the XML listed in the preceding code.
ColdFusion provides a tag and JavaScript objects that convert between CFML, WDDX, and JavaScript. Serializers and deserializers for other data formats are available on the web. For more information, see www.openwddx.org.

© 2014 Adobe Systems Incorporated. All rights reserved.

Note

The `cfwddx` tag and the wddx.js JavaScript functions use UTF-8 encoding to represent data. Any tools that deserialize ColdFusion-generated WDDX must accept UTF-8 encoded characters. UTF-8 encoding is identical to the ASCII and ISO 8859 single-byte encodings for the standard 128 "7-bit" ASCII characters. However, UTF-8 uses a two-byte representation for "high-ASCII" ISO 8859 characters where the initial bit is 1.

WDDX data type support

The following text describes the data types that WDDX supports. This information is a distillation of the description in the WDDX DTD. For more detailed information, see the DTD at www.openwddx.org.

Basic data types

WDDX can represent the following basic data types:

Data type	Description
Null	Null values in WDDX are not associated with a type such as number or string. The tag converts WDDX Nulls to empty strings.
Numbers	WDDX documents use floating-point numbers to represent all numbers. The range of numbers is restricted to /-1.7E/-308. The precision is restricted to 15 digits after the decimal point.
Date-time values	Date-time values are encoded according to the full form of ISO8601; for example, 2002-9-15T09:05:32+4:0.
Strings	Strings can be of arbitrary length and must not contain embedded nulls. Strings can be encoded using double-byte characters.

Complex data types

WDDX can represent the following complex data types:

Data type	Description
Array	Arrays are integer-indexed collections of objects of arbitrary type. Because most languages start array indexes at 0, while CFML array indexes start at 1, working with array indexes can lead to nonportable data.

© 2014 Adobe Systems Incorporated. All rights reserved.

Structure	Structures are string-indexed collections of objects of arbitrary type, sometimes called associative arrays. Because some of the languages supported by WDDX are not case-sensitive, no two variable names in a structure can differ only in their case.
Recordset	Recordsets are tabular rows of named fields, corresponding to ColdFusion query objects. Only simple data types can be stored in recordsets. Because some of the languages supported by WDDX are not case-sensitive, no two field names in a recordset can differ only in their case. Field names must satisfy the regular expression _A-Za-z_.0-9A-Za-z* where the period (.) stands for a literal period character, not "any character".
Binary	The binary data type represents strings (blobs) of binary data. The data is encoded in MIME base64 format.

Data type comparisons

The following table compares the basic WDDX data types with the data types to which they correspond in the languages and technologies commonly used on the web:

WDDX	CFML	XMLSchema	Java	ECMAScript/JavaScript	COM
null	N/A	N/A	null	null	VT_NULL
boolean	Boolean	boolean	java.lang.Boolean	boolean	VT_BOOL
number	Number	number	java.lang.Double	number	VT_R8
dateTime	DateTime	dateTime	java.lang.Date	Date	VT_DATE
string	String	string	java.lang.String	string	VT_BSTR
array	Array	N/A	java.lang.Vector	Array	VT_ARRAY | VT_VARIANT
struct	Structure	N/A	java.lang.Hashtable	Object	IWDDXStruct
recordset	Query object	N/A	coldfusion.runtime.QueryTable	WddxRecordset	IWDDXRecordset
binary	Binary	binary	byte[]	WddxBinary	V_ARRAY | UI1

© 2014 Adobe Systems Incorporated. All rights reserved.

Time zone processing

Producers and consumers of WDDX packets can be in geographically dispersed locations. Therefore, it is important to use time zone information when serializing and deserializing data, to ensure that date-time values are represented correctly.

The `cfwddx action=cfml2wddx` tag `useTimezoneInfo` attribute specifies whether to use time zone information in serializing the date-time data. In the JavaScript implementation, `useTimezoneInfo` is a property of the `WddxSerializer` object. In both cases, the default `useTimezoneInfo` value is `True`.

Date-time values in WDDX are represented using a subset of the ISO8601 format. Time zone information is represented as an hour/minute offset from universal time (UTC); for example, "2002-9-8T12:6:26-4:0".

When the `cfwddx` tag deserializes WDDX to CFML, it automatically uses available time zone information, and converts date-time values to local time. In this way, you need not worry about the details of time zone conversions. However, when the JavaScript objects supplied with ColdFusion deserialize WDDX to JavaScript expressions, they do not use time zone information, because in JavaScript it is difficult to determine the time zone of the browser.

© 2014 Adobe Systems Incorporated. All rights reserved.

Using WDDX

ColdFusion provides several tools for creating and converting WDDX that you can use for common application uses.

Using the cfwddx tag

The tag can do the following conversions:

From	To
CFML	WDDX
CFML	JavaScript
WDDX	CFML
WDDX	JavaScript

A typical `cfwddx` tag used to convert a CFML query object to WDDX looks like the following:

```
<cfwddx action="cfml2wddx" input="#MyQueryObject#" output="WddxTextVariable">
```

In this example, MyQueryObject is the name of the query object variable, and WddxTextVariable is the name of the variable in which to store the resulting WDDX XML.

> **Note**
>
> For more information on the{{cfwddx}} tag, see the CFML Reference.

Validating WDDX data

The `cfwddx` tag has a `Validate` attribute that you can use when converting WDDX to CFML or JavaScript. When you set this attribute to True, the XML parser uses the WDDX DTD to validate the WDDX data before deserializing it. If the WDDX is not valid, ColdFusion generates an error. By default, ColdFusion does not validate WDDX data before trying to convert it to ColdFusion or JavaScript data.
The `IsWDDX` function returns True if a variable is a valid WDDX data packet. It returns False otherwise. You can use this function to validate WDDX packets before converting them to another format. For example, you can use it instead of the `cfwddx` `validate` attribute, so that invalid WDDX is handled within conditional logic instead of error-handling code. You can also use it to pre-validate data that JavaScript at the browser deserializes.

Using JavaScript objects

ColdFusion provides two JavaScript objects, `WddxSerializer object` and `WddxRecordset object`, that you can use in JavaScript to convert data to WDDX. These objects are defined in the file *webroot*/cfide/scripts/wddx.js. The *CFML Reference* describes these objects and their methods in detail. The example *Transferring data from the browser to the server* shows how you can use these objects to serialize JavaScript to WDDX.

Converting CFML data to a JavaScript object

The following example demonstrates the transfer of a `cfquery` recordset from a ColdFusion page executing on the

© 2014 Adobe Systems Incorporated. All rights reserved.

server to a JavaScript object that is processed by the browser.
The application consists of four principal sections:

- Running a data query
- Including the WDDX JavaScript utility classes
- Calling the conversion function
- Writing the object data in HTML
  The following example uses the cfdocexamples data source that is installed with ColdFusion:

```
<!--- Create a simple query --->
<cfquery name = "q" datasource ="cfdocexamples">
 SELECT Message_Id, Thread_id, Username, Posted
 FROM messages
</cfquery>

<!--- Load the wddx.js file, which includes the dump function --->
<script type="text/javascript" src="/CFIDE/scripts/wddx.js"></script>

<script>
 // Use WDDX to move from CFML data to JavaScript
 <cfwddx action="cfml2js" input="#q#" topLevelVariable="qj">

 // Dump the recordset to show that all the data has reached
 // the client successfully.
 document.write(qj.dump(true));
</script>
```

Note

To see how `cfwddx Action="cfml2js"` works, save this code under your web root directory, for example in wwwroot/myapps/wddxjavascript.cfm, run the page in your browser and select View Source in your browser.

Transferring data from the browser to the server

The following example serializes form field data, posts it to the server, deserializes it, and displays the data. For simplicity, it only collects a small amount of data. In applications that generate complex JavaScript data collections, you can extend this basic approach effectively. This example uses the `WddxSerializer` JavaScript object to serialize the data, and the `cfwddx` tag to deserialize the data.

Use the example

1. Save the file under your web root directory, for example in wwwroot/myapps/ wddxserializedeserialze.cfm.
2. Display http://localhost/myapps/wddxserializedeserialze.cfm in your browser.
3. Enter a first name and last name in the form fields.
4. Click Next. The name appears in the Names added so far box.
5. Repeat steps 3 and 4 to add as many names as you wish.
6. Click Serialize to serialize the resulting data. The resulting WDDX packet appears in the WDDX packet display box. This step is intended only for test purposes. Real applications handle the serialization automatically.
7. Click Submit to submit the data.The WDDX packet is transferred to the server-side processing code, which deserializes it and displays the information.

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<!--- load the wddx.js file --->
<script type="text/javascript" src="/CFIDE/scripts/wddx.js"></script>

<!--- Data binding code --->
<script>

 // Generic serialization to a form field
 function serializeData(data, formField) {
  wddxSerializer = new WddxSerializer();
  wddxPacket = wddxSerializer.serialize(data);
  if (wddxPacket != null) {
   formField.value = wddxPacket;
  }
  else {
   alert("Couldn't serialize data");
  }
 }
 // Person info recordset with columns firstName and lastName
 // Make sure the case of field names is preserved
 var personInfo = new WddxRecordset(new Array("firstName", "lastName"), true);


 // Add next record to end of personInfo recordset
 function doNext() {
  // Extract data
  var firstName = document.personForm.firstName.value;
  var lastName = document.personForm.lastName.value;

  // Add names to recordset
  nRows = personInfo.getRowCount();
  personInfo.firstName[nRows] = firstName;
  personInfo.lastName[nRows] = lastName;

  // Clear input fields
  document.personForm.firstName.value = "";
  document.personForm.lastName.value = "";

  // Show added names on list
  // This gets a little tricky because of browser differences
  var newName = firstName + " " + lastName;
  if (navigator.appVersion.indexOf("MSIE") == -1) {
   document.personForm.names[length] =
    new Option(newName, "", false, false);
  }
  else {
   // IE version
   var entry = document.createElement("OPTION");
   entry.text = newName;
   document.personForm.names.add(entry);
  }
 }
</script>

<!--- Data collection form --->
<form action="#cgi.script_name#" method="Post" name="personForm">

 <!--- Input fields --->
 Personal information<br>
 First name: <input type=text name=firstName><br>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

```
 Last name: <input type=text name=lastName><br>
 <br>

 <!--- Navigation & submission bar --->
 <input type="button" value="Next" onclick="doNext()">
 <input type="button" value="Serialize"
 onclick="serializeData(personInfo, document.personForm.wddxPacket)">
 <input type="submit" value="Submit">
 <br><br>
 Names added so far:<br>
 <select name="names" size="5">
 </select>
 <br>

 <!--- The WDDX packet is stored here.--->
 <!--- In a real application this text area would be a hidden
   input field. --->
 <br>
 WDDX packet display:<br>
 <textarea name="wddxPacket" rows="10" cols="80" wrap="Virtual">
 </textarea>

</form>


<!--- Server-side processing --->
<hr>
<b>Server-side processing</b><br>
<br>
<cfif isdefined("form.wddxPacket")>
 <cfif form.wddxPacket neq "">

  <!--- Deserialize the WDDX data --->
  <cfwddx action="wddx2cfml" input=#form.wddxPacket#
  output="personInfo">

  <!--- Display the query --->
  The submitted personal information is:<br>
  <cfoutput query=personInfo>
   Person #CurrentRow#: #firstName# #lastName#<br>
  </cfoutput>
 <cfelse>
  The client did not send a well-formed WDDX data packet!
 </cfif>
<cfelse>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

```
 No WDDX data to process at this time.
</cfif>
```

Storing complex data in a string

The following simple example uses WDDX to store complex data, a data structure that contains arrays as a string in a client variable. It uses the `cfdump` tag to display the contents of the structure before serialization and after deserialization. It uses the `HTMLEditFormat` function in a `cfoutput` tag to display the contents of the client variable. The `HTMLEditFormat` function is required to prevent the browser from trying to interpret (and throwing away) the XML tags in the variable.

```
<!--- Enable client state management --->
<cfapplication name="relatives" clientmanagement="Yes">

<!--- Build a complex data structure --->
<cfscript>
 relatives = structNew();
 relatives.father = "Bob";
 relatives.mother = "Mary";
 relatives.sisters = arrayNew(1);
 arrayAppend(relatives.sisters, "Joan");
 relatives.brothers = arrayNew(1);
 arrayAppend(relatives.brothers, "Tom");
 arrayAppend(relatives.brothers, "Jesse");
</cfscript>

A dump of the original relatives structure:<br>
<br>
<cfdump var="#relatives#"><br>
<br>

<!--- Convert data structure to string form and save it in the client scope --->
<cfwddx action="cfml2wddx" input="#relatives#" output="Client.wddxRelatives">

The contents of the Client.wddxRelatives variable:<br>
<cfoutput>#HtmlEditFormat(Client.wddxRelatives)#</cfoutput><br>

<!--- Now read the data from client scope into a structure --->
<cfwddx action="wddx2cfml" input="#Client.wddxRelatives#" output="sameRelatives">
<br>
A dump of the sameRelatives structure generated from client.wddxRelatives<br>
<cfdump var="#sameRelatives#">
```

#back to top

© 2014 Adobe Systems Incorporated. All rights reserved.

Using Web Services

Web services let you publish and consume remote application functionality over the Internet. When you consume web services, you access remote functionality to perform an application task. When you publish a web service, you let remote users access your application functionality to build it into their own applications.

© 2014 Adobe Systems Incorporated. All rights reserved.

Web services

Since its inception, the Internet has allowed people to access content stored on remote computers. This content can be static, such as a document represented by an HTML file, or dynamic, such as content returned from a ColdFusion page or CGI script.
Web services let you access application functionality that someone created and made available on a remote computer. With a web service, you can make a request to the remote application to perform an action.
For example, you can request a stock quote, pass a text string for translation, or request information from a product catalog. The advantage of web services is that you do not have to re-create application logic that someone else has already created and, therefore, you can build your applications faster.
Referencing a remote web service within your ColdFusion application is called *consuming* web services. Since web services adhere to a standard interface regardless of implementation technology, you can consume a web service implemented as part of a ColdFusion application, or as part of a .NET or Java application.
You can also create your own web services and make them available to others for remote access, called *publishing* web service. Applications that consume your web service can be implemented in ColdFusion or by any application that recognizes the web service standard.

Accessing a web service

In its simplest form, an access to a web service is like a function call. Instead of the function call referencing a library on your computer, it references remote functionality over the Internet.
One feature of web services is that they are *self-describing*. A person who makes a web service available also publishes a description of the API to the web service as a Web Services Description Language (WSDL) file.
A WSDL file is an XML-formatted document that includes information about the web service, including the following information:

- Operations that you can call on the web service
- Input parameters that you pass to each operation
- Return values from an operation
  Consuming web services typically is a two-step process:

1. Parse the WSDL file of the web service to determine its interface. A web service makes its associated WSDL file available over the Internet. You must know the URL of the WSDL file defining the service. For example, you can access the WSDL file for the TemperatureService web service at the following URL:
   [www.xmethods.net/sd/2001/TemperatureService.wsdl]
   For an overview of WSDL syntax, see Working with WSDL files.
2. Make a request to the web service. The following example runs an operation on the Temperature web service to retrieve the temperature in ZIP code 55987:

```
<cfinvoke
  webservice="http://www.xmethods.net/sd/2001/TemperatureService.wsdl"
  method="getTemp"
  returnvariable="aTemp">
    <cfinvokeargument name="zipcode" value="55987"/>
</cfinvoke>
<cfoutput>The temperature at ZIP code 55987 is #aTemp#</cfoutput>
```

For more information on consuming web services, see Consuming web services.

Basic web service concepts

To fully understand how web services work make sure that you are familiar with the underlying architecture of a web service provider.

© 2014 Adobe Systems Incorporated. All rights reserved.

> **Note**
>
> For detailed information, consult one of the many web services books.

The following are three primary components of the web services platform:

- SOAP (Simple Object Access Protocol)
- WSDL (Web Services Description Language)
- UDDI (Universal Description, Discovery, and Integration)

Supporting web services with SOAP

SOAP provides a standard XML structure for sending and receiving web service requests and responses over the Internet. Usually you send SOAP messages using HTTP, but you also can send them using SMTP and other protocols. ColdFusion integrates the Apache Axis SOAP engine to support web services.
The ColdFusion Web Services Engine performs the underlying functionality to support web services, including generating WSDL files for web services that you create. In ColdFusion, to consume or publish web services does not require you to be familiar with SOAP or to perform any SOAP operations.
You can find additional information about SOAP in the W3C SOAP 1.1 note at www.w3.org/TR/SOAP/.

Describing web services with WSDL

A WSDL document is an XML file that describes the purpose of the web service, where it is located, and how to access it. The WSDL document describes the operations that you can invoke and their associated data types. ColdFusion can generate a WSDL document from a web service, and you can publish the WSDL document at a URL to provide information to potential clients. For more information, see Working with WSDL files.

Finding web services with UDDI

As a consumer of web services, you want to know what web services are available. As a publisher of web services, you want others to be able to find information about your web services. Universal Description, Discovery, and Integration (UDDI) provides a way for web service clients to dynamically locate web services that provide specific capabilities. You use a UDDI query to find service providers. A UDDI response contains information, such as business contact information, business category, and technical details, about how to invoke a web service. Although ColdFusion does not directly support UDDI, you can manually register or find a web service using a public UDDI registry, such as the IBM UDDI Business Registry.
You can find additional information about UDDI at www.uddi.org/about.htm.

© 2014 Adobe Systems Incorporated. All rights reserved.

Working with WSDL files

WSDL files define the interface to a web service. To consume a web service, you access the service WSDL file to determine information about it. If you publish your application logic as a web service, create a WSDL file for it. WSDL is a draft standard supported by the World Wide Web Consortium. You can access the specification at www.w3.org/TR/wsdl.

Creating a WSDL file

To publish a web service, you construct the service functionality and then create the WSDL file defining the service. In ColdFusion, you use components to create web services. ColdFusion automatically generates the WSDL file for a component that you use to produce a web service. For more information on creating web services, see Publishing web services.
For more information on components, see Building and Using ColdFusion Components.

Accessing web services using Dreamweaver

The Dreamweaver Components tab lets you view web services, including operation names, parameter names, and parameter data types.

Open the Components tab in Dreamweaver and add a web service

1. Select Window > Components, or use Control+F7, to open the Components panel.
2. In the Components panel, select Web Services from the drop-down list in the upper left of the panel.
3. Click the Plus (+) button.The Add Using WSDL dialog box appears.
4. Specify the URL of the WSDL file.
   After the web service is defined to Dreamweaver, you can drag it onto a page to call it using the `cfinvoke` tag.
   For more information on using Dreamweaver, see its online Help system.

> **Note**
>
> The Web Services option is not available if you are running Dreamweaver on the Macintosh. However, you can still use web services by writing code manually.

Reading a WSDL file

A WSDL file takes practice to read. You can view the WSDL file in a browser, or you can use a tool such as Dreamweaver, which contains a built-in utility for displaying WSDL files in an easy-to-read format.
The following example shows a WSDL file for the TemperatureService web service:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<?xml version="1.0"?>
<definitions name="TemperatureService"
targetNamespace="http://www.xmethods.net/sd/TemperatureService.wsdl"xmlns:tns="http:
//www.xmethods.net/sd/TemperatureService.wsdl"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns="http://schemas.xmlsoap.org/wsdl/">
 <message name="getTempRequest">
  <part name="zipcode" type="xsd:string"/>
 </message>
 <message name="getTempResponse">
  <part name="return" type="xsd:float"/>
 </message>
 <portType name="TemperaturePortType">
  <operation name="getTemp">
   <input message="tns:getTempRequest"/>
   <output message="tns:getTempResponse"/>
  </operation>
 </portType>
 <binding name="TemperatureBinding" type="tns:TemperaturePortType">
  <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
  <operation name="getTemp">
   <soap:operation soapAction=""/>
   <input>
    <soap:body use="encoded" namespace="urn:xmethods-Temperature"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
   </input>
   <output>
    <soap:body use="encoded" namespace="urn:xmethods-Temperature"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
   </output>
  </operation>
 </binding>
 <service name="TemperatureService">
  <documentation>Returns current temperature in a given U.S. zipcode</documentation>

  <port name="TemperaturePort" binding="tns:TemperatureBinding">
   <soap:address location="http://services.xmethods.net:80/soap/servlet/rpcrouter"/>

  </port>
 </service>
</definitions>
```

The following are the major components of the WSDL file:

Component	Definition
`definitions`	The root element of the WSDL file. This area contains namespace definitions that you use to avoid naming conflicts between multiple web services.
`types`	(Not shown) Defines data types used by the service messages.

© 2014 Adobe Systems Incorporated. All rights reserved.

`message`	Defines the data transferred by a web service operation, typically the name and data type of input parameters and return values.
`port type`	Defines one or more operations provided by the web service.
`operation`	Defines an operation that can be remotely invoked.
`input`	Specifies an input parameter to the operation using a previously defined message.
`output`	Specifies the return values from the operation using a previously defined message.
`fault`	(not shown) Optionally specifies an error message returned from the operation.
`binding`	Specifies the protocol used to access a web service including SOAP, HTTP GET and POST, and MIME.
`service`	Defines a group of related operations.
`port`	Defines an operation and its associated inputs and outputs.

For additional descriptions of the contents of this WSDL file, see Consuming web services.

© 2014 Adobe Systems Incorporated. All rights reserved.

Consuming web services

ColdFusion provides a variety of methods for consuming web services. The method that you choose depends on your ColdFusion programming style and application.
The following table describes these methods:

Method	CFML operator	Description
CFScript	()	Consumes a web service from within a CFScript block.
CFML tag		Consumes a web service from within a block of CFML code.
CFML tag		Consumes a web service from within a block of CFML code.

One important consideration is that all consumption methods use the same underlying technology and offer the same performance.

About the examples

The examples shown here reference the TemperatureService web service from XMethods. This web service returns the temperature for a given ZIP code. You can read the WSDL file for this web service in Reading a WSDL file.
The TemperatureService web service has one input parameter, a string that contains the requested ZIP code. It returns a float that contains the temperature for the specified ZIP code.

Passing parameters to a web service

The message and operation elements in the WSDL file contains subelements that define the web service operations and the input and output parameters of each operation, including the data type of each parameter. The following example shows a portion of the WSDL file for the TemperatureService web service:

```
<message name="getTempRequest">
 <part name="zipcode" type="xsd:string"/>
</message>
<message name="getTempResponse">
 <part name="return" type="xsd:float"/>
</message>
<portType name="TemperaturePortType">
 <operation name="getTemp">
  <input message="tns:getTempRequest"/>
  <output message="tns:getTempResponse"/>
 </operation>
</portType>
```

The operation name used in the examples is getTemp. This operation takes a single input parameter defined as a message of type getTempRequest.
You can see that the message element named getTempRequest contains one string parameter: `zipcode`. When you call the getTemp operation, you pass the parameter as input.

Handling return values from a web service

© 2014 Adobe Systems Incorporated. All rights reserved.

Web service operations often return information back to your application. You can determine the name and data type of returned information by examining subelements of the message and operation elements in the WSDL file. The following example shows a portion of the WSDL file for the TemperatureService web service:

```
<message name="getTempRequest">
 <part name="zipcode" type="xsd:string"/>
</message>
<message name="getTempResponse">
 <part name="return" type="xsd:float"/>
</message>
<portType name="TemperaturePortType">
 <operation name="getTemp">
  <input message="tns:getTempRequest"/>
  <output message="tns:getTempResponse"/>
 </operation>
</portType>
```

The operation getTemp returns a message of type getTempResponse. The message statement in the WSDL file defines the getTempResponse message as containing a single string parameter named `return`.

Using cfinvoke to consume a web service

With the `cfinvoke` tag, you reference the WSDL file and invoke an operation on the web service with a single tag. The `cfinvoke` tag includes attributes that specify the URL to the WSDL file, the method to invoke, the return variable, and input parameters. For complete {{}}syntax, see the *CFML Reference*.

Note

You can pass parameters to a web service using the `cfinvokeargument` tag or by specifying parameter names in the `cfinvoke` tag itself. For more information, see *Passing parameters to methods by using the cfinvoke tag* in Passing parameters to methods.

Access a web service using cfinvoke

1. Create a ColdFusion page with the following content:

```
<cfinvoke
webservice="http://www.xmethods.net/sd/2001/TemperatureService.wsdl"
method="getTemp"
returnvariable="aTemp">
<cfinvokeargument name="zipcode" value="55987"/>
</cfinvoke>
<cfoutput>The temperature at ZIP code 55987 is #aTemp#</cfoutput>
```

2. Save the page as wscfc.cfm in your web root directory.
3. View the page in your browser.
   You can omit a parameter by setting the `omit` attribute to `"yes"`. If the WSDL specifies that the argument is nullable, ColdFusion sets the associated argument to null. If the WSDL specifies minoccurs=0, ColdFusion omits the argument from the WSDL. However, CFC web services must still specify `required="true"` for all arguments.

© 2014 Adobe Systems Incorporated. All rights reserved.

You can also use an attribute collection to pass parameters. An attribute collections is a structure where each structure key corresponds to a parameter name and each structure value is the parameter value passed for the corresponding key. The following example shows an invocation of a web service using an attribute collection:

```
<cfscript>
 stArgs = structNew();
 stArgs.zipcode = "55987";
</cfscript>

<cfinvoke
webservice="http://www.xmethods.net/sd/2001/TemperatureService.wsdl"
method="getTemp"
argumentcollection="#stArgs#"
returnvariable="aTemp">
<cfoutput>The temperature at ZIP code 55987 is #aTemp#</cfoutput>
```

In this example, you create the structure in a CFScript block, but you can use any ColdFusion method to create the structure.

Using CFScript to consume a web service

The following example uses CFScript to consume a web service. In CFScript, you use the `CreateObject` function to connect to the web service. After connecting, you can make requests to the service. For `CreateObject` syntax, see the *CFML Reference*.
After creating the web service object, you can call operations of the web service using dot notation, in the following form:

```
webServiceName.operationName(inputVal1, inputVal2, ... );
```

You can handle return values from web services by writing them to a variable, as the following example shows:

```
resultVar = webServiceName.operationName(inputVal1, inputVal2, ... );
```

Or, you can pass the return values directly to a function, such as the `WriteOutput` function, as the following example shows:

```
writeoutput(webServiceName.operationName(inputVal1, inputVal2, ...) );
```

Access a web service from CFScript

1. Create a ColdFusion page with the following content:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cfscript>
ws = CreateObject("webservice",
"http://www.xmethods.net/sd/2001/TemperatureService.wsdl");
xlatstring = ws.getTemp("55987");
writeoutput(xlatstring);
</cfscript>
```

2. Save the page as wscfscript.cfm in your web root directory.
3. View the page in your browser.
   You can also use named parameters to pass information to a web service. The following example performs the same operation as the preceding code, except that it uses named parameters to make the web service request:

```
<cfscript>
ws = CreateObject("webservice",
"http://www.xmethods.net/sd/2001/TemperatureService.wsdl");
xlatstring = ws.getTemp(zipcode = "55987");
writeoutput("The temperature at 55987 is " & xlatstring);
</cfscript>
```

Consuming web services that ColdFusion does not generate

To consume a web service that is implemented in a technology other than ColdFusion, the web service must have one of the following sets of options:

- rpc as the SOAP binding style and encoding as the encodingStyle
- document as the SOAP binding style and literal as the encodingStyle
  The following example shows a portion of the WSDL file for the TemperatureService web service:

```
<binding name="TemperatureBinding" type="tns:TemperaturePortType">
 <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
 <operation name="getTemp">
  <soap:operation soapAction=""/>
  <input>
   <soap:body use="encoded" namespace="urn:xmethods-Temperature"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
  </input>
  <output>
   <soap:body use="encoded" namespace="urn:xmethods-Temperature"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
  </output>
 </operation>
</binding>
```

The WSDL file for the TemperatureService web service is compatible with ColdFusion because it uses rpc as the binding style, and encoding as the encodingStyle.

Calling web services from a Flash client

© 2014 Adobe Systems Incorporated. All rights reserved.

The Flash Remoting service lets you call ColdFusion pages from a Flash client, but it does not let you call web services directly. To call web services from a Flash client, you can use Flash Remoting to call a ColdFusion component that calls the web service. The Flash client can pass input parameters to the component, and the component can return to the Flash client any data returned by the web service.
For more information, see Using the Flash Remoting Service.

Catching errors when consuming web services

During processing, you can catch in your application errors, including SOAP faults, that otherwise propagate to the browser.
To catch errors, you specify an error type of application to the ColdFusion `cfcatch` tag, as the following example shows:

```
<cftry>
 Place your application code here ...
 <cfcatch type="application">
  <!--- Add exception processing code here ... --->
 </cfcatch>
 ...
 <cfcatch type="Any">
  <!--- Add exception processing code appropriate for all other
 exceptions here ... --->
 </cfcatch>
</cftry>
```

For more information on error handling, see Handling Errors.

Handling inout and out parameters

Some web services define inout and out parameters. You use *out* parameters to pass a placeholder for a return value to a web service. The web service then returns its result by writing it to the out parameter. *Inout* parameters let you pass a value to a web service and lets the web service return its result by overwriting the parameter value.
The following example shows a web service that takes as input an inout parameter containing a string and writes its results back to the string:

```
<cfset S="foo">
<cfscript>
 ws=createobject("webservice", "URLtoWSDL")
 ws.modifyString("S");
</cfscript>
<cfoutput>#S#</cfoutput>
```

Even though this web service takes as input the value of S, because you pass it as an inout parameter, you do not enclose it in number signs.

Note

ColdFusion supports the use of inout and out parameters to consume web services. However, ColdFusion does not support inout and out parameters when creating web services for publication.

© 2014 Adobe Systems Incorporated. All rights reserved.

Configuring web services in the ColdFusion Administrator

The ColdFusion Administrator lets you register web services so that you do not have to specify the entire WSDL URL when you reference the web service.

> **Note**
>
> The first time you reference a web service, ColdFusion automatically registers it in the Administrator.

For example, the following code references the URL to the TemperatureService WSDL file:

```
<cfscript>
ws = CreateObject("webservice",
"http://www.xmethods.net/sd/2001/TemperatureService.wsdl");
xlatstring = ws.getTemp("55987");
writeoutput(xlatstring);
</cfscript>
```

If you register the TemperatureService web service in the Administrator using (for example, the name wsTemp), you can then reference the web service as follows:

```
<cfscript>
  ws = CreateObject("webservice", "wsTemp");
  xlatstring = ws.getTemp("55987");
  writeoutput("wsTemp: " & xlatstring);
</cfscript>
```

Not only does registering the service in the Administrator enable you to shorten your code, it lets you change a web service URL without modifying your code. So, if the TemperatureService web service moves to a new location, you only update the administrator setting, not your application code.
For more information, see the ColdFusion Administrator online Help.

Data conversions between ColdFusion and WSDL data types

A WSDL file defines the input and return parameters of an operation, including data types. For example, the TemperatureService web service contains the following definition of input and return parameters:

```
<message name="getTempRequest">
 <part name="zipcode" type="xsd:string"/>
</message>
<message name="getTempResponse">
 <part name="return" type="xsd:float"/>
</message>
```

As part of consuming web services, understand how ColdFusion converts WSDL defined data types to ColdFusion data types. The following table shows this conversion:

© 2014 Adobe Systems Incorporated. All rights reserved.

ColdFusion data type	WSDL data type
numeric	SOAP-ENC:double
Boolean	SOAP-ENC:boolean
string	SOAP-ENC:string
array	SOAP-ENC:Array
binary	xsd:base64Binary
numeric	xsd:float
string	xsd:enumeration
date	xsd:dateTime
void (operation returns nothing)	
struct	complex type
query	tns1:QueryBean (Returned by CFCs)

For many of the most common data types, such as string and numeric, a WSDL data type maps directly to a ColdFusion data type. For complex WSDL data types, the mapping is not as straight forward. In many cases, you map a complex WSDL data type to a ColdFusion structure. For more information on handling complex data types, see Handling complex data types.

Consuming ColdFusion web services

Your application can consume web services created in ColdFusion. You do not have to perform any special processing on the input parameters or return values because ColdFusion handles data mappings automatically when consuming a ColdFusion web service.
For example, when ColdFusion publishes a web service that returns a query, or takes a query as an input, the WSDL file for that service lists its data type as QueryBean. However, a ColdFusion application consuming this web service can pass a ColdFusion query object to the function as an input, or write a returned QueryBean to a ColdFusion query object.

> **Note**
>
> For a list of how ColdFusion data types map to WSDL data types, see *Data conversions between ColdFusion and WSDL data types* in this page.

The following example shows a ColdFusion component that takes a query as input and echoes the query back to the caller:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cfcomponent>
 <cffunction name='echoQuery' returnType='query' access='remote'>
  <cfargument name='input' type='query'>
  <cfreturn #arguments.input#>
 </cffunction>
</cfcomponent>
```

In the WSDL file for the echotypes.cfc component, you see the following definitions that specify the type of the function input and output as QueryBean:

```
<wsdl:message name="echoQueryResponse">
  <wsdl:part name="echoQueryReturn" type="tns1:QueryBean"/>
</wsdl:message>
<wsdl:message name="echoQueryRequest">
  <wsdl:part name="input" type="tns1:QueryBean"/>
</wsdl:message>
```

For information on displaying WSDL, see Producing WSDL files.
Since ColdFusion automatically handles mappings to ColdFusion data types, you can call this web service as the following example shows:

```
<head>
<title>Passing queries to web services</title>
</head>
<body>
<cfquery name="GetEmployees" datasource="cfdocexamples">
 SELECT FirstName, LastName, Salary
 FROM Employee
</cfquery>

<cfinvoke
 webservice = "http://localhost/echotypes.cfc?wsdl"
 method = "echoQuery"
 input="#GetEmployees#"
 returnVariable = "returnedQuery">

<cfoutput>
 Is returned result a query? #isQuery(returnedQuery)# <br><br>
</cfoutput>

<cfoutput query="returnedQuery">
 #FirstName#, #LastName#, #Salary#<br>
</cfoutput>
</body>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

Publishing web services

To publish web services for consumption by remote applications, you create the web service using ColdFusion components. For more information on components, see Building and Using ColdFusion Components.

Creating components for web services

ColdFusion components (CFCs) encapsulate application functionality and provide a standard interface for client access to that functionality. A component typically contains one or more functions defined by the `cffunction` tag. For example, the following component contains a single function:

```
<cfcomponent>
 <cffunction name="echoString" returnType="string" output="no">
  <cfargument name="input" type="string">
  <cfreturn #arguments.input#>
 </cffunction>
</cfcomponent>
```

The function, named echoString, echoes back any string passed to it. To publish the function as a web service, modify the function definition to add the `access` attribute and specify remote, as the following example shows:

```
<cffunction name="echoString" returnType="string" output="no" access="remote">
```

By defining the function as remote, ColdFusion includes the function in the WSDL file. Only those functions marked as remote are accessible as a web service.
The following list defines the requirements for how to create web services for publication:

1. The value of the `access` attribute of the `cffunction` tag must be `remote`.
2. The `cffunction` tag must include the `returnType` attribute to specify a return type.
3. The `output` attribute of the `cffunction` tag must be set to `No` because ColdFusion converts all output to XML to return it to the consumer.
4. The attribute setting `required="false"` for the `cfargument` tag is ignored. ColdFusion considers all parameters as required.

Specifying data types of function arguments and return values

The `cffunction` tag lets you define a single return value and one or more input parameters passed to a function. As part of the function definition, you include the data type of the return value and input parameters.
The following example shows a component that defines a function with a return value of type string, one input parameter of type string, and one input parameter of type numeric:

```
<cfcomponent>
 <cffunction name="trimString" returnType="string" output="no">
  <cfargument name="inString" type="string">
  <cfargument name="trimLength" type="numeric">
 </cffunction>
</cfcomponent>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

As part of publishing the component for access as a web service, ColdFusion generates the WSDL file that defines the component where the WSDL file includes definitions for how ColdFusion data types map to WSDL data types. The following table shows this mapping:

ColdFusion data type	WSDL data type published
numeric	SOAP-ENC:double
Boolean	SOAP-ENC:boolean
string	SOAP-ENC:string
array	SOAP-ENC:Array
binary	xsd:base64Binary
date	xsd:dateTime
guid	SOAP-ENC:string
uuid	SOAP-ENC:string
void (operation returns nothing)	
struct	Map
query	QueryBean
any	complex type
component definition	complex type

In most cases, consumers of ColdFusion web services can easily pass data to and return results from component functions by mapping their data types to the WSDL data types shown in the preceding table.

Note

Document-literal web services use XML schema data types, not SOAP-ENC data types. For more information, see *Publishing document-literal style web services*.

For ColdFusion structures and queries, Some clients must process their data to map it to the correct type. For more information, see *Publishing web services that use complex data types*.
You can also define a data type in one ColdFusion component based on another component definition. For more information on using components to specify a data type, see *Using ColdFusion components to define data types for web services*.

Producing WSDL files

ColdFusion automatically creates a WSDL file for any component referenced as a web service. For example, if you have a component named echo.cfc in your web root directory, you can view its corresponding WSDL file by

© 2014 Adobe Systems Incorporated. All rights reserved.

requesting the component as follows:

```
http://localhost/echo.cfc?wsdl
```

The `cfcomponent` tag includes optional attributes that you can use to control the WSDL that ColdFusion generates. You can use these attributes to create meaningful WSDL attribute names, as the following example shows:

```
<cfcomponent style="document"
namespace = "http://www.mycompany.com/"
 serviceportname = "RestrictedEmpInfo"
 porttypename = "RestrictedEmpInfo"
 bindingname = "myns:RestrictedEmpInfo"
 displayname = "RestrictedEmpInfo"
 hint = "RestrictedEmpInfo">
```

For complete control of the WSDL, advanced users can specify the `cfcomponent` `wsdlFile_` attribute to use a predefined WSDL file._

The following example defines a ColdFusion component that can be invoked as a web service:

```
<cfcomponent>
 <cffunction
 name = "echoString"
 returnType = "string"
 output = "no"
 access = "remote">
  <cfargument name = "input" type = "string">
  <cfreturn #arguments.input#>
 </cffunction>
</cfcomponent>
```

If you register the component in Dreamweaver, it appears in the Components tab of the Application panel.

Requesting the WSDL file in a browser returns the following:

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions targetNamespace="http://ws"
xmlns="http://schemas.xmlsoap.org/wsdl/"
xmlns:apachesoap="http://xml.apache.org/xml-soap"
xmlns:impl="http://ws"
xmlns:intf="http://ws"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:tns1="http://rpc.xml.coldfusion"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
```

© 2014 Adobe Systems Incorporated. All rights reserved.

```
xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<!--WSDL created by ColdFusion -->
<wsdl:types>
<schema targetNamespace="http://rpc.xml.coldfusion"
xmlns="http://www.w3.org/2001/XMLSchema">
<import namespace="http://schemas.xmlsoap.org/soap/encoding/"/>
<complexType name="CFCInvocationException">
<sequence/>
</complexType>
</schema>
</wsdl:types>
<wsdl:message name="CFCInvocationException">
<wsdl:part name="fault" type="tns1:CFCInvocationException"/>
</wsdl:message>
<wsdl:message name="echoStringResponse">
<wsdl:part name="echoStringReturn" type="xsd:string"/>
</wsdl:message>
<wsdl:message name="echoStringRequest">
<wsdl:part name="input" type="xsd:string"/>
 </wsdl:message>
 <wsdl:portType name="echo">
 <wsdl:operation name="echoString" parameterOrder="input">
 <wsdl:input message="impl:echoStringRequest" name="echoStringRequest"/>
 <wsdl:output message="impl:echoStringResponse"
    name="echoStringResponse"/>
 <wsdl:fault message="impl:CFCInvocationException" name="CFCInvocationException"/>
 </wsdl:operation>
 </wsdl:portType>
 <wsdl:binding name="echo.cfcSoapBinding" type="impl:echo">
 <wsdlsoap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/
   http"/>
 <wsdl:operation name="echoString">
 <wsdlsoap:operation soapAction=""/>
 <wsdl:input name="echoStringRequest">
 <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/
     encoding/" namespace="http://ws" use="encoded"/>
 </wsdl:input>
 <wsdl:output name="echoStringResponse">
 <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/
     encoding/" namespace="http://ws" use="encoded"/>
 </wsdl:output>
 <wsdl:fault name="CFCInvocationException">
 <wsdlsoap:fault encodingStyle="http://schemas.xmlsoap.org/soap/
     encoding/" name="CFCInvocationException" namespace=
     "http://ws" use="encoded"/>
 </wsdl:fault>
 </wsdl:operation>
 </wsdl:binding>
 <wsdl:service name="echoService">
 <wsdl:port binding="impl:echo.cfcSoapBinding" name="echo.cfc">
 <wsdlsoap:address location="http://localhost:8500/ws/echo.cfc"/>
 </wsdl:port>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

```
  </wsdl:service>
 </wsdl:definitions>
```

Publish a web service

1. Create a ColdFusion page with the following content:

```
<cfcomponent output="false">
 <cffunction
    name = "echoString"
    returnType = "string"
    output = "no"
    access = "remote">
  <cfargument name = "input" type = "string">
  <cfreturn #arguments.input#>
 </cffunction>
</cfcomponent>
```

2. Save this file as echo.cfc in your web root directory.
3. Create a ColdFusion page with the following content:

```
<cfinvoke webservice ="http://localhost/echo.cfc?wsdl"
 method ="echoString"
 input = "hello"
 returnVariable="foo">

<cfoutput>#foo#</cfoutput>
```

4. Save this file as echoclient.cfm in your web root directory.
5. Request echoclient.cfm in your browser. The following string appears in your browser:

```
hello
```

You can also invoke the web service using the following code:

```
<cfscript>
 ws = CreateObject("webservice", "http://localhost/echo.cfc?wsdl");
 wsresults = ws.echoString("hello");
 writeoutput(wsresults);
</cfscript>
```

Using ColdFusion components to define data types for web services

ColdFusion lets you define components that contain only properties. Once defined, you can use components to define data types for web services. The following code defines a component in the file address.cfc that contains properties that represent a street address:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cfcomponent>
 <cfproperty name="AddrNumber" type="numeric">
 <cfproperty name="Street" type="string">
 <cfproperty name="City" type="string">
 <cfproperty name="State" type="string">
 <cfproperty name="Country" type="string">
</cfcomponent>
```

The following code defines a component in the filename.cfc that defines first and last name properties:

```
<cfcomponent>
 <cfproperty name="Firstname" type="string">
 <cfproperty name="Lastname" type="string">
</cfcomponent>
```

You can then use address and name to define data types in a ColdFusion component created to publish a web service, as the following example shows:

```
<cfcomponent>
 <cffunction
     name="echoName" returnType="name" access="remote" output="false">
   <cfargument name="input" type="name">
   <cfreturn #arguments.input#>
 </cffunction>

 <cffunction
     name="echoAddress" returnType="address" access="remote" output="false">
   <cfargument name="input" type="address">
   <cfreturn #arguments.input#>
 </cffunction>
</cfcomponent>
```

Note

If the component files are not in a directory under your web root, create a web server mapping to the directory that contains them. You cannot use ColdFusion mappings to access web services.

The WSDL file for the web service contains data definitions for the complex types name and address. Each definition consists of the elements that define the type as specified in the ColdFusion component file for that type. For example, the following example shows the definition for name:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<complexType name="name">
  <sequence>
    <element name="firstname" nillable="true" type="soapenc:string"/>
    <element name="lastname" nillable="true" type="soapenc:string"/>
  </sequence>
</complexType>
```

You can also specify an array of CFCs in the `returnType` attribute, as the following example shows:

```
<cfcomponent>
 <cffunction
 name="allNames" returnType="name[]" access="remote" output="false">
 <cfset var returnarray = ArrayNew(1)>
 <cfset var temp = "">
 <cfquery name="empinfo" datasource="cfdocexamples">
  SELECT firstname, lastname
  FROM employee
 </cfquery>
 <cfloop query="empinfo" >
       <cfobject component="name" name="tempname">
       <cfset tempname.Firstname = #empinfo.firstname#>
       <cfset tempname.Lastname = #empinfo.lastname#>
       <cfset temp = ArrayAppend(returnarray, tempname)>
     </cfloop>
     <cfreturn returnarray>
 </cffunction>
</cfcomponent>
```

When you invoke the web service, it returns an array of CFCs. Access the properties in the CFC by using dot notation, as the following example shows:

```
<cfinvoke webservice ="http://localhost:8500/ws/cfcarray.cfc?wsdl"
 method ="allNames"
 returnVariable="thearray">

<cfif IsArray(thearray)>
<h1>loop through the employees</h1>
<p>thearray has <cfoutput>#ArrayLen(thearray)#</cfoutput> elements.</p>
<cfloop index="i" from="1" to="#ArrayLen(thearray)#">
 <cfoutput>#thearray[i].firstname#, #thearray[i].lastname# </cfoutput><br>
</cfloop>
<cfelse>
  <h1>Error: thearray is not an array</h1>
</cfif>
```

Publishing document-literal style web services

In addition to RPC-oriented operations, for which consumers specify a method and arguments, ColdFusion also lets you publish web services using the document-literal style. When you use document-literal style, the WSDL for the

© 2014 Adobe Systems Incorporated. All rights reserved.

web service tells the client to use XML schemas rather than RPC calling conventions.
In most cases, the publisher of a web services identifies it as document-literal or RPC style. To identify the type, open the WSDL document and find the `soap:binding` element and examine its `style` attribute, as the following example shows:

```
<wsdl:binding name="WeatherForecastSoap" type="tns:WeatherForecastSoap">
<soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document" />
```

In this example, the style is document-literal. Examine the WSDL to determine the methods you can call and the parameters for each method.
On the client side, the `cfinvoke` tag and other ColdFusion methods for calling web services handle the style automatically. In most cases, no modifications are necessary. Similarly, when publishing CFCs as document-literal style web services, ColdFusion automatically creates and manages the appropriate WSDL.
To publish CFCs as document-literal style web services, specify `cfcomponent style="document"`, along with the other attributes required for document-literal style web services. For example, ColdFusion publishes the following CFC using document-literal style:

```
<cfcomponent style="document" >
 <cffunction
name = "getEmp"
returntype="string"
output = "no"
access = "remote">
 <cfargument name="empid" required="yes" type="numeric">
 <cfset var fullname = "">
 <cfquery name="empinfo" datasource="cfdocexamples">
  SELECT emp_id, firstname, lastname
  FROM employee
  WHERE emp_id = <cfqueryparam cfsqltype="cf_sql_integer"
   value="#arguments.empid#">
 </cfquery>
 <cfif empinfo.recordcount gt 0>
  <cfset fullname = empinfo.lastname & ", " & empinfo.firstname>
 <cfelse>
 <cfset fullname = "not found">
 </cfif>
<cfreturn #fullname#>
</cffunction>
</cfcomponent>
```

Securing your web services

You can restrict access to your published web services to control the users allowed to invoke them. You can use your web server to control access to the directories containing your web services, or you can use ColdFusion security in the same way that you would to control access to any ColdFusion page.

To browse the HTML description of a CFC file, you request the file by specifying a URL to the file in your browser. By default, ColdFusion secures access to all URLs that directly reference a CFC file, and prompts you to enter a password upon the request. Use the ColdFusion RDS password to view the file.
To disable security on CFC file browsing, use the ColdFusion Administrator to disable the RDS password.
For more information, see Building and Using ColdFusion Components.

© 2014 Adobe Systems Incorporated. All rights reserved.

Using your web server to control access

Most web servers, including IIS and Apache, implement directory access protection using the basic HTTP authentication mechanism. When a client attempts to access one of the resources under a protected directory, and has not properly authenticated, the web server automatically sends back an authentication challenge, typically an HTTP Error 401 Access Denied error.

In response, the client browser opens a login prompt containing a user name and password field. When the user submits this information, the browser sends it back to the web server. If authentication passes, the web server allows access to the directory. The browser also caches the authentication data as long as it is open, so subsequent requests automatically include the authentication data.

Web service clients can also pass the user name and password information as part of the request. The `cfinvoke` tag includes the `user name` and `password` attributes that let you pass login information to a web server using HTTP basic authentication. You can include these attributes when invoking a web service, as the following example shows:

```
<cfinvoke
 webservice = "http://some.cfc?wsdl"
 returnVariable = "foo"
 ...
 username="aName"
 password="aPassword">
<cfoutput>#foo#</cfoutput>
```

ColdFusion inserts the user name/password string in the `authorization` request header as a base64 binary encoded string, with a colon separating the user name and password. This method of passing the user name/password is compatible with the HTTP basic authentication mechanism used by web servers.

The ColdFusion Administrator lets you predefine web services. As part of defining the web service, you can specify the user name and password that ColdFusion includes as part of the request to the web service. Therefore, you do not have to encode this information using the `cfinvoke` tag. For information on defining a web service in the ColdFusion Administrator, see *Configuring web services in the ColdFusion Administrator* in Consuming web services.

Using ColdFusion to control access

Instead of letting the web server control access to your web services, you can handle the user name/password string in your Application.cfc or Application.cfm file as part of your own security mechanism. In this case, you use the `cflogin` tag to retrieve the user name/password information from the `authorization` header, decode the binary string, and extract the user name and password, as the following excerpt from an Application.cfc `onRequestStart` method shows:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cflogin>
 <cfset isAuthorized = false>

 <cfif isDefined("cflogin")
  <!--- Verify user name from cflogin.name and password from
 cflogin.password using your authentication mechanism. --->
  >
  <cfset isAuthorized = true>
 </cfif>
</cflogin>

<cfif not isAuthorized>
 <!--- If the user does not pass a user name/password, return a 401 error.
  The browser then prompts the user for a user name/password. --->
 <cfheader statuscode="401">
 <cfheader name="WWW-Authenticate" value="Basic realm=""Test""">
 <cfabort>
</cfif>
```

This example does not show how to perform user verification. For more information on verification, see Securing Applications.

Best practices for publishing web services

ColdFusion web services provide a powerful mechanism for publishing and consuming application functionality. However, before you produce web services for publication, consider the following best practices:

1. Minimize the use of ColdFusion complex types, such as query and struct, in the web services you create for publication. These types require consumers, especially ones that consume the web service using a technology other than ColdFusion, to create special data structures to handle complex types.
2. Locally test the ColdFusion components implemented for web services before publishing them over the Internet.

© 2014 Adobe Systems Incorporated. All rights reserved.

Using request and response headers

ColdFusion includes a set of functions that enable your web service to get and set request and response headers. You use these functions to retrieve the response headers from a web service request and to create SOAP headers in a request that has the `mustUnderstand` attribute set to be True.
You typically use different functions in web services clients and in the web service CFC, itself:
In the client:

- AddSOAPRequestHeader, called before the request to set a SOAP header.
- GetSOAPResponseHeader, called after the request to retrieve a SOAP header.
- GetSOAPResponse, called after the request to retrieve a SOAP response.

In the web service CFC:

- IsSOAPRequest, called to determine whether the CFC method is being called as a web service.
- GetSOAPRequestHeader, called to retrieve a SOAP header set by the client.
- GetSOAPRequest, called to retrieve a SOAP request sent by the client.
- AddSOAPResponseHeader, called to set a SOAP header that is returned to the client.

> **Note**
>
> When used in a CFC, you can only use these functions in CFC methods if they are being used as web services. Use the `IsSOAPRequest` function to determine whether the CFC method is being called as a web service.

The following example CFM page uses the `AddSOAPRequestHeader`, `getSOAPRequest`, and `GetSOAPRespons` e functions:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cfsavecontent variable="my_xml">
<Request xmlns="http://www.oasis-open.org/asap/0.9/asap.xsd">
<SenderKey>ss</SenderKey>
<ReceiverKey>zz</ReceiverKey>
<ResponseRequired>Yes</ResponseRequired>
<RequestID>id</RequestID>
</Request>
</cfsavecontent>
<cfset xml_obj = xmlparse(my_xml)>

<cfscript>
ws = CreateObject("webservice",
"http://localhost:8500/soapexamples/HeaderFuncs.cfc?WSDL");
addSOAPRequestHeader(ws, "http://www.cfdevguide.com/", "testrequestheader",
"#xml_obj#");
</cfscript>

<cfscript>
ret=ws.showSOAPHeaders();
inxml = getSOAPRequest(ws);
outxml = getSOAPResponse(ws);
</cfscript>

<cfoutput>
<h2>Return Value</h2>
<!--- This code is XML, so use HTMLCodeFormat. --->
The return value was #ret#
<h2>Complete Request XML</h2>
#htmlcodeformat(inxml)#
<h2>Complete Response XML</h2>
#htmlcodeformat(outxml)#
</cfoutput>
```

The following example CFC uses the `IsSOAPRequest` and `AddSOAPResponseHeader` functions:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cfcomponent>
<cffunction
 name = "showSOAPHeaders"
 returnType = "string"
 output = "no"
 access = "remote"
  hint="After calling this function, use GetSOAPRequest and GetSOAPResponse to view
headers">
 <cfset var xml_obj = "">
 <cfset var ret = "">

<cfif IsSOAPRequest()>
<!--- Define a response header --->
<cfsavecontent variable="response_xml">
 <ThisResponseHeader xmlns="http://www.cfdevguide.com">
 <CreatedDateTime><cfoutput>#now()#</cfoutput></CreatedDateTime>
 <ExpiresInterval>6000</ExpiresInterval>
 </ThisResponseHeader>
</cfsavecontent>
<cfset xml_obj = xmlparse(response_xml)>
<!--- Add the response header --->
<cfscript>
addSOAPResponseHeader("http://www.cfdevguide.com/", "testresponseheader",
"#xml_obj#");
ret = "Invoked as a web service. Use GetSOAPRequest and GetSOAPResponse to view
headers.";
 </cfscript>
<cfelse>
<cfset ret = "Not invoked as a web service">
</cfif>
<cfreturn ret>
</cffunction>
</cfcomponent>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

Handling complex data types

When dealing with web services, handling complex types falls into the following categories:

- Mapping the data types of a web service to consume to ColdFusion data types
- Understanding how clients reference your ColdFusion data types when you publish a web service

Consuming web services that use complex data types

The following table shows how WSDL data types are converted to ColdFusion data types:

ColdFusion data type	WSDL data type
numeric	SOAP-ENC:double
Boolean	SOAP-ENC:boolean
string	SOAP-ENC:string
array	SOAP-ENC:Array
numeric	SOAP-ENC:float
binary	xsd:base64Binary
date	xsd:dateTime
void (operation returns nothing)	
structure	complex type

This table shows that complex data types map to ColdFusion structures. ColdFusion structures offer a flexible way to represent data. You can create structures that contain single-dimension arrays, multi-dimensional arrays, and other structures.
The ColdFusion mapping of complex types to structures is not automatic. Accessing the data as a structure requires some processing of the data. The next sections describe how to pass complex types to web services, and how to handle complex types returned from web services.

Passing input parameters to web services as complex types

A web service can take a complex data type as input. In this situation, you can construct a ColdFusion structure that models the complex data type, then pass the structure to the web service.
For example, the following excerpt from a WSDL file shows the definition of a complex type named Employee:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<s:complexType name="Employee">
 <s:sequence>
   <s:element minOccurs="1" maxOccurs="1" name="fname" type="s:string" />
   <s:element minOccurs="1" maxOccurs="1" name="lname" type="s:string" />
   <s:element minOccurs="1" maxOccurs="1" name="active" type="s:boolean" />
   <s:element minOccurs="1" maxOccurs="1" name="age" type="s:int" />
   <s:element minOccurs="1" maxOccurs="1" name="hiredate" type="s:dateTime" />
   <s:element minOccurs="1" maxOccurs="1" name="number" type="s:double" />
 </s:sequence>
</s:complexType>
```

The Employee data type definition includes six elements, the data type of each element, and the name of each element.
Another excerpt from the WSDL file shows a message definition using the Employee data type. This message defines an input parameter, as the following code shows:

```
<message name="updateEmployeeInfoSoapIn">
 <part name="thestruct" type="s0:Employee" />
</message>
```

A third excerpt from the WSDL file shows the definition of an operation, named updateEmployeeInfo, possibly one that updates the employee database with the employee information. This operation takes as input a parameter of type Employee, as the following code shows:

```
<operation name="updateEmployeeInfo">
 <input message="s0:updateEmployeeInfoSoapIn" />
</operation>
```

To call the updateEmployeeInfo operation, create a ColdFusion structure, initialize six fields of the structure that correspond to the six elements of Employee, and then call the operation, as the following code shows:

```
<!--- Create a structure using CFScript, then call the web service. --->
<cfscript>
 stUser = structNew();
 stUser.active = TRUE;
 stUser.fname = "John";
 stUser.lname = "Smith";
 stUser.age = 23;
 stUser.hiredate = createDate(2002,02,22);
 stUser.number = 123.321;

 ws = createObject("webservice", "http://somehost/EmployeeInfo.asmx?wsdl");
 ws.updateEmployeeInfo(stUser);
</cfscript>
```

You can use structures for passing input parameters as complex types in many situations. However, to build a

© 2014 Adobe Systems Incorporated. All rights reserved.

structure to model a complex type, inspect the WSDL file for the web service to determine the layout of the complex type. This process can take some practice.

Handling return values as complex types

When a web service returns a complex type, you can write that returned value directly to a ColdFusion variable. The previous section used a complex data type named Employee to define an input parameter to an operation. A WSDL file can also define a return value using the Employee type, as the following code shows:

```
<message name="updateEmployeeInfoSoapOut">
 <part name="updateEmployeeInfoResult" type="s0:Employee" />
</message>
<operation name="updateEmployeeInfo">
 <input message="s0:updateEmployeeInfoSoapIn" />
 <output message="s0:updateEmployeeInfoSoapOut" />
</operation>
```

In this example, the operation updateEmployeeInfo takes a complex type as input and returns a complex type as output. To handle the input parameter, you create a structure. To handle the returned value, you write it to a ColdFusion variable, as the following example shows:

```
<!--- Create a structure using CFScript, then call the web service. --->
<!--- Write the returned value to a ColdFusion variable. --->
<cfscript>
 stUser = structNew();
 stUser.active = TRUE;
 stUser.fname = "John";
 stUser.lname = "Smith";
 stUser.age = 23;
 stUser.hiredate = createDate(2002,02,22);
 stUser.number = 123.321;

 ws = createObject("webservice", "http://somehost/echosimple.asmx?wsdl");
 myReturnVar = ws.echoStruct(stUser);

</cfscript>

<!--- Output the returned values. --->
<cfoutput>
 <br>
 <br>Name of employee is: #myReturnVar.fname##myReturnVar.lname#
 <br>Active status: #myReturnVar.active#
 <br>Age:#myReturnVar.age#
 <br>Hire Date: #myReturnVar.hiredate#
 <br>Favorite Number: #myReturnVar.number#
</cfoutput>
```

You access elements of the variable myReturnVar using dot notation in the same way that you access structure fields. If a complex type has nested elements, in the way a structure can have multiple levels of nested fields, you use dot notation to access the nested elements, as in a.b.c.d, to whatever nesting level is necessary.
However, the variable myReturnVar is not a ColdFusion structure. It is a container for the complex type, but has none of the attributes of a ColdFusion structure. Calling the ColdFusion function `isStruct` on the variable returns

© 2014 Adobe Systems Incorporated. All rights reserved.

False.
You can copy the contents of the variable to a ColdFusion structure, as the following example shows:

```
<cfscript>
...
 ws = createObject("webservice", "http://somehost/echosimple.asmx?wsdl");
 myReturnVar = ws.echoStruct(stUser);

 realStruct = structNew();
 realStruct.active = #myReturnVar.active#;
 realStruct.fname = "#myReturnVar.fname#";
 realStruct.lname = "#myReturnVar.lname#";
 realStruct.age = #myReturnVar.age#;
 realStruct.hiredate = #myReturnVar.hiredate#;
 realStruct.number = #myReturnVar.number#;

</cfscript>
```

Calling `IsStruct` on realStruct returns True and you can use all ColdFusion structure functions to process it. This example shows that ColdFusion variables and structures are useful for handling complex types returned from web services. To understand how to access the elements of a complex type written to a ColdFusion variable, inspect the WSDL file for the web service. The WSDL file defines the API to the web service and provides you with the information necessary to handle data returned from it.

Publishing web services that use complex data types

The two ColdFusion data types that do not map exactly to WSDL data types are struct and query. When you publish a ColdFusion web service that uses parameters of type struct or query, the consuming application must be able to handle the data.

> **Note**
>
> If the consumer of a ColdFusion web service is another ColdFusion application, you do not have to perform any special processing. ColdFusion correctly maps struct and query data types in the web service publisher with the consumer. For more information, see *Consuming ColdFusion web services* in Consuming web services.

Publishing structures

A ColdFusion structure can hold an unlimited number of key-value pairs where the values can be of any ColdFusion data type. While it is a useful and powerful way to represent data, it cannot be directly mapped to any XML data types defined in the SOAP 1.1 encoding and XML Schema specification. Therefore, ColdFusion structures are treated as a custom type and the complex type XML schema in WSDL looks like the following:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<complexType name="mapItem">
  <sequence>
    <element name="key" nillable="true" type="xsd:anyType"/>
    <element name="value" nillable="true" type="xsd:anyType"/>
  </sequence>
</complexType>
<complexType name="Map">
  <sequence>
    <element maxOccurs="unbounded" minOccurs="0" name="item"
type="apachesoap:mapItem"/>
  </sequence>
</complexType>
```

This complex type defines a representation of a structure, where the structure keys and values can be any type. In the WSDL mapping of a ColdFusion structure, each key-value pair in the structure points to the next element in the structure except for the final field, which contains a value. Use dot notation to access the key-value pairs.

Publishing queries

ColdFusion publishes query data types as the WSDL type QueryBean. The QueryBean data type contains two elements, as the following excerpt from a WSDL file shows:

```
<complexType name="QueryBean">
 <all>
  <element name="data" nillable="true" type="intf:ArrayOf_SOAP-ENC_Array" />
  <element name="ColumnList" nillable="true"
 type="intf:ArrayOf_SOAP-ENC_string" />
 </all>
</complexType>
```

The following table describes the elements of QueryBean:

Element name	Description
ColumnList	String array that contains column names
data	Two-dimensional array that contains query data

The WSDL file for a QueryBean defines these elements as follows:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<complexType name="ArrayOf_SOAP-ENC_Array">
 <complexContent>
  <restriction base="SOAP-ENC:Array">
   <attribute ref="SOAP-ENC:arrayType" wsdl:arrayType="SOAP-ENC:Array[]" />
  </restriction>
 </complexContent>
</complexType>
<complexType name="ArrayOf_SOAP-ENC_string">
 <complexContent>
 <restriction base="SOAP-ENC:Array">
  <attribute ref="SOAP-ENC:arrayType" wsdl:arrayType="xsd:string[]" />
 </restriction>
 </complexContent>
</complexType>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

Troubleshooting SOAP requests and responses

ColdFusion provides the following facilities for troubleshooting SOAP requests and responses:

- The `getSOAPRequest` and `getSOAPResponse` functions.
- The TCP monitor.

Viewing SOAP requests and responses

You can use the `getSOAPRequest` and `getSOAPResponse` functions to retrieve and display the XML passed to and from a web service. Although advanced users can use this information for custom functionality, you typically use these functions for debugging.
Use these functions in the following places:

- `GetSOAPRequest` Clients call this function after the web service request; web service CFCs call this function in the web service CFC method.
- `GetSOAPResponse` Clients call this function after the web service request completes; web service CFCs cannot use this method.
  The following example uses the `GetSOAPRequest` and `GetSOAPResponse` functions in a web service client:

```
<cfscript>
 ws = CreateObject("webservice",
"http://localhost:8500/soapexamples/tester.cfc?WSDL");
 addSOAPRequestHeader(ws, "http://mynamespace/", "username", "randy");
 ret = ws.echo_me("value");
</cfscript>

<cfset soapreq = GetSOAPRequest(ws)>
<h2>SOAP Request</h2>
<cfdump var="#soapreq#">

<cfset soapresp = GetSOAPResponse(ws)>
<h2>SOAP Response</h2>
<cfdump var="#soapresp#">
...
```

The following example uses the `GetSOAPRequest` function in a web service CFC method:

```
<cfcomponent displayName="testerdebug" hint="Test for underscores">

<cffunction access="remote" name="echo_me" output="false" returntype="string"
displayname="Echo Test" hint="Header test">
<cfargument name="in_here" required="true" type="string">
  <cfset var soapreq = "">

<cfif IsSOAPRequest()>
<cfset soapreq = GetSOAPRequest()>
 <cflog text="#soapreq#"
 log="APPLICATION"
 type="Information">
...
```

© 2014 Adobe Systems Incorporated. All rights reserved.

Using the TCP monitor

TCPMonitor is a swing-based application that lets you watch the request and response flow of HTTP traffic. You can also watch the request and response flow of SOAP traffic. TCPMonitor replaces the JRun-based Sniffer tool (sniffer.exe) which existed in ColdFusion 9 and earlier, though it works the same way in its interface.

To run TCPMonitor:

On Windows and UNIX platforms, you can execute the TCPMonitor by using the following command:

```
java -cp cf_root/cfusion/lib/axis.jar org.apache.axis.utils.tcpmon [listening_port]
[target_host] [target_port]
```

The TCP Monitor main window appears. Enter the values in the main window as described in the following table:

Field	Description
Listen Port#	Enter a local port number, such as 8123, to monitor for incoming connections. Instead of requesting the usual port on which your server runs, you request this port. TCPMonitor intercepts the request and forwards it to the Target Port.
Listener	Select Listener to use TCPMonitor as a sniffer service in JRun.
Proxy	Select Proxy to enable proxy support for TCPMonitor.
Target Hostname	Enter the target host to which incoming connections are forwarded.For example, if you are monitoring a service running on a local server, the hostname is localhost.
Target Port#	Enter the port number on the target machine to which TCPMonitor connects. For example, if you are monitoring a service running on your local ColdFusion server in the server configuration, the default port number is 8500.
HTTP Proxy Support	Select this check box only to configure proxy support for TCPMonitor.

1. To add this profile to your TCPMonitor session, click Add. A tab appears for your new tunneled connection. Select the new tab.

   Note that you can also specify the port and host names when launching the tool, as in:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
java -cp cf_root/cfusion/lib/axis.jar org.apache.axis.utils.tcpmon
[listening_port] [target_host] [target_port]
```

In that case, the tool will use the specified port/host values and open a new tab automatically.

2. If port conflicts exist, TCPMonitor alerts you in the Request panel.
3. Request a page using the Listen Port defined in this TCPMonitor session. For example, if you entered 8123 for the Listen Port, enter the following URL in your browser:

```
http://localhost:8123/
```

TCPMonitor displays the current request and response information:For each connection, the request appears in the Request panel and the response appears in the Response panel. TCPMonitor keeps a log of all request-response pairs and lets you view any particular pair by selecting an entry in the top panel.

4. To save results to a file for later viewing, click Save. To clear the top panel of older requests that you do not want to save, click Remove Selected and Remove All.
5. To resend the request that you are currently viewing and view a new response, click Resend. You can edit the request in the Request panel before resending, and test the effects of different requests.
6. To change the ports, click Stop, change the port numbers, and click Start.
7. To add another listener, click the Admin tab and enter the values as described previously.
8. To end this TCPMonitor session, click Close.

#back to top

© 2014 Adobe Systems Incorporated. All rights reserved.

Using ColdFusion Web Services

ColdFusion can now expose many of its features as document or literal style SOAP web services. You can leverage ColdFusion functionality using web services from other languages like PHP, .NET, or Ruby.You can access the features of the following tags (and their child tags) as SOAP services:

- cfchart
- cfdocument
- cfimage
- cfmail
- cfpop
- cfpdf

© 2014 Adobe Systems Incorporated. All rights reserved.

WorkFlow for a ColdFusion web service

To start using ColdFusion web services, you need to first enable access to ColdFusion services using ColdFusion Administrator. Once you configure the access permissions:

1. Upload the file to the server (for example, PHP), which calls ColdFusion web services.
2. Call ColdFusion web service with the HTTP source URL pointing to the uploaded file, service user name, and password and other attributes.
3. ColdFusion server returns the response.

© 2014 Adobe Systems Incorporated. All rights reserved.

Enable ColdFusion services

Using ColdFusion Administrator you can set the access permissions for a user or create multiple users with access to different subsets of the available service. For example, you can allow an internal application access to all services, but limit the access of another application to image and charting services.
To enable service access:

1. On the Administrator Services > IP Addresses page specify the URL addresses that can access the services. You can specify individual addresses, address ranges of the form 10-30, or * wild cards. You can specify IPv4 or IPv6 addresses. For example, you can use the following address patterns:10.**..10**.30-50.20-3010:10:10:10:10:10:10-FF:*
2. On the Administrator Service > User Manager page, specify the users that can access the services. You must specify the user name and password. Also specify the allowed services in the Exposed Services section of the page.

© 2014 Adobe Systems Incorporated. All rights reserved.

AccessColdFusionservicesusingSOAP

Access ColdFusion services using SOAP

You use SOAP to access ColdFusion services as document/style web services. To see the available service requests and responses, view the service WSDL in the cfWebRoot\CFIDE\services directory. The service WSDL is available from the following CFC files: chart.cfc, document.cfc, image.cfc, mail.cfc, pdf.cfc, pop.cfc.
For example, to see the image services, use a URL such as the following:

```
http://<CF server >:<port>/CFIDE/services/image.cfc?wsdl
```

All the methods and attributes exposed through services can be found in WSDL. Attribute explanation is same as the attributes of corresponding ColdFusion TAGS/Functions.

© 2014 Adobe Systems Incorporated. All rights reserved.

Access ColdFusion services from a PHP server

To access ColdFusion services from PHP:

1. Install PHP
2. Use a library or framework, which helps convert WSDL to PHP classes, for example, "WSO2 Web services Frame Work for PHP".
3. Invoke the class with required parameters.The following example shows a section of a PHP page that adds a border to an image:

```
$input = new AddBorder();
//Fill in the class fields of $input to match your business logic
$input->serviceusername = "myuser";
$input->servicepassword = "mypassword";
$input->source = "http://myPHPSite/Images/image.jpg";
$input->thickness = "30";
$input->color = "blue";
$input->bordertype = "";
// call the operation
$response = $proxy->AddBorder($input);
```

In this code snippet, `serviceusername` is the user name set in the ColdFusion administrator with the permission to access the specific service being requested.`servicepassword` is the password set for the service user name.`source` is the source URL of the image in the PHP server.Other attributes are the same as the `ImageAddborder()` ColdFusion function.

Batch operation on Image:

The following code performs multiple operations on an uploaded image such as `batchOperation()` to crop an image and add a border to it.

© 2014 Adobe Systems Incorporated. All rights reserved.

```
$input = new batchOperation();
//TODO: fill in the class fields of $input to match your business logic
//Crop
$element1 = new Element();
$element2 = new Element();
$element3 = new Element();
$element4 = new Element();
$elementArray1 = new ArrayOf_xsd_anyType();
$element1->key = 'x';
$element1->value = '10';
$element2->key = 'y';
$element2->value = '10';
$element3->key = 'width';
$element3->value = '200';
$element4->key = 'height';
$element4->value = '200';
$elementArray1->item[0] = $element1;
$elementArray1->item[1] = $element2;
$elementArray1->item[2] = $element3;
$elementArray1->item[3] = $element4;
$ElementcollectionCrop = new Elementcollection();
$ElementcollectionCrop->key = 'Crop';
$ElementcollectionCrop->value = $elementArray1;
//AddBorder
$element5 = new Element();
$element6 = new Element();
$element7 = new Element();
$elementArray2 = new ArrayOf_xsd_anyType();
$element5->key = 'thickness';$element5->value = '30';
$element6->key = 'color';
$element6->value = 'green';
$element7->key = 'bordertype';
$element7->value = '';
$elementArray2->item[0] = $element5;
$elementArray2->item[1] = $element6;
$elementArray2->item[2] = $element7;
$ElementcollectionAddBorder = new Elementcollection();
$ElementcollectionAddBorder->key = 'AddBorder';
$ElementcollectionAddBorder->value = $elementArray2;
$input->serviceusername = "myuser";
$input->servicepassword = "mypassword";
$input->source = "http:/<php server>:<port>/image.jpg";
$input->attributes =
array($ElementcollectionCrop,$ElementcollectionAddBorder);
// call the operation
$response = $proxy->batchOperation($input);
```

© 2014 Adobe Systems Incorporated. All rights reserved.

Access a ColdFusion service from .NET

Create a new Web service project in Visual Studio and add ColdFusion WSDL as a web reference to the project. Then add a new item such a web form.

Example: Creating a PDF using .NET

Following is the sample .NET code for creating a PDF from the web site www.google.com.

```
Document.DocumentService objWebService = new
Document.DocumentService();
Document.Documentsection[] docsectionArray = { };
Document.Documentitem[] docitemArray = { };
string result = objWebService.generate("myuser", "mypassword", "pdf",
"", "", "", "", "yes", "", "", "", "", "","", "", "", "", "", "", "", "", "",
"", "", "", "http://www.google.com/", "", "", "", "", docsectionArray,
docitemArray);
```

In this code snippet:
Document is the web reference for document WSDL.
generate() is the method to generate the PDF from the HTML source.

© 2014 Adobe Systems Incorporated. All rights reserved.

Web service enhancements in ColdFusion 10

ColdFusion 10 has Axis 2 Web service framework integrated. This enables your web services to use WSDL 2 specifications, SOAP 1.2 protocol, and document literal wrapped style. Also the enhancements resolve many interoperability issues that you might encounter while working with Web services in ColdFusion 9.
The following table shows how the integration helps:

Consumption features	Publishing features
Supports WSDL 1.1 and WSDL 2.0 specifications.	Supports WSDL 1.1 and WSDL 2.0 specifications. • To access WSDL2, use the URL as follows: {{ http://localhost:8500/ <path of cfc>?wsdl2.}} • To access WSDL1, use the URL as follows: {{ http://localhost:8500/ <path of cfc>?wsdl.}}
Axis 2 support in ColdFusion lets you consume web services that publish WSDL in the following styles: • RPC encoded • Document Literal • Document Literal Wrapped To use Axis 2 for consumption purpose, specify `wsversion` as 2 in `cfinvoke` tag. You can also specify the version at the application level. For details, see Application level changes. If you have specified `wsversion` as `1` in `cfinvoke`, then Axis 1 is used. Axis 1 services published from ColdFusion are consumed only by Axis 1. If no value is specified, then If you publish WSDL in RPC encoded style, ColdFusion uses Axis 1 to consume the web service whereas, if you publish in either Document Literal or Document Literal Wrapped, the Axis 2 is used for consumption.	In ColdFusion 9, only the following WSDL styles were supported: • RPC Encoded • Document Literal The `style` attribute that you specified in the `cfcomponent` tag decided the style used for publishing the WSDL.In ColdFusion 10, integration with Axis 2 lets you publish the WSDL in the new style Document Literal Wrapped. For publishing WSDL in this style, specify the `style` attribute as `wrapped` in `cfcomponent`.Note that you can specify `style` attribute as `wrapped` only if the `wsversion` specified is 2.

© 2014 Adobe Systems Incorporated. All rights reserved.

Supports SOAP 1.1 and SOAP 1.2 protocols.

- While accessing the web service using `cfinvoke`, specify the serverport as shown in the following code:

```
<cfset str = structNew()>
<cfset str.serviceport =
"cfsuite.webservices.axis2.wscf.
basic.cfcHttpSoap12Endpoint">
<cfscript>
//invoke method
 ws = createObject("webservice",
 "http://localhost:8500/cfsuite/
webservices/axis2/wscf/basic.cfc
?wsdl",str);
</cfscript>
<cfinvoke webservice="#ws#"
method="Echo"
returnvariable="foo">
<cfoutput>#foo#</cfoutput>
```

Here, you have specified SOAP 1.2 endpoint and therefore, SOAP 1.2 protocol is used to send SOAP messages.|Supports SOAP 1.1 and SOAP 1.2. ColdFusion 9 supports only SOAP 1.1 endpoint. Integration with Axis 2 in ColdFusion 10 provides support for SOAP 1.1 and SOAP 1.2 protocols.That is, if you publish a Web service with `WSVersion` set to 2, the endpoints W3C SOAP 1.2 and W3C SOAP 1.1 are supported whereas if you specify 1 as the value, only W3C SOAP 1.1 is supported. For details on specifying WSVersion, see the configuration section.For details on SOAP, see www.w3.org/TR/SOAP/.|

Specifying the Axis settings

You can specify the Axis version at server level, application level, or component level.

Server level changes

Modify the following section in the neo-xmlrpc.xml available in the directory CFusion\lib.

```
<struct type='coldfusion.server.ConfigMap'>
 <var name='version'>
  <string>  2</string> </var>
</struct>
```

You could also specify web service version in the application.cfc by modifying the following tag:

```
<cfset this.wssettings.version.publish="1|2">
```

Application level changes

© 2014 Adobe Systems Incorporated. All rights reserved.

You can specify the Axis version that you want to use at the application level as follows:

- **For publication:** `<cfset this.wssettings.version.publish="2">`.

  > **Note**
  >
  > The version you specify overrides the version specified at server level.

- **For consumption:** `<cfset this.wssettings.version.consume="2">`

Setting the attribute style at application level

Set as follows: `<cfset this.wssettings.style="rpc/document/wrapped">`

Handing ColdFusion's complex datatypes in Web services

Set the `includeCFTypesInWSDL` attribute at application level as follows: `<cfset this.wssettings.includeCFTypesInWSDL="true">`
The default value is `true`. If set to `true`, schemas are generated for complex datatypes and included in the WSDL. At the client-side, stubs are generated for complex datatypes. User can pass it as the argument.The following scenarios explain the need of setting the attribute `includeCFTypesInWSDL`:

```
<cffunction name="echoObject" access="remote" returnType="any">
   <cfargument name="argObj" type="any">
   <cfdump var="#argObj#" output="console">
   <cfreturn argObj>
 </cffunction>
```

Axis 2 generates WSDL by introspecting CFC's method signatures. In this function, both argument type and return type are `any`. Therefore, the schema generated with WSDL will not contain the schemas for ColdFusion's complex datatypes such as struct, query, or xml. If these schemas are not present in the WSDL, then at the client side, there will not be stub classes for the complex types, making the function useless.Similarly, in the following function, both argument type and return type are struct. In this case also, the WSDL will contain schema only for the struct. If you want to pass Document inside struct or Query inside struct, they are not available as schemas in the WSDL.

```
<cffunction name="echoComplexStruct" access="remote" returnType="struct">
   <cfargument name="argStr" type="struct">
   <cfdump var="#argStr#" output="console">
   <cfreturn argStr>
</cffunction>
```

Component level changes

- New attribute `wsversion` has been added to `cfcomponent.}}If you specify {{2`, CFC is deployed using Axis 2 engine. The value you specify overrides the value you specify at application or server level.
- A new value `wrapped` can be specified for the attribute `style`. If you are setting `wsversion` as `2`, the default value is `wrapped` and if it is `1`, then the default value is `rpc`.

© 2014 Adobe Systems Incorporated. All rights reserved.

> **Note**
>
> If you have set style at the application level, in both the cases, instead of the default values, the values at the application level are used.

The syntax is as follows:

```
<cfcomponent
 wsversion = 1|2>
 style = "rpc|document|wrapped"
 ....
 <cffunction ...>
        ...
    </cffunction>

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

Modifications to createObject and cfobject

- New property `wsVersion` has been added to `createObject` to specify the Axis version.
- New attribute `wsVersion` has been added to the tag `cfobject` to specify the Axis version.

Variations from ColdFusion 9

If the same display name is used in multiple CFCs, in ColdFusion 9, both the CFCs are published. But in ColdFusion 10, if `wsVersion` is set to 2, in the case of multiple CFCs with same display name, only the first CFC is published. Attempt to access the second CFC results in an error.
Web services are not automatically registered when you access the service using `cfinvoke`, `cfobject,` or `creat eObject`. You have to register the Web service in the ColdFusion Administrator (Data & Services > Web Services).

Limitations

> **Note**
>
> The following limitations will be addressed in the future releases of ColdFusion 10.

- Unlike in Axis 1, Axis 2 cannot support two CFCs having the same display name.
- The following attributes in the tag `cfinvoke` are not supported in Axis 2: `Serviceaddress`, `portTypeNam e`.
- If you are publishing JWS files, only Axis 1 is supported.

© 2014 Adobe Systems Incorporated. All rights reserved.

RESTful Web Services in ColdFusion Developing Guide

ColdFusion 10 lets you create and publish REST (Representational State Transfer) services that can be consumed by clients over HTTP/HTTPS request.
For details, see RESTful Web Services in ColdFusion.

#back to top

© 2014 Adobe Systems Incorporated. All rights reserved.

Integrating JEE and Java Elements in CFML Applications

You can integrate JEE elements, including JSP pages and servlets; JSP tags; and Java objects, including Enterprise JavaBeans (EJBs); into your Adobe ColdFusion application.

© 2014 Adobe Systems Incorporated. All rights reserved.

About ColdFusion, Java, and JEE

ColdFusion is built on a Java Enterprise Edition (JEE, formerly termed "J2EE") technology platform. This structure lets ColdFusion applications take advantage of, and integrate with, JEE elements. ColdFusion pages can do any of the following:

- Include JavaScript and client-side Java applets on the page.
- Use JSP tags.
- Interoperate with JSP pages.
- Use Java servlets.
- Use Java objects, including JavaBeans and Enterprise JavaBeans.

About ColdFusion and client-side JavaScript and applets

ColdFusion pages, like HTML pages, can incorporate client-side JavaScript and Java applets. To use JavaScript, you write the JavaScript code just as you do on any HTML page. ColdFusion ignores the JavaScript and sends it to the client.
The `cfapplet` tag simplifies using Java client-side applets.

Use an applet on a ColdFusion page

1. Register the applet .class file in ColdFusion Administrator Java Applets Extensions page. (For information on registering applets, see the ColdFusion Administrator online Help.)
2. Use the cfapplet tag to call the applet. The `appletSource` attribute must be the Applet name assigned in the ColdFusion Administrator.
   For example, ColdFusion includes a Copytext sample applet that copies text from one text box to another. The ColdFusion Setup automatically registers the applet in the Administrator. To use this applet, incorporate it on your page. For example:

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

About ColdFusion and JSP

ColdFusion supports JSP tags and pages in the following ways:

- Interoperates with JSP pages: ColdFusion pages can include or forward to JSP pages, JSP pages can include or forward to ColdFusion pages, and both types of pages can share data in persistent scopes.
- Imports and uses JSP tag libraries: the cfimport tag imports JSP tag libraries and lets you use its tags. ColdFusion pages are not JSP pages, however, and you cannot use most JSP syntax on ColdFusion pages. In particular, you *cannot* use the following features on ColdFusion pages:
  Include, Taglib, and Page directives: Instead, you use CFML `import` tag to import tag libraries, and the `include` (or `forward`) method of the page context object returned by the ColdFusion `GetPageContext` function to include pages. For more information, see Using JSP tags and tag libraries and Interoperating with JSP pages and servlets.
  Expression, Declaration, and Scriptlet JSP scripting elements: Instead, you use CFML elements and expressions.
  JSP comments: Instead, you use CFML comments. (ColdFusion ignores JSP comments and passes them to the browser.)
  Standard JSP tags: Such as jsp:plugin, unless your JEE server provides access to these tags in a JAR file. Instead, you use ColdFusion tags and the PageContext object.

About ColdFusion and servlets

© 2014 Adobe Systems Incorporated. All rights reserved.

Some Java servlets are not exposed as JSP pages; instead they are Java programs. You can incorporate JSP servlets in your ColdFusion application. For example, your enterprise could have an existing servlet that performs some business logic. To use a servlet, the ColdFusion page specifies the servlet by using the ColdFusion GetPageContext function.
When you access the servlet with the `GetPageContext` function, the ColdFusion page shares the Request, Application, and Session scopes with the servlet, so you can use these scopes for shared data.
ColdFusion pages can also access servlets by using the `cfservlet` tag, use the servlet URL in a `form` tag, or access an SHTML page that uses a `servlet` tag.

> **Note**
>
> The cfservlet tag, which provides access to servlets on JRun servers, is deprecated since ColdFusion MX.

About ColdFusion and Java objects

Java objects include the following:

- Standard Java classes and methods that make up the JEE API
- Custom-written Java objects, including the following:
  - Custom classes, including JavaBeans
  - Enterprise JavaBeans
    ColdFusion pages use the `cfobject` tag to access Java objects.
    ColdFusion searches for the objects in the following order:

1. The ColdFusion Java Dynamic Class Load directories:
2. Java archive (.jar) files in *web_root*/WEB-INF/lib
3. Class (.class) files in *web_root*/WEB-INF/classes
   ColdFusion reloads classes from these directories, as described in the next section, "About class loading."

1. The classpath specified on the JVM and Java Settings page in the ColdFusion Administrator.
2. The default JVM classpath.

About class loading

ColdFusion dynamically loads classes that are either .class files in the *web_root*/WEB-INF/classes directory or in JAR files in the *web_root*/WEB-INF/lib directory. ColdFusion checks the timestamp on the file when it creates an object that is defined in either directory, even when the class is already in memory. If the file that contains the class is newer than the class in memory, ColdFusion loads the class from that directory. To use this feature, make sure that the Java implementation classes that you modify are not in the general JVM classpath.To disable automatic class loading of your classes, place the classes in the JVM classpath. Classes located on the JVM classpath are loaded once per server lifetime. To reload these classes, stop and restart ColdFusion.

About GetPageContext and the PageContext object

Because ColdFusion pages are JEE servlet pages, all ColdFusion pages have an underlying Java PageContext object. CFML includes the GetPageContext function that you can then use in your ColdFusion page. The PageContext object exposes several fields and methods that can be useful in JEE integration. In particular, it includes the `include` and `forward` methods that provide the equivalent of the corresponding standard JSP tags. For more information on other features of the PageContext object, see Java documentation. You can find the Javadoc description of this class at http://docs.oracle.com/javaee/7/api/javax/servlet/jsp/PageContext.html.

About CFML variables and Java variables

© 2014 Adobe Systems Incorporated. All rights reserved.

Because ColdFusion variables are case-independent and Java variables are case-dependent, be careful about variable names. Use the following rules and guidelines when sharing data between ColdFusion and Java code, including JSP pages and servlets.

Rules

- If you use mixed case variables, all variable names must be unique, independent of case. For example, you must not have two Java variables, MyVariable and MYVARIABLE. ColdFusion cannot distinguish between the two.
- If you share Request scope variables between a CFML page and a JSP page or servlet, all shared Request scope variable names *must* be all-lowercase in the JSP page or servlet. Mixed case or all-uppercase variables cause null pointer exceptions if CFML refers to these variables.
- If you share Application or Session scope variables between a CFML page and a JSP page or servlet and use a named ColdFusion application (the common usage), the variables on the JSP page or servlet are case-independent.
- If you share the Application or Session scope variables between a CFML page and a JSP page or servlet, and use an *unnamed* ColdFusion application, the variable names in the JSP page or servlet *must* be all lowercase.
- When you specify a class name in the `cfobject` tag or `CreateObject` function, the name must be case-correct.

Guidelines

- You can prevent problems by consistently using all-lowercase variable names.
- In your CFML, use the same case as you do in your Java or JSP. Doing so does not change how the application works, but does help prevent confusion.

© 2014 Adobe Systems Incorporated. All rights reserved.

Interoperating with JSP pages and servlets

ColdFusion pages and JSP pages can interoperate in several ways:

- ColdFusion pages can invoke JSP pages and servlets.
- JSP pages can invoke ColdFusion pages.
- ColdFusion pages, JSP pages, and servlets can share data in three scopes.

Integrating JSP and servlets in a ColdFusion application

You can integrate JSP pages and servlets in your ColdFusion application. For example, you can write some application pages in JSP and write others in CFML. ColdFusion pages can access JSP pages by using the JSP `include` and `forward` methods to call the page. As with any web application, you can use `href` links in ColdFusion pages to open JSP pages.
The ability to use JSP lets you incorporate legacy JSP pages in your ColdFusion application, or conversely, use CFML to expand an existing JSP application using ColdFusion pages.
If you have a JSP page that must call a ColdFusion page, you also use a `jsp:forward` or `jsp:include` tag to call the ColdFusion page. For an example of calling a ColdFusion page from a JSP page, see *Calling a JSP page from a ColdFusion page* in this page.

Accessing a JSP page or servlet from a ColdFusion page

To access a JSP page or servlet from a ColdFusion page, you use the GetPageContext function with the `forward` or the `include` method. For example, to include a JSP "Hello World" page in your ColdFusion application, use the following line:

```
GetPageContext().include("hello.jsp");
```

To pass parameters to the JSP page, include the parameters in the page URL.
For example, you could want to integrate an existing JSP customer response component into a new ColdFusion order processing application. The order processing application provides the order number, total cost, and expected shipping date, and the customer response component sends the response to the e-mail address on file for the particular customer number. The ColdFusion application could use the following CFScript code to call the response JSP page:

```
urlParams =
"UID=#order.uid#&cost=#order.total#&orderNo=#order.orderNo#&shipDate=#order.shipDate
No#"
getPageContext().forward(URLEncodedFormat("/responsegen/responsegen.jsp?#urlParams#"
));
```

To access a servlet that exposes the same functionality, you use the same code, although the URL would change. For example, to run a servlet called HelloWorldServlet, you place the servlet .java or .class file in the _serverroot/_WEB-INF/classes directory and access the servlet with the URL /servlet/HelloWorldServlet.

Accessing ColdFusion application and session variables in JSP pages

ColdFusion runs as a JEE application on a JEE application server. The JEE application ServletContext is a data structure that stores objects as attributes. A ColdFusion Application scope is represented as an attribute named by the Application scope name. The attribute contains the scope values as a hash table. Therefore, you access

© 2014 Adobe Systems Incorporated. All rights reserved.

ColdFusion Application scope variable in a JSP page or servlet using the following format:

```
((Map)application.getAttribute("CFApplicationName"))).get("appVarName")
```

Similarly, the ColdFusion Session scope is a structure within the JEE session (assuming that the *J2EE Sessions* fea ture has been enabled, as discussed in the next session.) Because ColdFusion identifies sessions by the application name. the session structure is contained in an attribute of the JEE session that the application name identifies. Therefore, you access ColdFusion session variables as follows:

```
((Map)(session.getAttribute("CFApplicationName"))).get("sessionVarName")
```

Sharing data between ColdFusion pages and JSP pages or servlets

If an application includes ColdFusion pages and JSP pages or servlets, they can share data in the Request, Session and Application scopes. The following table lists the ways that you can access JSP pages with which you want to share the scope data:

Scope	Can share data using
Request	forward, include*Note:* Shared Request scope variable names in the JSP page or servlet must be all-lowercase.
Session	href, cfhttp, forward, include
Application	href, cfhttp, forward, include

Note

When you share data between ColdFusion pages and JSP pages, be careful about data type conversion issues. For more information, see *Java and ColdFusion data type conversions* in Usin g Java objects.

To share session variables, specify J2EE session management in the ColdFusion Administrator. For more information on configuring and using J2EE Session scope management, see *ColdFusion and J2EE session management* in Configuring and using session variables. (While the term "J2EE" has been updated in Java parlance with the simpler JEE, the ColdFusion Administrator still refers to storing sessions in the JEE servlet scope as "J2EE sessions".)

For example, you could put the customer order structure used in the previous example in the Session scope. Then, you would not have to pass the order values as a set of parameters. Instead, the JSP pages could access the Session scope variables directly, and the ColdFusion page would only require a line like the following to call the JSP page:

```
getPageContext().forward(URLEncodedFormat("/responsegen/responsegen.jsp"));
```

© 2014 Adobe Systems Incorporated. All rights reserved.

For examples of using the Request, Session, and Application scopes to share data between ColdFusion pages and JSP pages, including samples of the appropriate JSP code, see *Using JSP with CFML* below.

> **Note**
>
> When running in the server configuration, ColdFusion also shares the Form scope when calling a JSP or servlet. In the JEE configuration, however, sharing the Form scope is dependent on the JEE application server. For example, JRun shares the Form scope, IBM WebSphere does not. ColdFusion always shares the Request, Session, and Application scopes.

Unnamed ColdFusion Application and Session scopes

If you do not specify an application name in the This.name variable in the Application.cfc initialization code or by using the ColdFusion cfapplication tag, the application is unnamed, and the Application scope corresponds to the ColdFusion JEE servlet context. ColdFusion, therefore, supports only a single unnamed application. If multiple `cfapplication` tags and Application.cfc files do not specify an application name, all pages in these applications share the servlet context as their Application scope.

All sessions of unnamed applications correspond directly to the session object of the JEE application server. (If you do not use J2EE session variables, ColdFusion ensures that the JEE session lasts at least as long as the session time-out.)

You access an Application scope variable from a ColdFusion unnamed application in a JSP page using the following format:

```
application.getAttribute("applicationVariableName")
```

You access Session scope variables in a ColdFusion unnamed application as follows:

```
session.getAttribute("sessionVariableName")
```

> **Note**
>
> When you use application and session variables for the unnamed ColdFusion application in JSP pages and servlets, the variable names must be case-correct. The characters in the variable name must have the same case as you used when you created the variable in ColdFusion. You do not have to use case-correct application and session variable names for named ColdFusion applications.

Examples: using JSP with CFML

The following simple examples show how you can integrate JSP pages, servlets, and ColdFusion pages. They also show how you can use the Request, Application, and Session scopes to share data between ColdFusion pages, JSP pages, and servlets.

Calling a JSP page from a ColdFusion page

The following page sets Request, Session, and application variables and calls a JSP page, passing it a name

© 2014 Adobe Systems Incorporated. All rights reserved.

parameter:

```
<cfapplication name="myApp" sessionmanagement="yes">
<cfscript>
Request.myVariable = "This";
Session.myVariable = "is a";
Application.myVariable = "test.";
GetPageContext().include("hello.jsp?name=Bobby");
</cfscript>
```

Reviewing the code

The following table describes the CFML code and its function:

Code	Description
`<cfapplication name="myApp" sessionmanagement="yes">`	Specifies the application name as myApp and enables session management. In most applications, this tag is in the Application.cfm page.

`<cfscript>`
`Request.myVariable = "This";`
`Session.myVariable = "is a";`
`Application.myVariable = "test.";`
	Sets ColdFusion Request, Session, and Application, scope variables. Uses the same name, myVariable, for each variable.

`GetPageContext().include("hello.jsp?name=Bobby");` `</cfscript>`	Uses the GetPageContext function to get the current servlet page context for the ColdFusion page. Uses the `include` method of the page context object to call the hello.jsp page. Passes the name parameter in the URL.

The ColdFusion page calls the hello.jsp page. It displays the `name` parameter in a header and the three variables in the remainder of the body.

```
<%@page import="java.util.*" %>
<h2>Hello <%= request.getParameter("name")%>!</h2>

<br>Request.myVariable: <%= request.getAttribute("myVariable")%>
<br>session.myVariable: <%= ((Map)
(session.getAttribute("myApp"))).get("myVariable")%>
<br>Application.myVariable: <%=
((Map)(application.getAttribute("myApp"))).get("myVariable")%>
```

The following table describes the JSP code and its function (line breaks added for clarity):

© 2014 Adobe Systems Incorporated. All rights reserved.

Code	Description
`<%@page import="java.util.*" %>`	Imports the java.util package. This package contains methods required in the JSP page.
`<h2>Hello <%= request.getParameter ("name")%>!</h2>`	Displays the name passed as a URL parameter from the ColdFusion page. The parameter name is case sensitive,Note: The `getParameter` request method cannot get all ColdFusion page request parameter values on some application servers. For example, on IBM WebSphere, you cannot use `getParameter` to get form fields.
` request.myVariable: <%= request. getAttribute("myvariable")%>`	Uses the `getAttribute` method of the JSP request object to displays the value of the Request scope variable myVariable.The JSP page must use all lowercase characters to reference all request scope variables that it shares with CFML pages. You can use any case on the CFML page, but if you use mixed case or all uppercase on the JSP page, the variable is not set from the ColdFusion page.
` session.myVariable: <%= ((Map)(session.getAttribute("myApp ")) ).get("myVariable")%>`	Uses the `getAttribute` method of the JSP session object to get the myApp object (the Application scope). Casts this object to a Java Map object and uses the get method to obtain the myVariable value for display.CFML pages and JSP pages share Session variables independent of the variable name case. The variable on the JSP page can have any case mixture and still receive the value from the ColdFusion page. For example, instead of myVariable, you could use MYVARIABLE or myvariable on this line.
` Application.myVariable: <%=((Map) (application.getAttribute("myApp") )).get("myVariable")%>`	Uses the `getAttribute` method of the JSP myApp application object to obtain the value of myVariable in the Application scope.CFML pages and JSP pages share Application variables independent of the variable name case. The variable on the JSP page can have any case mixture and still receive the value from the ColdFusion page. For example, instead of myVariable, you could use MYVARIABLE or myvariable on this line.

Calling a ColdFusion page from a JSP page

The following JSP page sets Request, Session, and application variables and calls a ColdFusion page, passing it a name parameter:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<%@page import="java.util.*" %>

<% request.setAttribute("myvariable", "This");%>
<% ((Map)session.getAttribute("myApp")).put("myVariable", "is a");%>
<% ((Map)application.getAttribute("myApp")).put("myVariable", "test.");%>

<jsp:include page="hello.cfm">
 <jsp:param name="name" value="Robert" />
</jsp:include>
```

Reviewing the code

The following table describes the JSP code and its function:

Code	Description
`<%@page import="java.util.*" %>`	Imports the java.util package. This package contains methods required in the JSP page.
`<% request.setAttribute("myvariable", "This");%>`	Uses the `setAttribute` method of the JSP request object to set the value of the Request scope variable myVariable. The JSP page must use all lowercase characters to refer to all request scope variables that it shares with CFML pages. You can use any case on the CFML page, but if you use mixed case to all uppercase on the JSP page, the JSP page does not share it with the ColdFusion page.
`<% ((Map)session.getAttribute("myApp")).put("myVariable", "is a");%>`	Uses the `getAttribute` method of the JSP session object to get the myApp object (the Application scope). Casts this object to a Java Map object and uses the `set` method to set the myVariable value.CFML pages and JSP pages share Session variables independent of the variable name case. The variable on the JSP page can have any case mixture and still share the value with the ColdFusion page. For example, instead of myVariable, you could use MYVARIABLE or myvariable on this line.

© 2014 Adobe Systems Incorporated. All rights reserved.

`<% ((Map)application.getAttribute("myApp")).put("myVariable","test.");%>`	Uses the `getAttribute` method of the JSP application object to get myApp object (the Application scope) and casts it to a Map object. It then sets the value of myVariable in the myApp application scope object.CFML pages and JSP pages share Application variables independent of the variable name case. The variable on the JSP page can have any case mixture and still share the value with the ColdFusion page. For example, instead of myVariable, you could use MYVARIABLE or myvariable on this line.
`<jsp:include page="hello.cfm">` `<jsp:param name="name" value="Robert"/>` `</jsp:include>`	Sets the name parameter to Robert and calls the ColdFusion page hello.cfm.

The JSP page calls the following hello.cfm page. It displays the Name parameter in a heading and the three variables in the remainder of the body.

```
<cfapplication name="myApp" sessionmanagement="yes">
<cfoutput>
<h2>Hello #URL.name#!</h2>
Request.myVariable: #Request.myVariable#<br>
Session.myVariable: #Session.myVariable#<br>
Application.myVariable: #Application.myVariable#<br>

</cfoutput>
```

The following table describes the CFML code and its function:

Code	Description
`<cfapplication name="myApp" sessionmanagement="yes">`	Specifies the application name as myApp and enables session management. In most applications, this tag is in the Application.cfm page.
`<cfoutput><h2>Hello #URL.name#!</h2>`	Displays the name passed using the `jsp:param` tag on the JSP page. The parameter name is not case sensitive.

© 2014 Adobe Systems Incorporated. All rights reserved.

`Request.myVariable:` `#Request.myVariable# ` `Session.myVariable:` `#Session.myVariable# ` `Application.myVariable:` `#Application.myVariable# ` `</cfoutput>`	Displays the Request.myVariable, Session.myVariable, and Application.myVariable values. All variable names on CFML pages are case independent.

© 2014 Adobe Systems Incorporated. All rights reserved.

Using JSP tags and tag libraries

You can use JSP tags from any JSP tag library. For example, you can use any of the custom tags in the open-source Apache Jakarta Project Taglibs project tag libraries, located at http://jakarta.apache.org/taglibs/index.html. This project consists of several individual JSP custom tag libraries for purposes ranging from JNDI access to generating random text strings.

Using a JSP tag in a ColdFusion page

JSP pages use a standard set of tags, such as `jsp:forward` and `jsp:include`. You can also import custom JSP tag libraries into a JSP application. You can use both the standard JSP tags and custom JSP tags in ColdFusion pages.

Standard JSP tags and ColdFusion

ColdFusion tags provide equivalent features to most standard JSP tags. For example, the cfapplet tag provides the same service as the `jsp:plugin` tag, and cfobject tag lets you use JavaBeans, as does the `jsp:usebean` tag. Similarly, you do not use the `jsp:getproperty` tag because ColdFusion automatically gets properties when you reference them. Therefore, ColdFusion does not support the use of standard JSP tags directly.However, two standard JSP tags provide functionality that is useful in ColdFusion pages: the `forward` and `include` tags invoke JSP pages and Java servlets. The PageContext object described in *About GetPageContext and the PageContext object* section has `forward` and `include` methods that provide the same operations. For more information about using these methods, see *Accessing a JSP page or servlet from a ColdFusion page* in Interoperating with JSP pages and servlets.

Using custom JSP tags in a ColdFusion page

Follow these steps to use a custom JSP tag on a ColdFusion page:

Use a custom tag

1. Place the tag library, consisting of the *taglibname*.jar file, and the *taglibname*.tld file, if one is supplied, in the *web_root*/WEB-INF/lib directory. The JSP custom tag library must be in this directory for you to use the `cfimport` tag.
2. Restart ColdFusion.
3. In the ColdFusion page that uses a JSP tag from the tag library, specify the tag library name in a `cfimport`tag; for example:

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

If the TLD file is not included in the JAR file, use the .tld extension in place of the .jar extension.

> **Note**
>
> The `cfimport` tag must be on the page that uses the imported tag. You cannot place the `cfimport_` tag in Application.cfm._

1. Use the custom tag using the form *prefix*:*tagName*; for example:

```
<random:number id="myNum" range="000000-999999" />
```

© 2014 Adobe Systems Incorporated. All rights reserved.

Note

You cannot use the `cfsavecontent` tag to suppress output of a custom JSP tag.

Example: using the random tag library

The following example uses the random tag library from the Apache Jakarta Taglibs project and calls the `number` tag. The `number` tag initializes a random number generator that uses a secure algorithm to generate a six-digit random number. You get a new random number each time you reference the variable randPass.random.

```
<cfimport taglib="/WEB-INF/lib/taglibs-random.jar" prefix="myrand">
<myrand:number id="randPass" range="000000-999999" algorithm="SHA1PRNG"
provider="SUN" />
<cfset myPassword = randPass.random>
<cfoutput>
 Your password is #myPassword#<br>
</cfoutput>
```

For more information on the Jakarta random tag library and how to use its tags, see the documentation at the Apache Jakarta Taglibs project website, http://jakarta.apache.org/taglibs/index.html. The Taglibs project includes many open source custom tag libraries.

© 2014 Adobe Systems Incorporated. All rights reserved.

Enhanced Java integration in ColdFusion 10

Integrating Java libraries

ColdFusion 10 lets you load Java libraries from a custom path that you specify. In the previous versions, you use Java libraries placed in the `lib` directory of ColdFusion. Those libraries are not application-specific and adding a Java library or updating an existing library is not easy. You also have to restart ColdFusion.

ColdFusion 10 lets you place the Java libraries for an application in a directory of your choice. You specify the path of this directory in the Application.cfc. Then, use the libraries in your application by creating a `cfobject` of Java type.

Specifying custom Java library path in the Application.cfc without dynamic loading

Specify the custom path from where you want to load the Java library in the Application.cfc. of your project.

In this case, if there is an update to the file, you have to restart ColdFusion to load the updated files.

Add the following entry in this file:

```
THIS.javaSettings = {LoadPaths = [".\java_lib\",".\java\myjar.jar"],
loadColdFusionClassPath = true, reloadOnChange = false}
```

Parameters

Parameter	Description
`loadPaths`	An array of paths to the directories that contain Java classes or JAR files.You can also provide the path to a JAR or a class. If the paths are not resolved, an error occurs.
`loadColdFusionClassPath`	Indicates whether to load the classes from the ColdFusion lib directory. The default value is `false`.
`reloadOnChange`	Indicates whether to reload the updated classes and JARs dynamically, without restarting ColdFusion. The default value is `false`.

Specifying the custom Java library path in the Application.cfc with dynamic loading

Specify the custom path from where you want to load the Java library in the Application.cfc of your project.

In this case, if there is an update to the file, you need not restart ColdFusion to load the updated files.

Add the following entry in this file:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
THIS.javaSettings = {LoadPaths = [".\java_lib\",".\java\myjar.jar"],
loadColdFusionClassPath = true, reloadOnChange= true, watchInterval = 100,
watchExtensions = "jar,class,xml"}
```

Parameters

Parameter	Description
`loadPaths`	An array of paths to the directories that contain Java classes or JAR files.You can also provide the path to a JAR or a class. If the paths are not resolved, an error occurs.
`loadColdFusionClassPath`	Indicates whether to load the classes from the ColdFusion lib directory. The default value is `false`.
`reloadOnChange`	Indicates whether to reload the updated classes and JARs dynamically, without restarting ColdFusion. The default value is `false`.
`watchInterval`	Specifies the time interval in seconds after which to verify any change in the class files or JAR files. This attribute is applicable only if the `reloadOnChange` attri bute is set to `true`. The default value is 60 seconds.
`watchExtensions`	Specifies the extensions of the files to monitor for changes. By default, only `.class` and `.jar` files are monitored.

Using a Java class

Save the Java class files or JARs in the directory that you specified in the Application.cfc. Then, access the methods in the JARs or class files by creating a cfobject of Java type.

In this example, you create a Java class and access a method in the Java class in the sample application.

1. Create a Java file, `Test.java`.

```
public class Test
 {
  public String testJava()
  {
  return "Hello Java!!";
  }
 }
```

2. Compile the Java file using a Java compiler.

© 2014 Adobe Systems Incorporated. All rights reserved.

3. Add the following entry to the `Application.cfc`of your project:

```
<cfset THIS.javaSettings = {LoadPaths =
["/myJava/lib"],reloadOnChange=true,watchInterval=30}/>
```

4. Create the following directory structure in your application folder:

```
/myJava/lib
```

5. Copy the `Test.class` file to the `/myJava/lib` folder.
6. Create a CFM file with the following content.

```
<cfobject type="java" class="Test" name="myObj">
 <cfset y = myObj.init()>
  <cfoutput >
  #y.testJava()#
  </cfoutput>
```

7. Deploy the application and access the CFM file.

Using the CFC Proxy

Using the CFC Proxy, you can access a ColdFusion component from Java classes. To call CFC, ColdFusion class loader must be the current class loader. For example, the following code creates a CFC Proxy from the file location provided:

```
CFCProxy(String fully-qualified-path-of-CFC-file)
```

Similarly, the following code creates a CFC Proxy from the file location provided. It also initializes the This scope of the CFC with the name value pairs.

```
CFCProxy(String fully-qualified-path-of-CFC-file name-value-pairs)
```

ColdFusion 10 introduces a new argument in CFProxy classes, `directInvoke`.
If this argument is set `true`, the request does not go through the ColdFusion Filter chain. It results in improved performance.
The following example creates a Java class that access a ColdFusion component using CFC Proxy.

1. Create the following Java class.

© 2014 Adobe Systems Incorporated. All rights reserved.

```
import coldfusion.cfc.CFCProxy;
public class CFCInvoker
{
 public String directInvoke(String cfcPath)
 {
  String myMessage = "";
  try
   {
    CFCProxy myCFC = new CFCProxy(cfcPath, true);
    Object[] myArgs = { "Hello" };
    myMessage = (String)myCFC.invoke("getData", myArgs);
   }
  catch (Throwable e)
   {
    e.printStackTrace();
   }
    return myMessage;
 }
}
```

2. Compile the file and place it in the lib directory of the project folder.
3. Create a ColdFusion component as follows:

```
<cfcomponent>
<cffunction name="getData" returntype="string">
 <cfargument name="msg" required="Yes">
  <cfreturn msg & "Java" />
 </cffunction>
</cfcomponent>
```

4. In the `application.cfc`, add the following entry:

```
THIS.javaSettings = {LoadPaths = ["/lib"],reloadOnChange=
true,loadColdFusionClassPath=true};
```

5. Create a CFM file to print the output:

```
<cfobject action="create" type="java" class="CFCInvoker" name="x">
<cfset cfcPath = "C:\ColdFusion10\cfusion\wwwroot\Project\name.cfc"/>
<cfset y = x.directInvoke2(cfcPath)>
<cfoutput>#y#</cfoutput>
```

6. Access the CFM file.

Using the createDynamicProxy function

The function `createDynamicProxy` creates a dynamic proxy of the ColdFusion component that is passed to a

© 2014 Adobe Systems Incorporated. All rights reserved.

Java library. Dynamic proxy lets you pass ColdFusion components to Java objects. Java objects can work with the ColdFusion components seamlessly as if they are native Java objects.
To create a dynamic proxy, provide the name of the ColdFusion component and an array of Java interfaces. The component is treated as if it implements the specified interfaces.
Create a dynamic proxy as follows:

```
createDynamicProxy("fullyQualifiedNameOfCFC", ["interfaceName"]);
```

Specify the following parameters:

- Fully qualified name of the ColdFusion component or a CFC instance.
- An array of Java interfaces for which you want to create the dynamic proxy.

Creating a dynamic proxy

The following example shows how to create a Java interface. Interface defines a method. A ColdFusion component implements the method as a ColdFusion function. The dynamic proxy of the ColdFusion component calls a Java class by passing the object of the interface. It then calls the method in ColdFusion as if it is a native Java class.

1. Create a Java interface, `MyInterface`.

```
public interface MyInterface
{
 public String sayHello();
}
```

2. Compile the Java file and place it in a directory, `lib`.
3. Create a Java class, `InvokeHelloProxy`, that calls the ColdFusion object using the instance of the interface.The constructor, `InvokeHelloProxy`, creates an object of `MyInterface`. The `invokeHello()` method calls the `sayHello()`method using the object.

```
public class InvokeHelloProxy
{
 private MyInterface myInterface;
  public InvokeHelloProxy(MyInterface x)
  {
   this.myInterface = x;
  }
  public String invokeHello()
  {
   return myInterface.sayHello();
  }
}
```

4. Compile the Java file and place it in a directory, `lib`.
5. Create a CFC file, `HelloWorld.cfc,` that implements the method defined in the interface and save it in a directory, `cfc`.

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cfcomponent>
 <cffunction name="sayHello" returntype="string">
  <cfreturn "Hello World!!!!">
 </cffunction>
</cfcomponent>
```

6. Add the following code in the Application.cfc.{{}}

```
<cfset THIS.javaSettings = {LoadPaths =
["/lib"],reloadOnChange=true,watchInterval=10}/>
```

7. Add a CFM file that creates a dynamic proxy for `HelloWorld` as interface, `MyInterface`. Create an object of `InvokeHelloProxy` class and initiate the class.The code creates a dynamic proxy of `MyInterface`.

```
<cfset dynInstnace = createDynamicProxy("cfc.HelloWorld", ["MyInterface"])>
<cfset x = createObject("java","InvokeHelloProxy").init(dynInstnace)>
<cfset y = x.invokeHello()>
<cfoutput>#y#</cfoutput>
```

Example: using a CFC instance

```
<cfset instance=new cfc.helloWorld()>
<cfset dynInstnace = createDynamicProxy(instance, ["MyInterface"])>
<cfset x = createObject("java","InvokeHelloProxy").init(dynInstnace)>
<cfset y = x.invokeHello()>
<cfoutput>#y#</cfoutput>
```

8. Deploy the application and access the CFM file.

#back to top

© 2014 Adobe Systems Incorporated. All rights reserved.

Using Java objects

You use the cfobject tag to create an instance of a Java object. You use other ColdFusion tags, such as cfset and cfoutput, or CFScript to invoke properties (attributes), and methods (operations) on the object.
Method arguments and return values can be any valid Java type; for example, simple arrays and objects. ColdFusion does the appropriate conversions when strings are passed as arguments, but not when they are received as return values. For more information on type conversion issues, see *Java and ColdFusion data type conversions* below.
The examples in this discussion assume that the `name` attribute in the `cfobject` tag specified the value obj, and that the object has a property called Property, and methods called Method1, Method2, and Method3.

> **Note**
>
> The `cfdump` tag displays the public methods and data of an object.

Using basic object techniques

You can use ColdFusion to invoke Java objects and access object methods and properties.

Invoking objects

The `cfobject` tag makes Java objects available in ColdFusion. It can access any Java class that is available on the JVM classpath or in either of the following locations:

- In a Java archive (.jar) file in *web_root*/WEB-INF/lib
- In a class (.class) file in *web_root*/WEB-INF/classes
  For example:

```
<cfobject type="Java" class="MyClass" name="myObj">
```

Although the `cfobject` tag loads the class, it does **not** create an instance object. Only static methods and fields are accessible immediately after the call to `cfobject`.
If you call a public non-static method on the object without first calling the `init` method, ColdFusion makes an implicit call to the default constructor.
To call an object constructor explicitly, use the special ColdFusion `init` method with the appropriate arguments after you use the `cfobject` tag; for example:

```
<cfobject type="Java" class="MyClass" name="myObj">
<cfset ret=myObj.init(arg1, arg2)>
```

> **Note**
>
> The `init` method is not a method of the object, but a ColdFusion identifier that calls the `new` function on the class constructor. So, if a Java object has an `init` method, a name conflict exists and you cannot call the object `init` method.

To have persistent access to an object, use the `init` function, because it returns a reference to an instance of the object, and `cfobject` does not.

© 2014 Adobe Systems Incorporated. All rights reserved.

An object created using `cfobject` or returned by other objects is implicitly released at the end of the ColdFusion page execution.

Using properties

Use the following coding syntax to access properties if the object does either of the following actions:

- Exposes the properties as public properties.
- Does not make the properties public, but is a JavaBean that provides public getter and setter methods of the form get_PropertyName_() and set_PropertyName_(*value*). For more information, see *Calling JavaBean get and set methods* below.
- To set a property: `<cfset obj.property = "somevalue">`
- To get a property: `<cfset value = obj.property>`

> **Note**
>
> ColdFusion does not require consistently capitalized property and method names. However, it is good programming practice to use the same case in ColdFusion as you do in Java to ensure consistency.

Calling methods

Object methods usually take zero or more arguments. Some methods return values, while others might not. Use the following techniques to call methods:

1. If the method has no arguments, follow the method name with empty parentheses, as in the following `cfsett` ag:

```
<cfset retVal = obj.Method1()>
```

2. If the method has one or more arguments, place the arguments in parentheses, separated by commas, as in the following example, which has one integer argument and one string argument:

```
<cfset x = 23>
<cfset retVal = obj.Method1(x, "a string literal")>
```

> **Note**
>
> When you invoke a Java method, the type of the data being used is important. For more information see *Java and ColdFusion data type conversions* below.

Calling JavaBean get and set methods

ColdFusion can automatically invoke get_PropertyName_() and set_PropertyName_(*value*) methods if a Java class conforms to the JavaBeans pattern. As a result, you can set or get the property by referencing it directly, without having to explicitly invoke a method.
For example, if the myFishTank class is a JavaBean, the following code returns the results of calling the getTotalFish() method on the myFish object:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cfoutput>
 There are currently #myFish.TotalFish# fish in the tank.
</cfoutput>
```

The following example adds one guppy to a myFish object by implicitly calling the setGuppyCount(int *number*) method:

```
<cfset myFish.GuppyCount = myFish.GuppyCount + 1>
```

Note

You can use the direct reference method to get or set values in some classes that have getProperty and setProperty methods but do not conform fully to the JavaBean pattern. However, you cannot use this technique for all classes that have getProperty and setProperty methods. For example, you cannot directly reference any of the following standard Java classes, or classes derived from them: Date, Boolean, Short, Integer, Long, Float, Double, Char, Byte, String, List, Array.

Calling nested objects

ColdFusion supports nested (scoped) object calls. For example, if an object method returns another object and you invoke a property or method on that object, you can use the following syntax:

```
<cfset prop = myObj.X.Property>.
```

Similarly, you can use code such as the following CFScript line:
GetPageContext().include("hello.jsp?name=Bobby");
In this code, the ColdFusion GetPageContext function returns a Java PageContext object, and the line invokes the `include` method of the PageContext object.

Creating and using a simple Java class

Java is a strongly typed language, unlike ColdFusion, which does not enforce data types. As a result, some subtle considerations exist when calling Java methods.

The Employee class

The Employee class has four data members: FirstName and LastName are public, and Salary and JobGrade are private. The Employee class has three overloaded constructors and an overloaded SetJobGrade method. Save the following Java source code in the file Employee.java, compile it, and place the resulting Employee.class file in a directory that is specified in the classpath:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
public class Employee {

public String FirstName;
public String LastName;
private float Salary;
private int JobGrade;

public Employee() {
 FirstName ="";
 LastName ="";
 Salary = 0.0f;
 JobGrade = 0;
}

public Employee(String First, String Last) {
 FirstName = First;
 LastName = Last;
 Salary = 0.0f;
 JobGrade = 0;
}

public Employee(String First, String Last, float salary, int grade) {
 FirstName = First;
 LastName = Last;
 Salary = salary;
 JobGrade = grade;
}

public void SetSalary(float Dollars) {
 Salary = Dollars;
}

public float GetSalary() {
 return Salary;
}

public void SetJobGrade(int grade) {
 JobGrade = grade;
}

public void SetJobGrade(String Grade) {
 if (Grade.equals("CEO")) {
  JobGrade = 3;
 }
 else if (Grade.equals("MANAGER")) {
  JobGrade = 2;
 }
 else if (Grade.equals("DEVELOPER")) {
  JobGrade = 1;
 }
}

public int GetJobGrade() {
 return JobGrade;
}

}
```

© 2014 Adobe Systems Incorporated. All rights reserved.

A CFML page that uses the Employee class

Save the following text as JEmployee.cfm:

```
<html>
<body>
<cfobject action="create" type="java" class="Employee" name="emp">
<!--- <cfset emp.init()> --->
<cfset emp.firstname="john">
<cfset emp.lastname="doe">
<cfset firstname=emp.firstname>
<cfset lastname=emp.lastname>
</body>

<cfoutput>
 Employee name is #firstname# #lastname#
</cfoutput>
</html>
```

When you view the page in your browser, you get the following output: Employee name is john doe

Reviewing the code

The following table describes the CFML code and its function:

Code	Description
`<cfobject action=create type=java class=Employee name=emp>`	Loads the Employee Java class and gives it an object name of emp.
`<!--- <cfset emp.init()> --->`	Does not call a constructor. ColdFusion calls the default constructor when it first uses the class; in this case, when it processes the next line.
`<cfset emp.firstname="john">` `<cfset emp.lastname="doe">`	Sets the public fields in the emp object to your values.
`<cfset firstname=emp.firstname>` `<cfset lastname=emp.lastname>`	Gets the field values back from emp object.

© 2014 Adobe Systems Incorporated. All rights reserved.

`<cfoutput>`
`Employee name is #firstname#`
`#lastname#`
`</cfoutput>`
	Displays the retrieved values.

Java considerations

The following points are important when you write a ColdFusion page that uses a Java class object:

- The Java class name is case sensitive. Ensure that the Java code and the CFML code use Employee as the class name.
- Although Java method and field names are case sensitive, ColdFusion variables are not case sensitive, and ColdFusion does any necessary case conversions. As a result, the sample code works even though the CFML uses emp.firstname and emp.lastname; the Java source code uses FirstName and LastName for these fields.
- If you do not call the constructor (or, as in this example, comment it out), ColdFusion automatically runs the default constructor when it first uses the class.

Using an alternate constructor

The following ColdFusion page explicitly calls one of the alternate constructors for the Employee object:

```
<html>
<body>

<cfobject action="create" type="java" class="Employee" name="emp">
<cfset emp.init("John", "Doe", 100000.00, 10)>
<cfset firstname=emp.firstname>
<cfset lastname=emp.lastname>
<cfset salary=emp.GetSalary()>
<cfset grade=emp.GetJobGrade()>

<cfoutput>
 Employee name is #firstname# #lastname#<br>
 Employee salary #DollarFormat(Salary)#<br>
 Employee Job Grade #grade#
</cfoutput>

</body>
</html>
```

In this example, the constructor takes four arguments: the first two are strings, the third is a float, and the fourth is an integer.

Java and ColdFusion data type conversions

ColdFusion does not use explicit types for variables, while Java is strongly typed. However, ColdFusion data does use underlying Java types to represent data.
Under most situations, when the method names are not ambiguous, ColdFusion can determine the data types that a Java object requires, and often it can convert ColdFusion data to the required types. For example, ColdFusion text

© 2014 Adobe Systems Incorporated. All rights reserved.

strings are implicitly converted to the Java String type. Similarly, if a Java object contains a doIt method that expects a parameter of type int, and CFML is issuing a doIt call with a CFML variable x that contains an integer value, ColdFusion converts the variable x to Java int type. However, ambiguous situations can result from Java method overloading, where a class has multiple implementations of the same method that differ only in their parameter types.

Default data type conversion

Whenever possible, ColdFusion automatically matches Java types to ColdFusion types.
The following table lists how ColdFusion converts ColdFusion data values to Java data types when passing arguments. The left column represents the underlying ColdFusion representation of its data. The right column indicates the Java data types into which ColdFusion can automatically convert the data:

CFML	Java
Integer	short, int, long (short and int can result in a loss of precision).
Real number	float double (float can result in a loss of precision.
Boolean	boolean
Date-time	java.util.Date
String, including lists	Stringshort, int, long, float, double, java.util.Date, when a CFML string represents a number or date.boolean, for strings with the value Yes, No, True, and False (case-insensitive).
Array	java.util.Vector (ColdFusion Arrays are internally represented using an instance of a java.util.Vector object.)ColdFusion can also map a CFML array to any of the following when the CFML array contains consistent data of a type that can be converted to the data type of the Java arr: byte[], char[], boolean[], int[], long[], float[], double[], String[], or Object[]. When a CFML array contains data of different of types, the conversion to a simple array type could fail.
Structure	java.util.Map
Query object	java.util.Map
XML document object	Not supported.
ColdFusion component	Not applicable.

The following table lists how ColdFusion converts data returned by Java methods to ColdFusion data types:

Java	CFML

© 2014 Adobe Systems Incorporated. All rights reserved.

boolean/Boolean	Boolean
byte/Byte	String
char/Char	String
short/Short	Integer
int/Integer	Integer
long/Long	Integer
float/Float	Real Number
double/Double	Real Number
String	String
java.util.Date	Date-time
java.util.List	Comma-delimited list
byte[]	Array
char[]	Array
boolean[]	Array
String[]	Array
java.util.Vector	Array
java.util.Map	Structure

Resolving ambiguous data types with the JavaCast function

You can overload Java methods so a class can have several identically named methods. At runtime, the JVM resolves the specific method to use based on the parameters passed in the call and their types.
In the section *The Employee class*, the Employee class has two implementations for the SetJobGrade method. One method takes a string variable, the other an integer. If you write code such as the following, which implementation to use is ambiguous:

```
<cfset emp.SetJobGrade("1")>
```

The "1" could be interpreted as a string or as a number, so no way exists to know which method implementation to use. When ColdFusion encounters such an ambiguity, it throws a user exception.
The ColdFusion JavaCast function helps you resolve such issues by specifying the Java type of a variable, as in the

© 2014 Adobe Systems Incorporated. All rights reserved.

following line:

```
<cfset emp.SetJobGrade(JavaCast("int", "1"))>
```

The `JavaCast` function takes two parameters: a string representing the Java data type, and the variable whose type you are setting. You can specify the following Java data types: boolean, int, long, float, double, and String. For more information about the `JavaCast` function, see the *CFML Reference*.

Handling Java exceptions

You handle Java exceptions just as you handle standard ColdFusion exceptions, with the cftry and cfcatch tags. You specify the name of the exception class in the `cfcatch` tag that handles the exception. For example, if a Java object throws an exception named myException, you specify myException in the `cfcatch` tag.

> **Note**
>
> To catch any exception generated by a Java object, specify java.lang.Exception for the `cfcatch` `type` attribute. To catch any Throwable errors, specify java.lang.Throwable in the `cfcatch` tag `type` attribute.

For more information on exception handling in ColdFusion, see Handling Errors.

Example: exception-throwing class

The following Java code defines the testException class that throws a sample exception. It also defines a myException class that extends the Java built-in Exception class and includes a method for getting an error message.
The myException class has the following code. It throws an exception with a message that is passed to it, or if no argument is passed, it throws a canned exception.

```
//class myException
public class myException extends Exception
{
 public myException(String msg) {
  super(msg);
 }
 public myException() {
  super("Error Message from myException");
 }
}
```

The testException class contains one method, doException, which throws a myException error with an error message, as follows:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
public class testException {
 public testException ()
 {
 }
 public void doException() throws myException {
  throw new myException("Throwing an exception from testException class");
 }
}
```

Example: CFML Java exception handling code

The following CFML code calls the testException class doException method. The `cfcatch` block handles the resulting exception.

```
<cfobject action=create type=java class=testException name=Obj>
<cftry>
 <cfset Obj.doException() >
 <cfcatch type="myException">
  <cfoutput>
   <br>The exception message is: #cfcatch.Message#<br>
  </cfoutput>
 </cfcatch>
</cftry>
```

Examples: using Java with CFML

The following examples show several examples of using Java objects in CFML. They include examples of using a custom Java class, a standard Java API class in a user-defined function, a JavaBean, and an Enterprise JavaBean (EJB).

Using a Java API in a UDF

The following example of a user-defined function (UDF) is functionally identical to the `GetHostAddress` function from the NetLib library of UDFs from the Common Function Library Project, www.cflib.org. It uses the `InetAddress` class from the standard Java 2 java.net package to get the Internet address of a specified host:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cfscript>
function GetHostAddress(host) {
 // Define the function local variables.
 var iaddrClass="";
 var address="";
 // Initialize the Java class.
 iaddrClass=CreateObject("java", "java.net.InetAddress");
 // Get the address object.
 address=iaddrClass.getByName(host);
 // Return the address
 return address.getHostAddress();
}
</cfscript>
<cfoutput>#gethostaddress("adobe.com")#</cfoutput>
```

Using an EJB

ColdFusion can use EJBs that JRun 4.0 servers provide. The JRun server jrun.jar file must have the same version as the jrun.jar file in ColdFusion.
To call an EJB, you use `cfobject` to create and call the appropriate objects. Before you use an EJB, do the following:

1. Have a properly deployed EJB running on a J2EE server. The bean must be registered with the JNDI server.
2. Have the following information:
   - Name of the EJB server
   - Port number of the JNDI naming service on the EJB server
   - Name of the EJB, as registered with the naming service
3. Install the EJB home and component interface compiled classes on your ColdFusion web server, either as class files in the *web_root*/WEB-INF/classes directory or packaged in a JAR file the *web_root*/WEB-INF/lib directory.

> **Note**
>
> To use an EJB served by a JRUN server, your ColdFusion installation and the JRun server that hosts the EJB must have the same version of the jrun.jar file (located in cf_root\runtime\lib directory in ColdFusion).

Although the specific steps for using an EJB depend on the EJB server and on the EJB itself, they generally correspond to the following order.

1. Use the `cfobject` tag to create an object of the JNDI naming context class (javax.naming.Context). You use fields from this class to define the information that you use to locate the EJB. Because you only use fields, you do not initialize the object.
2. Use the `cfobject` tag to create a java.util.Properties class object to contain the context object properties.
3. Call the `init` method to initialize the Properties object.
4. Set the Properties object to contain the properties that are required to create an initial JNDI naming context. These properties include the INITIAL_CONTEXT_FACTORY and PROVIDER_URL properties. You could also need to provide SECURITY_PRINCIPAL and SECURITY_CREDENTIALS values required for secure access to the naming context. For more information on these properties, see the JNDI documentation.
5. Use the `cfobject` tag to create the JNDI InitialContext (javax.naming. InitialContext) object.
6. Call the `init` method for the InitialContext object with the Properties object values to initialize the object.

© 2014 Adobe Systems Incorporated. All rights reserved.

7. Call the `lookup` method of the InitialContextext object to get a reference to the home interface for the bean that you want. Specify the JNDI name of the bean as the `lookup` argument.
8. Call the `create` method of the bean home object to create an instance of the bean. If you are using Entity beans, you typically use a finder method instead. A finder method locates one or more existing entity beans.
9. Now you can use the bean methods as required by your application.
10. When finished, call the `close` method of the context object to close the object.
    The following code shows this process using a simple Java Entity bean on a JRun 4.0 server. It calls the `getMessage` method of the bean to obtain a message.

```
<html>
<head>
 <title>cfobject Test</title>
</head>

<body>
<H1>cfobject Test</H1>
<!--- Create the Context object to get at the static fields. --->
<CFOBJECT
 action=create
 name=ctx
 type="JAVA"
 class="javax.naming.Context">

<!--- Create the Properties object and call an explicit constructor--->
<CFOBJECT
 action=create
 name=prop
 type="JAVA"
 class="java.util.Properties">

<!--- Call the init method (provided by cfobject)
  to invoke the Properties object constructor. --->
<cfset prop.init()>

<!--- Specify the properties These are required for a remote server only --->
<cfset prop.put(ctx.INITIAL_CONTEXT_FACTORY, "jrun.naming.JRunContextFactory")>
<cfset prop.put(ctx.PROVIDER_URL, "localhost:2908")>
<!--- <cfset prop.put(ctx.SECURITY_PRINCIPAL, "admin")>
  <cfset prop.put(ctx.SECURITY_CREDENTIALS, "admin")>
 --->
<!--- Create the InitialContext --->
<CFOBJECT
 action=create
 name=initContext
 type="JAVA"
 class="javax.naming.InitialContext">

<!--- Call the init method (provided through cfobject)
  to pass the properties to the InitialContext constructor. --->
<cfset initContext.init(prop)>

<!--- Get reference to home object. --->
<cfset home = initContext.lookup("SimpleBean")>

<!--- Create new instance of entity bean.
  (hard-wired account number). Alternatively,
```

© 2014 Adobe Systems Incorporated. All rights reserved.

```
  you would use a find method to locate an existing entity bean. --->
<cfset mySimple = home.create()>

<!--- Call a method in the entity bean. --->
<cfset myMessage = mySimple.getMessage()>

<cfoutput>
 #myMessage#<br>
</cfoutput>

<!--- Close the context. --->
<cfset initContext.close()>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

```
</body>
</html>
```

Using a custom Java class

The following code provides a more complex custom class than in the example *Creating and using a simple Java class*. The Example class manipulates integer, float, array, Boolean, and Example object types.

The Example class

The following Java code defines the Example class. The Java class Example has one public integer member, `mPublicInt`. Its constructor initializes `mPublicInt` to 0 or an integer argument. The class has the following public methods:

Method	Description
ReverseString	Reverses the order of a string.
ReverseStringArray	Reverses the order of elements in an array of strings.
Add	Overloaded: Adds and returns two integers or floats or adds the `mPublicInt` members of two Example class objects and returns an Example class object.
SumArray	Returns the sum of the elements in an integer array.
SumObjArray	Adds the values of the `mPublicInt` members of an array of Example class objects and returns an Example class object.
ReverseArray	Reverses the order of an array of integers.
Flip	Switches a Boolean value.

```
public class Example {
 public int mPublicInt;

 public Example() {
  mPublicInt = 0;
 }

 public Example(int IntVal) {
  mPublicInt = IntVal;
 }

 public String ReverseString(String s) {
  StringBuffer buffer = new StringBuffer(s);
  return new String(buffer.reverse());
 }

 public String[] ReverseStringArray(String [] arr) {
```

© 2014 Adobe Systems Incorporated. All rights reserved.

```
 String[] ret = new String[arr.length];
 for (int i=0; i < arr.length; i++) {
  ret[arr.length-i-1]=arr[i];
 }
 return ret;
}

public int Add(int a, int b) {
 return (a+b);
}

public float Add(float a, float b) {
 return (a+b);
}

public Example Add(Example a, Example b) {
 return new Example(a.mPublicInt + b.mPublicInt);
}

static public int SumArray(int[] arr) {
 int sum=0;
 for (int i=0; i < arr.length; i++) {
  sum += arr[i];
 }
 return sum;
}

static public Example SumObjArray(Example[] arr) {
 Example sum= new Example();
 for (int i=0; i < arr.length; i++) {
  sum.mPublicInt += arr[i].mPublicInt;
 }
 return sum;
}

static public int[] ReverseArray(int[] arr) {
 int[] ret = new int[arr.length];
 for (int i=0; i < arr.length; i++) {
  ret[arr.length-i-1]=arr[i];
 }
 return ret;
}

static public boolean Flip(boolean val) {
 System.out.println("calling flipboolean");
 return val?false:true;
```

© 2014 Adobe Systems Incorporated. All rights reserved.

```
  }
 }
```

The useExample ColdFusion page

The following useExample.cfm page uses the `Example` class to manipulate numbers, strings, Booleans, and Example objects. The CFML JavaCast function ensures that CFML variables convert into the appropriate Java data types.

```
<html>
<head>
 <title>CFOBJECT and Java Example</title>
</head>
<body>

<!--- Create a reference to an Example object --->
<cfobject action=create type=java class=Example name=obj>
<!--- Create the object and initialize its public member to 5 --->
<cfset x=obj.init(JavaCast("int",5))>

<!--- Create an array and populate it with string values,
  then use the Java object to reverse them. --->
<cfset myarray=ArrayNew(1)>
<cfset myarray[1]="First">
<cfset myarray[2]="Second">
<cfset myarray[3]="Third">
<cfset ra=obj.ReverseStringArray(myarray)>

<!--- Display the results --->
<cfoutput>
 <br>
 original array element 1: #myarray[1]#<br>
 original array element 2: #myarray[2]#<br>
 original array element 3: #myarray[3]#<br>
 after reverseelement 1: #ra[1]#<br>
 after reverseelement 2: #ra[2]#<br>
 after reverseelement 3: #ra[3]#<br>
 <br>
</cfoutput>


<!--- Use the Java object to flip a Boolean value, reverse a string,
  add two integers, and add two float numbers --->
<cfset c=obj.Flip(true)>
<cfset StringVal=obj.ReverseString("This is a test")>
<cfset IntVal=obj.Add(JavaCast("int",20),JavaCast("int",30))>
<cfset FloatVal=obj.Add(JavaCast("float",2.56),JavaCast("float",3.51))>

<!--- Display the results --->
<cfoutput>
 <br>
 StringVal: #StringVal#<br>
 IntVal: #IntVal#<br>
 FloatVal: #FloatVal#<br>
 <br>
</cfoutput>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<!--- Create a two-element array, sum its values,
  and reverse its elements --->
<cfset intarray=ArrayNew(1)>
<cfset intarray[1]=1>
<cfset intarray[2]=2>
<cfset IntVal=obj.sumarray(intarray)>
<cfset reversedarray=obj.ReverseArray(intarray)>

<!--- Display the results --->
<cfoutput>
 <br>
 IntVal1 :#IntVal#<br>
 array1: #reversedarray[1]#<br>
 array2: #reversedarray[2]#<br>
 <br>
</cfoutput><br>

<!--- Create a ColdFusion array containing two Example objects.
  Use the SumObjArray method to add the objects in the array
  Get the public member of the resulting object--->
<cfset oa=ArrayNew(1)>
<cfobject action=create type=java class=Example name=obj1>
<cfset VOID=obj1.init(JavaCast("int",5))>
<cfobject action=create type=java class=Example name=obj2>
<cfset VOID=obj2.init(JavaCast("int",10))>
<cfset oa[1] = obj1>
<cfset oa[2] = obj2>
<cfset result = obj.SumObjArray(oa)>
<cfset intval = result.mPublicInt>

<!--- Display the results --->
<cfoutput>
 <br>
 intval1: #intval#<br>
 <br>
</cfoutput><br>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

```
</body>
</html>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

Using Microsoft .NET Assemblies

You can use Adobe ColdFusion to call local or remote Microsoft .NET assembly class methods and access assembly fields. This documentation describes how to configure and run the ColdFusion .NET extension software and how to access and use .NET classes in your ColdFusion code. For information about .NET technology or how to develop .NET applications, see Microsoft .NET documentation.

© 2014 Adobe Systems Incorporated. All rights reserved.

About ColdFusion and .NET

ColdFusion lets you access and use Microsoft .NET assembly classes as CFML objects. CFML applications can use .NET assemblies in the following ways:

- Directly access and control Microsoft products, such as Word, Excel, or PowerPoint.
- Use existing .NET components.
- Use .NET assemblies that you create to leverage features that are difficult to use or not available in ColdFusion or Java. (Because ColdFusion is a J2EE application, if you cannot code a feature in CFML, it is more efficient to create it in Java than to use .NET.)
  The .NET classes that your application uses do not have to be local; your ColdFusion application can access .NET components that are located on remote systems, even systems that are located outside your firewall. Also, the ColdFusion system does not require .NET run-time software installed to use remote .NET components, so ColdFusion running on a UNIX, Linux, Solaris, or OS-X system can access and use .NET assemblies.
  You can use the `cfobject` tag or `CreateObject` function to create a reference to a .NET class object, by specifying either `.NET` or `dotnet` as the object type. You use the reference to access the .NET class fields and call the .NET class methods. This technique provides a tightly coupled, stateful, efficient method for accessing .NET classes from ColdFusion. As an alternative, your .NET application can make the class methods available as web services; however, using a web service is less reliable, has lower performance, and is less scalable than using ColdFusion objects for the .NET classes.

Note

.NET applications cannot access ColdFusion component functions directly. You can make the functions available as web services by specifying remote access. For more information on creating ColdFusion web services, see Using Web Services.

Because you use the .NET assembly classes the same way that you use any other ColdFusion object, you do not have to understand the details of .NET technology; you only have to understand how to use the specific .NET class that you are accessing. Code that uses a .NET method can be as simple as the following lines:

```
<cfobject type = ".NET" name = "mathInstance" class = "mathClass"
  assembly = "C:/Net/Assemblies/math.dll">
<cfset myVar = mathInstance.multiply(1,2)>
```

ColdFusion .NET access has the following additional features:

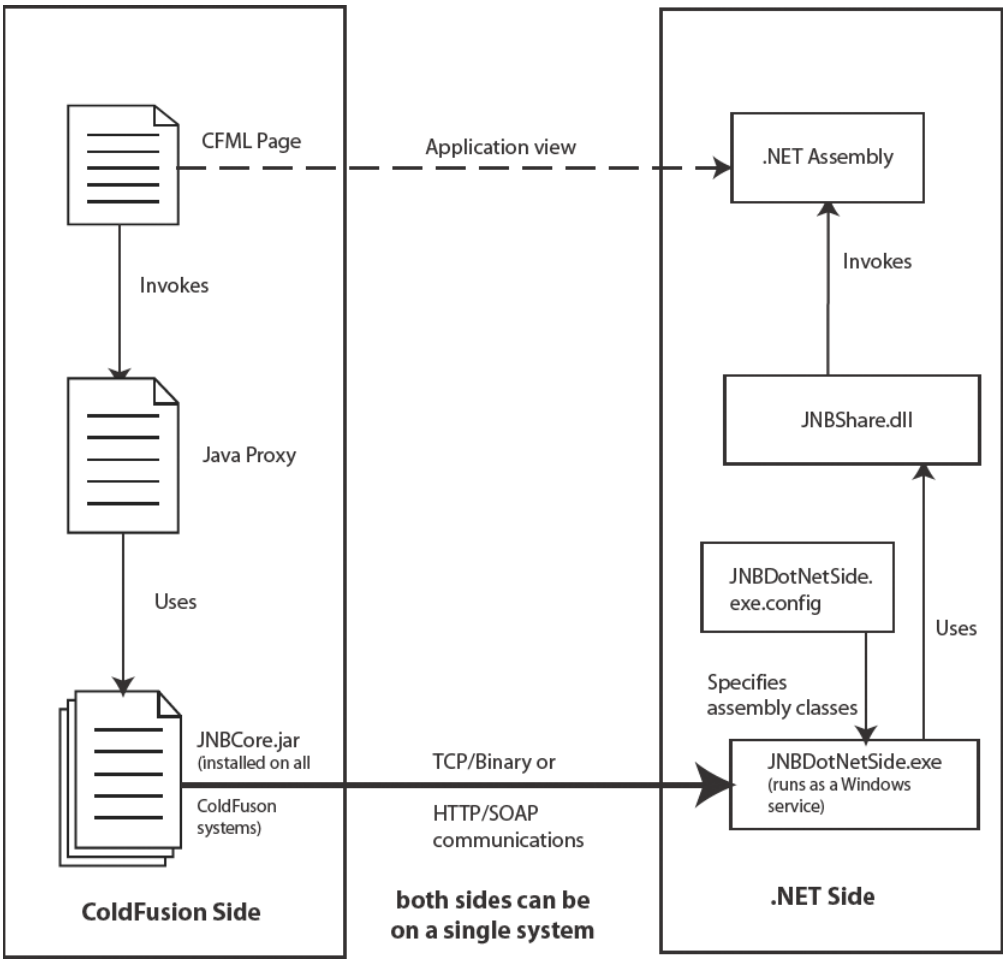
- If you make a change in the .NET assembly, ColdFusion automatically recognizes the change and uses that version for the next invocation.
- Your application can access .NET assemblies running on multiple machines.
- You can secure the communication between ColdFusion and .NET by using SSL.
- Primitive data types are automatically mapped between ColdFusion and .NET data types.

How .NET access works

For ColdFusion to access .NET assemblies, ColdFusion .NET extension software must run on the system that hosts the assemblies. A ColdFusion system that accesses only remote assemblies does not require the .NET extension. The .NET extension software provides the .NET-side connectivity features that enable access to .NET assemblies, including a .NET-side agent (which normally runs as the ColdFusion .NET service) that listens for and handles requests from the ColdFusion system.
On the ColdFusion system, the ColdFusion objects use Java proxies that act as local representatives of the .NET

© 2014 Adobe Systems Incorporated. All rights reserved.

classes. These proxies use binary TCP or SOAP-based HTTP communication to access a .NET-side agent. The agent then uses a DLL to invoke the .NET assembly classes. This communication is required in all cases, even if ColdFusion and the .NET assemblies are on the same system.
The following image shows how CFML-to-.NET access works:



If your .NET assemblies are on the local system, ColdFusion automatically creates and manages all required proxies and configuration information. Ensure only that the .NET extension is installed on your system and that the ColdFusion .NET Service is running. You can use the `cfobject` tag or `CreateObject` function to access the assemblies without any additional steps.
If the assemblies are on a remote system, you install and use the ColdFusion .NET extension software on the .NET system to create Java proxies for the .NET classes, and then move or copy them to the ColdFusion system. Also edit the JNBDotNetSide.exe.config file on the remote system to specify the .NET classes you use. The .NET system requires the following .NET extension software:

- JNBDotNetSide.exe, the .NET-side agent that communicates with the ColdFusion system (normally run as the ColdFusion .NET service).
- JNBDotNetSide.exe.config, a configuration file that identifies the .NET assemblies that ColdFusion can access.
- jnbproxy.exe and jnbproxyGui.exe are command line and GUI-based programs that generate the Java proxies that represent the .NET assemblies.
- Additional support files, including JNBShare.dll, which invoke the .NET assembly classes.
  For information on installing the ColdFusion .NET extension, see *Installing ColdFusion* guide.

© 2014 Adobe Systems Incorporated. All rights reserved.

Note

When you install a new .NET version, reinstall the ColdFusion .NET extension.

© 2014 Adobe Systems Incorporated. All rights reserved.

Accessing .NET assemblies

ColdFusion provides two methods for accessing .NET assemblies:

- A local access method for .NET objects that are installed on the ColdFusion system
- A remote access method for .NET objects located on other systems.
  For both methods, install the ColdFusion .NET extension and run the ColdFusion .NET service on the system that hosts the assemblies. You need not install the extension or run the service on a ColdFusion system that accesses only remote assemblies. For information on installing the ColdFusion .NET extension, see *Installing ColdFusion* guide.

Accessing local assemblies

For local access, ColdFusion automatically generates and uses proxies for the required .NET assemblies when you first use the `cfobject` tag or `CreateObject` function. ColdFusion caches the proxies for future use, so it does not generate assembly proxies each time.
Usually when you are accessing local .NET assemblies, you do not have to override the default communication configuration settings. Sometimes you could have to specify these settings, however. If other software on your system uses the default 6086 port, for example, change the port number specification in the jnbridge\DotNetSide.exe.config file, and specify the changed port number in your `cfobject` tag or `CreateObject` tag. For information on changing the port number specification, see *Configuring the .NET-side system* below,
To use the local access method, use the `cfobject` tag or `CreateObject` function to create and access the proxy. You can use the resulting ColdFusion object to construct the .NET object, call the .NET object's methods, and access its fields. For detailed information on using .NET classes, see Using .NET classes.

Accessing remote assemblies

The remote access technique accesses .NET assemblies by using TCP or HTTP to communicate with a .NET-side agent on a remote system. You create proxy instances and call assembly methods as you do in the Local access method, but first configure the remote .NET-side agent and, in most cases, the proxy classes that represent the remote .NET classes.

Configure remote .NET access

1. On the remote system, install the ColdFusion .NET integration software and run the .NET-side agent (see *Inst alling ColdFusion* guide).
2. If the .NET assemblies reside only on the remote system, generate proxy JAR files on that system that represent the assemblies (see *Generating the Java proxy classes* below). Then copy or move the proxy files to the local system. If identical .NET assemblies also reside on the local system, you can skip this step.
3. Configure the .NET-side system for remote access (see *Configuring the .NET-side system* below).

Generating the Java proxy classes

The Java proxy generation code requires direct access to the .NET assemblies to generate the proxy classes. Therefore, if the system that runs your ColdFusion application does not have the assemblies installed, run a tool on the .NET-side system to create the Java proxies. ColdFusion installs two proxy generation programs, jnbproxyGui.exe and jnbproxy.exe in the jnbridge directory when you install the .NET services. The jnbproxyGui.exe program is a Windows user interface application, and the jnbproxy.exe program is a command line application. Both programs have identical capabilities.

© 2014 Adobe Systems Incorporated. All rights reserved.

> **Note**
>
> If the system running the ColdFusion application has the assemblies installed, but must access remote versions of the assemblies (for example, because of configuration differences), you need not manually generate the proxy classes, and you can skip this step. Instead, specify the paths to the local .exe or .dll files in the `assembly` attribute of the `cfobject` tag (or `CreateObject` function) and specify the remote server in the `server` attribute. Configure the remote system for access, however.

On a ColdFusion system, the jnbproxyGui and jnbproxy programs are located in the *cfroot*\jnbridge directory. When you use the stand-alone installer, the programs are located in the *installDir*\jnbridge directory.
This document provides the basic information necessary to generate a proxy JAR file using the jnbproxyGui tool. Additional information is available in the following locations:

- The jnbridge directory includes a jnbproxy.chm Windows Help file with more complete documentation on the JNBridge technology that powers the ColdFusion .NET feature, including detailed information on both the jnbproxyGui and jnbproxy programs.
- The jnbridge\docs subdirectory includes additional documentation, including users guide.pdf, a PDF version of the information in the Help file.

> **Note**
>
> The JNBridge documentation includes information on features that are not supported in ColdFusion. ColdFusion, for example, does not support access from .NET assemblies to ColdFusion or memory-only communication.

Using the jnbproxyGui tool

You use the jnbproxyGui program to generate a proxy JAR file.

Generate and install a proxy JAR

1. Start JNBProxyGui.exe.
2. The first time you run the program, it displays the Enter Java Options dialog box. Configure the options, and click OK. You can change the configuration settings at a later time by selecting Project > Java Options.

On a system with ColdFusion: If ColdFusion is currently running on this system, ensure that the Start Java Automatically option, located on the right side of the JNBProxy Enter Java Options (Project > Java Options) dialog box is cleared. Leave the default values for the other settings.When you open an existing project, you could get a Restart Java Side pop-up window with the message "You must stop and restart the Java side before these changes to the classpath can take effect." You can ignore this message and click OK to proceed. When you start the program, the Java Options dialog box could appear. You do not have to change anything; click OK or Cancel to open the Launch JNBProxy dialog box.In some cases, JNBProxyGui could behave as follows when the Start Java Automatically option is not selected.

On a system without ColdFusion: If ColdFusion is not currently running on the system, ensure that the following options, which are located on the right side of the interface, are set. Leave the default values for the other settings.

- Ensure that the Start Java Automatically option is selected.
- Specify the java.exe file to use to compile the JAR file. You can use a Java 1.4 or 1.5 (J2SE 5.0) version of this file.
- Specify the jnbcore.jar file. The ColdFusion server installer places this file in the *cfroot*\lib directory. The J2EE installer places the file in the _cf_webapp_root_WEB-INF\cfusion\lib directory.
- Specify the bcel.jar file. The ColdFusion server installer places this file in the *cfroot*\lib directory. The

© 2014 Adobe Systems Incorporated. All rights reserved.

J2EE installer places the file in the _cf_webapp_root_WEB-INF\cfusion\lib directory.

2. In the Launch JNBProxy dialog box, select Create New Java > .NET Project, and click OK.
3. In the main Java proxy generation interface, set up and build a project:
   a. If you have not already done so, add the directory that contains your assembly files to the JNBProxy your project. Select Project >Edit Assembly List. In the Assembly List dialog box, click the Add button. In the New Assembly List Element dialog box, navigate to the directory that contains your assemblies. Select the directory (or directories) in the tree, and click OK. Then click OK in the Edit Assembly List dialog box.
   b. Open the Locate Assembly File dialog box (Project > Add Classes From Assembly File) and navigate to the directory that you added to the assembly list in step a. Select the assembly file or files that contain classes that require proxies and click OK.
   c. The classes in the selected file, and other .NET core classes on which they depend, appear in the Environment pane. Select all classes for which you want proxies in your JAR file, and click the Add+ button to add the selected classes and all supporting classes.
   d. In the Exposed Proxies list, select the classes to include in the JAR file. Normally, select all the listed classes, which ensures that all required classes are included.
   e. Select Project > Build from the main menu. In the Save Generated Proxies dialog box, specify the location and JAR file in which to save the generated proxies, and click Save.
   f. After the project is built, select File > Save Project and specify the file in which to save your project. The next time you run the jnbproxyGui program, you can select your project and reuse your previous settings, including the Assembly List.
4. Copy the JAR file to a directory on your ColdFusion system. You specify this path in the `cfobject` tag `assembly` attribute.

Supporting classes

JNBProxy can generate proxies not only for the .NET classes that are explicitly listed, but also for_ supporting classes_. A supporting class for a given .NET class is any class that could be needed as a direct or indirect result of using that .NET class. For a given .NET class, supporting classes include all of the following:

- The class.
- The class's superclass or superinterface (if it exists) and all of its supporting classes.
- The class's implemented interfaces (if any) and all of their supporting classes.
- For each field in the class:
  - The field's class and all of its supporting classes.
  - For each of the field's index parameters, the parameter's class and all of its supporting classes.
- For each method in the class:
  - The method's return value's class (if any) and all of its supporting classes.
  - For each of the method's parameters, the parameter's class and all of its supporting classes.
- For each constructor in the class, for each of the constructor's parameters, the parameter's class and all of its supporting classes.

  Unlike Java, where supporting classes include exceptions that methods throw, .NET supporting classes don't include thrown exceptions, because they are not declared in advance.

  The number of supporting classes depends on the classes explicitly listed, but it often can be 200-250 classes. Usually you generate all supporting classes. However, to save time or space, you can generate only those classes explicitly specified, without supporting classes.

  If a proxy for a supporting class has not been generated, and a proxy for such a class is later needed when the proxies are used, the proxy for the nearest superclass to the required class is used instead. If that proxy hasn't been generated, the proxy for the superclass of that superclass is used if it has been generated, and so forth, until the proxy for System.Object (which is always generated) is encountered. Thus, even with an incomplete set of proxies, code executes, although functionality and other information could be lost.

  In the jnbproxyGui tool, when you click the Add button, the list includes only the explicitly listed classes. When

© 2014 Adobe Systems Incorporated. All rights reserved.

you click the Add+ button, the list also includes the supporting classes. In the jnbproxy command line program, the default command generates proxies for the supporting classes; use the `/ns` option to override this default.

Configuring the .NET-side system

To configure the .NET-side system, you edit the jnbridge\JNBDotNetSide.exe.config configuration file in the following ways:

- For local assemblies, edit this file only if you do not use the default port, or if you use SSL security.
- For a .NET assembly on a remote machine, register the assemblies in this file to make it accessible to ColdFusion.

Edit the configuration file

1. Ensure that the following lines are in the <configSections> subsection of the <configuration> section:

```
<jnbridge>
 <javaToDotNetConfig scheme="Protocol" port="local port number"
  useSSL="true|false" certificateLocation="server certificate path"/>
</jnbridge>
```

   - The `scheme` attribute specifies the communications protocol, and must be `jtcp` or `http`.
   - The port number is the port of the .NET-side agent, normally 6086.
   - The `useSSL` attribute specifies whether to use SSL for secure communications. The attribute is optional; the default is to not use SSL.
   - The `certificateLocation` attribute specifies the location of the server SSL certificate. It is required only if the `useSSL` attribute is `true`.
     These settings must be the same as the corresponding attributes in your `cfobject` tag.
2. If the .NET assemblies are on a remote system, specify the assemblies that ColdFusion accesses by adding the following elements inside the <jnbridge> section.

```
<assemblyList>
 <assembly file="path to assembly or fully qualified name"/>
 ...
</assemblyList>
```

3. Stop and restart the .NET-side agent, if it is running. For example, on a ColdFusion system, restart the ColdFusion .NET Service. Your ColdFusion application can now access the .NET classes that you configured.
   The following example is a bare-bones JNBDotNetSide.exe.config file that specifies a .NET-side TCP server configuration. The server communicates by using TCP binary mode and listens on port 6086. Java clients can access \\\\x

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
 <sectionGroup name="jnbridge">
  <section name="dotNetToJavaConfig"
    type="System.Configuration.SingleTagSectionHandler"/>
  <section name="javaToDotNetConfig"
    type="System.Configuration.SingleTagSectionHandler"/>
  <section name="tcpNoDelay"
    type="System.Configuration.SingleTagSectionHandler"/>
  <section name="javaSideDeclarations"
    type="System.Configuration.NameValueSectionHandler"/>
  <section name="assemblyList"
    type="com.jnbridge.jnbcore.AssemblyListHandler, JNBShare"/>
 </sectionGroup>
 <jnbridge>
  <javaToDotNetConfig scheme="jtcp" port="6086"/>
  <assemblyList>
   \\x
   <assembly file="System.Windows.Forms, Version=1.0.5000.0,
    Culture=neutral, PublicKeyToken=b77a5c561934e089"/>
  </assemblyList>
 </jnbridge>
</configuration>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

Using .NET classes

You use .NET assembly classes the same way you use Java and other objects that you create using the cfobject tag or CreateObject function. In the simplest case, your application code only has to use the following format to include a local .NET class method:

```
<cfobject type = ".NET" name = "mathInstance" class = "mathClass"
  assembly = "C:/Net/Assemblies/math.dll">
<cfset myVar = mathInstance.multiply(1,2)>
```

Using CFScript and the `CreateObject` function, you can do the following:

```
<cfscript>
 mathInstance = CreateObject(".NET", "mathClass",
  "C:/Net/Assemblies/math.dll");
 myVar = mathInstance.multiply(1,2);
</cfscript>
```

Note

You cannot load two DLLs with same fully qualified name. ColdFusion always uses the first DLL that it accesses until the server is restarted. For example, if page1.cfm uses c:\dev\a.dll and page2.cfm uses c:\dev2\a.dll, and both DLLs have the same fully qualified name, the first DLL file to be loaded remains loaded, and both CFML pages use it.

When you create objects and access class methods and fields, and convert data types between ColdFusion and .NET, be aware of the following considerations and limitations:

- Data type conversion considerations described in *Converting between .NET and ColdFusion data types* below.
- Limitations described in the "Limitations" section of [cfobject: .NET object] in the *CFML Reference*.

Instantiating objects and calling class constructors

When you use the `cfobject` tag to create a .NET object, ColdFusion does not create an instance of the object. ColdFusion creates the object instance in either of the following cases:

- If the class has a default constructor, ColdFusion automatically calls the constructor when you first invoke a non-static method of the object.
- If the class does not have a default constructor, or if the class has multiple constructors and you do not want to use the default, call the special `init` method of the ColdFusion object. The `cfobject` tag{{}} automatically creates `init` methods for all class constructors. Using the `init`method causes ColdFusion to call the class constructor with the corresponding number and types of parameters. For example, the following tags cause ColdFusion to call the MyClass constructor that takes two integer parameters:

```
<cfobject type=".NET" name="myObj" class="com.myCo.MyClass"
  assembly="c:\assemblies\myLib.dll">
<cfset myObj.init(10, 5)>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

> **Note**
>
> ColdFusion does not create instances of objects if you use only their static methods.

Calling methods

You call .NET methods in the same way that you use any other ColdFusion object methods. For example, if the MyClass class has a `getName` method that takes a numeric ID and returns a name, you would call the method as follows:

```
<cfset theID="2343">
<cfset userName=mObj.getName(theID)>
```

Getting and setting fields

You can access and change public fields of any .NET class by calling the following methods:

```
Get_fieldName()
Set_fieldName(value)
```

For example, if the .NET class has a public field named `accountID`, you can access and change its value by using the `Get_accountID()` and `Set_accountID()` methods, as follows:

```
<cfobject type=".NET" class="com.myCo.MyClass"
  assembly="c:\assemblies\myLib.dll" name="myObj">
<cfset theAccount=myObj.Get_accountID()>
<cfset myObj.Set_accountID(theAccount + 1)>
```

You can access, but not modify final fields, so you can only call `Get_`*fieldName*`()` for these fields.

Converting between .NET and ColdFusion data types

Accessing .NET classes requires a Java proxy on the ColdFusion system and .NET code on the target system, so data must be converted among ColdFusion, Java, and .NET (to be exact, Microsoft Intermediate Language, or MSIL) data types. ColdFusion converts data types automatically. Usually, you do not have to take any special steps to ensure correct conversion. Some conversion limitations exist, and in some cases you must explicitly specify a data type when you call a method in a .NET proxy object.
The following paragraphs describe data conversion issues and how to handle them. For a detailed specification of how ColdFusion converts among ColdFusion data, Java data types, and .NET data types, see [cfobject: .NET object] in the *CFML Reference*.

Data type conversion rules and techniques

ColdFusion converts data automatically among ColdFusion, Java, and CLR data types. The following table indicates how ColdFusion converts among .NET Common Language Runtime (CLR) primitive and standard data types, the Java data types used in the proxies to represent CLR data types, and ColdFusion data types in your CFML application.

© 2014 Adobe Systems Incorporated. All rights reserved.

.NET type	Java type	ColdFusion type
sbyte	byte	Integer
byte	short	Integer
short	short	Integer
ushort	int	Integer
int	int	Integer
uint	long	Number
char	char	Integer or string
long	long	Number
ulong	float	Number
float	float	Number
double	double	NumberThe returned number retains greater precision than is normally displayed in ColdFusion. Use the PrecisionEvaluate function to access and display the full precision of a returned double value. You can also pass a value with full double precision to a .NET method.
bool	boolean	Boolean
enum		Not converted, but enumerator elements can be accessed directly by using the format Enumerator_variable.enumerator, as in MyColor.Red
array	array	Array
string	String	String
System.Collections.ArrayList	java.util.ArrayList	Array*Note:* ColdFusion converts from .NET type to ColdFusion type only, it does not convert ColdFusion Arrays to .NET ArrayLists.

© 2014 Adobe Systems Incorporated. All rights reserved.

System.Collections.Hashtable	java.util.Hashtable	Structure*Note:* ColdFusion converts from .NET type to ColdFusion type only, it does not convert ColdFusion Structures to .NET Hashtables
System.Data.DataTable		Query*Note:* ColdFusion converts from .NET type to ColdFusion type only, it does not convert ColdFusion Queries to .NET DataTables
System.DateTime	java.util.Date	Date/time
decimalSystem.Decimal	java.math.BigDecimal	String representation of the decimal number.For details on using decimal numbers, see *Using decimal numbers* below.
System.Object		If a .NET argument is of type System.Object, ColdFusion Strings are converted directly. Other types require using the `JavaCast` functio n.ColdFusion cannot convert System.object instances returned by .NET methods to ColdFusion types, but you can access them using the Object methods.For detailed information, see *Converting data to System.Object type* below.

Using decimal numbers

Use the `JavaCast` function to convert ColdFusion data into BigDecimal format before you pass the value to a .NET function, as in the following example:

```
<cfset netObj.netFunc(javacast("bigdecimal","439732984732048"))>
```

ColdFusion automatically converts returned decimal and System.Decimal values to ColdFusion string representations.

Ensuring decimal and date/time conversions

ColdFusion converts .NET decimal or System.Decimal types only if the proxy for System.Decimal is a value type proxy. Similarly, it converts .NET System.DateTime values to ColdFusion Date-time values only if the proxy for System.DateTime is a value type proxy. The ColdFusion server always uses value proxies when it generates these proxies. If you use the JNBProxyGUI.exe tool to generate the proxy, however, make sure to generate the proxy for System.Decimal as value type.

Converting data to System.Object type

© 2014 Adobe Systems Incorporated. All rights reserved.

When a .NET method specifies `System.Object` (as opposed to a specific Object subclass, such as `System.Boolean`) as the argument type, and you want to pass primitive values as arguments to that method, use the `javacast` function to identify the data conversion. Once ColdFusion knows the data type, it automatically converts to the appropriate .NET type. Here is the table that describes the conversion rule from ColdFusion type to .NET type.

.NET Type	Type used in javacast
bool / System.Boolean	boolean
bool[] / System.Boolean[]	boolean[]
char / System.Char	char
char[] / System.Char[]	char[]
double / System.Double	double
double[] / System.Double[]	double[]
float / System.Single	float
float[] / System.Single[]	float[]
int / System.Int32	int
int[] / System.Int32[]	int[]
long / System.Int64	long
long[] / System.Int64[]	long[]
sbyte / System.Sbyte	byte
sbyte []/ System.Sbyte[]	byte []
short / System.Int16	short
short[] / System.Int16[]	short[]
System.Decimal	bigdecimal
System.String	String

Note

You do not have to use a `JavaCast` function to convert ColdFusion string variables. They are automatically converted to .NET System.String.

© 2014 Adobe Systems Incorporated. All rights reserved.

Create special objects for .NET primitive unsigned data types, such as byte (unsigned byte), ushort (unsigned short), uint (unsigned int) and ulong (unsigned long), for which no corresponding java types exist. The following table lists the .NET primitive types and the corresponding class you must use.

.NET type	Class used in cfobject/createObject
byte / System.Byte	System.BoxedByte
ushort / System.UInt16	System.BoxedUShort
uint / System.UInt32	System.BoxedUInt
ulong / System.UInt64	System.BoxedULong

Use the `createObject` function or `cfobject` tag to create these special objects, in the same manner as you create other .NET classes, before you use them in your assignment statement. For example, the following line creates a ushort representation of the value 100:

```
<cfset boxedUShort = createObject(".NET". "System.BoxedUShort").init(100)>
```

The following example creates a System.Hashtable object and populates it with examples of all types of primitives.

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<!--- create a .NET Hashtable --->
<cfset table = createObject(".NET", "System.Collections.Hashtable")>

<!--- call HashTable.add(Object, Object) method for all primitives --->
<cfset table.add("shortVar", javacast("short", 10))>
<cfset table.add("sbyteVar", javacast("byte", 20))>
<cfset table.add("intVar", javacast("int", 123))>
<cfset table.add("longVar", javacast("long", 1234))>
<cfset table.add("floatVar", javacast("float", 123.4))>
<cfset table.add("doubleVar", javacast("double", 123.4))>
<cfset table.add("charVar", javacast("char", 'c'))>
<cfset table.add("booleanVar", javacast("boolean", "yes"))>
<cfset table.add("StringVar", "Hello World")>
<cfset table.add("decimalVar", javacast("bigdecimal", 123234234.505))>

<!--- call HashTable.add(Object, Object) for unsigned primitive types. --->
<cfset boxedByte = createObject(".NET", "System.BoxedByte").init(10)>
<cfset table.add("byteVar", boxedByte)>

<cfset boxedUShort = createObject(".NET", "System.BoxedUShort").init(100)>
<cfset table.add("ushortVar", boxedUShort)>

<cfset boxedUInt = createObject(".NET", "System.BoxedUInt").init(123)>
<cfset table.add("uintVar", boxedUInt)>

<cfset boxedULong = createObject(".NET", "System.BoxedULong").init(123123)>
<cfset table.add("ulongVar", boxedULong)>

<cfdump var="#DotNetToCFType(table)#">
```

Any other .NET objects can be passed as it is.

Handling ambiguous type conversions

ColdFusion cannot determine the correct data type conversion if a method has multiple signatures with the same number of parameters that differ only in the parameter data types. In this case, use the JavaCast method to convert the ColdFusion data to the Java type that corresponds to the .NET type.
For example, if a .NET class has methods myFunc(ulong) and myFunc(int), use the `JavaCast` method to convert your ColdFusion variable to the Java float or int data type, as the following line shows:

```
myFunc(JavaCast(int, MyVar));
```

Similarly, if a .NET class has methods myFunc(int) and myFunc(String), use the `JavaCast` method to convert your ColdFusion variable to the Java int or String data type, as shown in the following line:

```
myFunc(JavaCast(String, "123");
```

In some cases, the `JavaCast` function cannot eliminate ambiguity because a single Java type corresponds to

© 2014 Adobe Systems Incorporated. All rights reserved.

multiple .NET types. In these cases, ColdFusion creates a proxy with only one method, which uses the .NET data type that corresponds directly to a Java type.
For example, if the .NET class has methods myFunc(ulong) and myFunc(float), the generated proxy has only one method. This method calls myFunc(float), because the Java float type used to handle ColdFusion floating-point numbers corresponds directly to the .NET float type. In this case, you can never call the .NET myFunc(ulong) method.

Working with complex .NET data types

When you use complex .NET data such as Hashtable, ArrayList and DataTable, ColdFusion normally automatically converts the data to the corresponding ColdFusion data type: structure, array, and query, respectively. When you work with this data you take specific actions to enable the proper access and conversion of the data, as follows:

- Use associative array notation to properly access .NET Hashtable data from ColdFusion
- You cannot use ColdFusion variables directly in parameters that take Hashtable, ArrayList, or DataTable input.
- You can disable automatic conversion of complex .NET data to ColdFusion types.
- You can manually convert complex .NET data to ColdFusion types.

Using Hashtable data in ColdFusion

.NET Hashtables are case sensitive, but most methods of ColdFusion structure access are not case sensitive. Only associative array notation of the form *structName*`["`*keyName*`"]` is case sensitive. When .NET Hashtables are converted to CF structure, the entire data set is converted, even if the element keys differ only in case. Therefore, to get the values of the keys that differ only in case, use associative array notation.
The following example shows this issue. It creates a Hashtable object with three entries whose key values vary only in case. In the example, output using dot-delimited structure notation always returns the same value, corresponding to the all-uppercase key, but associative array notation returns the correct result.

```
<!--- Create a Hashtable and convert it to a ColdFusion structure. --->
<cfset table = createObject(".NET", "System.Collections.Hashtable")>
<cfset table.add("Key", "Value1")>
<cfset table.add("KEY", "Value2")>
<cfset table.add("key", "Value3")>
<cfset cftable = DotNetToCFType(table)>

<cfdump var="#cftable#">

<h3>Using dot notation</h3>
Key : <cfoutput>#cftable.Key#</cfoutput><br>
KEY : <cfoutput>#cftable.KEY#</cfoutput><br>
key : <cfoutput>#cftable.key#</cfoutput><br>
<p>

<h3>Using associative array notation</h3>
Key : <cfoutput>#cftable["Key"]#</cfoutput><br>
KEY : <cfoutput>#cftable["KEY"]#</cfoutput><br>
key : <cfoutput>#cftable["key"]#</cfoutput><br>
```

Using .Net ArrayList in ColdFusion

ColdFusion converts System.Collections.ArrayList objects to ColdFusion arrays, and you can perform all standard ColdFusion array operations on them. The following example shows this usage:
.Net Code:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
public ArrayList getList(){
 ArrayList myAL = new ArrayList();
 myAL.Add("Hello");
 myAL.Add(1);
 myAL.add(true);
 Return AL;
}
```

ColdFusion Code:

```
<cfset cflist = netObject.getList()>
<cfloop array="#cflist#" index="item">
 <cfoutput>#item#</cfoutput><br>
</cfloop>

<cfif cflist[3]>
  <cfoutput>Third element in the list is true</cfoutput>
</cfif>
```

Using ADO.Net DataTable in ColdFusion

ColdFusion converts System.Data.DataTable objects to ColdFusion query objects, and you can perform all standard ColdFusion query operations on them. The following example shows this usage:
.Net code:

```
public DataTable datasetMethod()
{
 //conn string
 string connectionString = "...";

 //connection
 using (SqlConnection connection = new SqlConnection(connectionString))
 {
  SqlCommand cmd = new SqlCommand(@"SELECT * FROM [tblEmployees]", connection);
  connection.Open();
  SqlDataReader reader = cmd.ExecuteReader();
  DataTable dt = new DataTable();
  dt.Load(reader);
  return dt;
}
}
```

ColdFusion code:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cfset query1 = netObject.datasetMethod()>
<cfoutput query="query1">
 Query1.CurrentRow = #query1.CurrentRow#<br>
</cfoutput>
```

Using ColdFusion complex types in .NET input parameters

When a .NET method returns an ArrayList, Hashtable, or DataTable, ColdFusion automatically converts it to a ColdFusion array, structure, or query, respectively. However ColdFusion does not automatically convert from ColdFusion data types to these .NET types. (ColdFusion does automatically convert ColdFusion arrays to .Net array types.) Therefore, you cannot use ColdFusion variables directly as input parameters to .NET object instance methods that require .NET System.Collection.ArrayList, System.Collection.Hashtable, or System.Data.DataTable types. Instead create instances of these .NET types and populate them with the required data before you pass them to the .NET method. For an example of creating and populating a System.Collection.Hashtable object, see the example at the end of the "Converting data to System.Object type" section.

Disabling automatic conversion of complex .NET data

You can disable automatic conversion of .NET `System.Collections.Hashtable`, `System.Collections.ArrayList` or S{{ystem.Data.DataTable}} objects to the corresponding ColdFusion structure, array, or query objects. You could want to disable the conversion under the following circumstances:

- If a collection or DataTable returned by a .NET method is large and you only want a small subset of the data. If auto conversion is enabled, ColdFusion creates a data structure with all the object's fields. Creating the structure could take significant time and resources, because ColdFusion must invoke .NET methods internally to get each of the fields. You can disable the automatic conversion and retrieve the fields or data from .NET objects like any other objects.
- If you invoke a .NET method that returns a complex variable, and then pass the variable to another .NET method as argument. If automatic conversion is enabled, you cannot pass the Hashtable object from the first method directly to the second method.
  To disable automatic conversion, set the JVM coldfusion.dotnet.disableautoconversion system property to true. For example, in a ColdFusion stand-alone server, or if you use JRun as your J2EE server, include the following setting in the JVM.config file:

```
-Dcoldfusion.dotnet.disableautoconversion=true
```

Manually converting complex .NET objects

Use the `DotNetToCFType` function to convert a `System.Collections.Hashtable`, `System.Collections.ArrayList` or S{{ystem.Data.DataTable}} object to a ColdFusion structure, array, or query respectively when either of the following circumstances are true:

- You have set the coldfusion.dotnet.disableautoconversion system property to true.
- Automatic conversion is enabled, you created the complex .NET object by using the `createObject` function or `cfobject` tag, and you want to convert this object into the corresponding ColdFusion representation.
  For an example of using the function, see DotNetToCFType in the *CFML Reference*.

Using .NET objects

.NET fields and return values with class types are available in ColdFusion as .NET objects. You can use the object's

© 2014 Adobe Systems Incorporated. All rights reserved.

methods to access object data and make it available to ColdFusion using supported data types.
The following example gets information about a system's drives. It calls the System.IO.DriveInfo.GetDrives() method to get an array of System.IO.DriveInfo objects, one per drive. It then calls the object methods to get specific information about the drives, and displays the information. The example uses a `cfdump` tag to simplify the code.

> **Note**
>
> The System.IO.DriveInfo is not included in the .NET 1.x framework. It is included in .NET 2.0 and later frameworks. For information on determining the .NET framework, see *Determining and changing the .NET version* in Advanced tools.

```
<!--- Create a query for the drive information results. --->
<cfset result=QueryNew("name,type,isready,format,label,totalsize,freespace"
     ,"varchar,varchar,bit,varchar,varchar,double,double")>
<!--- Create a .NET System.IO.DriveInfo object. --->
<cfobject type=".NET" name="sidiClass" class="System.IO.DriveInfo">
<!--- Get the drives. --->
<cfset drives=sidiClass.GetDrives()>
<!--- Loop through drives. --->
<cfloop from="1" to="#ArrayLen(drives)#" index="i">
 <!--- Add a row to the query.--->
 <cfset QueryAddRow(result)>
 <!--- Get the drive name, type, and ready flag. --->
 <cfset QuerySetCell(result, "name", drives[i].Get_Name())>
 <cfset QuerySetCell(result, "type",
    drives[i].Get_DriveType().ToString())>
 <cfset QuerySetCell(result, "isready", drives[i].Get_IsReady())>
 <!--- Get extra details ONLY if the drive is ready. --->
 <cfif drives[i].Get_IsReady()>
  <cfset QuerySetCell(result, "format", drives[i].Get_DriveFormat())>
  <cfset QuerySetCell(result, "label", drives[i].Get_VolumeLabel())>
  <cfset QuerySetCell(result, "totalsize", drives[i].Get_TotalSize())>
  <cfset QuerySetCell(result, "freespace",
     drives[i].Get_AvailableFreeSpace())>
 </cfif>
</cfloop>
<cfdump var="#result#">
```

© 2014 Adobe Systems Incorporated. All rights reserved.

.NET Interoperability Limitations

ColdFusion .NET interoperability has the following limitations

- You cannot invoke methods with pointers as arguments or the return type.
- You cannot invoke methods that take Out parameters.
- ColdFusion can only convert from System.Data.DataTable, System.Collection.Hashtable and System.Collection.ArrayList to ColdFusion data types. ColdFusion cannot convert from ColdFusion queries, structures, and arrays to these System data types; however, it can convert from ColdFusion arrays to the CLR array type. Therefore, you cannot pass structures or queries directly to .NET methods.
- You cannot access .NET user interface components.
- You cannot use callbacks (events and Delegates) from .NET side.
- ColdFusion cannot determine the correct data type conversion if a method has multiple signatures that have the same number of parameters and differ only in the parameter data types. In this case, use the JavaCast m ethod to convert the ColdFusion data to the Java type that corresponds to the .NET type.
- If the `JavaCast` function cannot eliminate ambiguity between functions with the same number of parameters because a single Java type corresponds to multiple .NET types, ColdFusion creates a single proxy that uses the .NET data type that corresponds directly to a Java type.For more information on how to ambiguous handle type conversions, see *Converting between .NET and ColdFusion data types* in Using .NET classes.
- Assemblies registered in the DotNetSide.exe.config file must have unique class names. If two or more assemblies have the same class name, method invocation can result in an error or can give the wrong result. For example, do not have two DLLs, a.dll and b.dll, that contain the same class name, nam1.name2.MyClass. If you use one DLL and later want to use another DLL that contains a class that clashes with first, restart the ColdFusion .NET Service if ColdFusion and .NET both are on the same machine. If they are on the different machines, remove the entry for the first DLL from the DotNetSide.exe.config file and restart the ColdFusion .NET Service on the Windows machine hosting the .NET service.

© 2014 Adobe Systems Incorporated. All rights reserved.

Example applications

The first application example uses a Microsoft .NET system class method directly. The second application example uses a custom C# class to access Microsoft Word.

Example: Using a .NET class directly

The following example uses the Microsoft .NET System.Net.NetworkInformation.Ping class method directly to ping servers. This class is supported in .NET version 2.0 and later.

```
<!--- This function pings the specified host. --->
<cffunction name="Ping" returntype="string" output="false">
 <cfargument name="host" type="string" required="yes">
 <!--- Local variables --->
 <cfset var pingClass="">
 <cfset var pingReply="">
 <!--- Get Ping class --->
 <cfobject type=".NET" name="pingClass"
   class="System.Net.NetworkInformation.Ping">
 <!--- Perform synchronous ping (using defaults) --->
 <cfset pingReply=pingClass.Send(Arguments.host)>
 <!--- Return result --->
 <cfreturn pingReply.Get_Status().ToString()>
</cffunction>

<h3>Ping Test</h3>
<cfoutput>
 127.0.0.1: #Ping("127.0.0.1")#<br>
 www.adobe.com: #Ping("www.adobe.com")#<br>
</cfoutput>
```

Example: Using a custom class to access Microsoft Word

The following ColdFusion application uses a custom C# WordCreator class, and supporting classes in Microsoft Office and Word DLLs, to create a Word document. The application opens Microsoft Word, writes five copies of the text specified by the someText variable, and saves the document in the file specified by the filename variable. The application leaves the instance of Word open.

> **Note**
>
> For an example that uses a .NET System class directly and does not require any cousin .NET code, see the "Limitations" section of [cfobject: .NET object] in the CFML Reference.

The second listing shows the WordCreator C# source code. To run this application locally, compile this class and the Microsoft Interop.Word.dll file, and place them in the C:\dotnet directory. (Alternatively, you can place them elsewhere and change the paths in the `cfobject assembly` attribute.) You could need additional or different Microsoft DLL files, depending on the version of Microsoft Office that you have installed.
The ColdFusion application contains the following code:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cfset filename="C:\dotNet\demo.doc">
<cfif fileexists(filename)>
 <cffile action="delete" file="#filename#">
</cfif>
<cfobject type=".NET"
assembly="C:\dotNetApp\WordApp.dll,C:\dotNet\Interop.Office.dll" name="wordCreator"
class="WordApp.WordCreator">
<cfset wordCreator.init("#filename#")>
<cfdump label="WordCreator Class Dump" var="#wordCreator#">

<cfset someText = "ColdFusion created this sample document using Windows .NET class
methods. The text is long enough to appear in the Word file on multiple lines.">

<cfloop from=1 to=5 index =i>
 <cfset wordCreator.addText(someText)>
 <cfset wordCreator.newParagraph()>
 <cfset wordCreator.newParagraph()>
 <cfset wordCreator.addText("Starting a new paragraph. It starts a
   a new line.")>
 <cfset wordCreator.newParagraph()>
 <cfset wordCreator.newParagraph()>
</cfloop>
<cfset wordCreator.save()>
```

The C# source for the WordCreator class is as follows:

```
using System;
using System.IO;
using System.Collections.Generic;
using System.Text;

// The commented-out lines could be needed on some systems in place of,
// or in addition to, the line that follows them.
// using Microsoft.Office.Core;
// using Word;
// using Microsoft.Office;
// using Microsoft.Office.Interop;
using Microsoft.Office.Interop.Word;
namespace WordApp {
 public class WordCreator {
  object readOnly = false;
  object isVisible = true;
  object missing = System.Reflection.Missing.Value;
  object docType = 0;
  object newTemplate = false;
  object template = "normal.dot";
  object format = WdSaveFormat.wdFormatDocument;
  ApplicationClass app = new ApplicationClass();
  private object fileName;
  private Document doc;
  private bool isNewDoc = false;

   public WordCreator(String fileName) {
   this.fileName = fileName;
```

© 2014 Adobe Systems Incorporated. All rights reserved.

```
    app.Visible = true;
    if (File.Exists(fileName))
     doc = app.Documents.Open(ref this.fileName, ref missing, ref
      readOnly, ref missing, ref missing, ref missing, ref missing,
      ref missing, ref missing, ref missing, ref missing, ref
      isVisible, ref missing, ref missing, ref missing, ref missing);
    else {
     doc = app.Documents.Add(ref template, ref newTemplate,
      ref docType, ref isVisible);
     isNewDoc = true;
    }
    doc.Activate();
   }

   public void addText(String text) {
    app.Selection.TypeText(text);
   }

   public void newParagraph() {
    app.Selection.TypeParagraph();
   }

   public void save() {
    if(!isNewDoc)
     doc.Save();
    else doc.SaveAs(ref fileName, ref format, ref missing, ref missing,
     ref missing, ref missing, ref missing, ref missing, ref missing,
     ref missing, ref missing, ref missing, ref missing, ref missing,
     ref missing, ref missing);
   }
```

© 2014 Adobe Systems Incorporated. All rights reserved.

```
  }
 }
```

© 2014 Adobe Systems Incorporated. All rights reserved.

Advanced tools

Occasionally, the use of additional tools for generating proxies and running the .NET extension software can be helpful in some workflows.

Using the jnbproxy command

You can use the `jnbproxy` command-line tool as an alternative to the `jnbproxyGui` program, to generate Java proxies. For more information, see *Generating the Java proxy classes* in Accessing .NET assemblies.
For example, you can use this command in a batch file to generate multiple proxy JAR files in a single operation.
The jnbproxy command has the following format:

```
jnbproxy options... classes...
```

For example:

```
jnbproxy /al C:\dotNet\netdll\PrimitiveTypes.dll /d C:\dotNet\MyJavajars
 /host localhost /n PrimitiveTypes /nj /pd j2n /port 6085 /pro b
 /pp C:\ColdFusion8\lib CSharpDatatypes.PrimitiveTypes
```

Options

The following table lists the options that you can use. To create proxies on a system that is running ColdFusion, use the /nj option and do not specify the `/bp`, `/java`, or `/jp` options.

Option	Req/Opt	Default	Description
/al *assemblylist*	Required		Specifies a semicolon-separated series of file paths of .NET assemblies (DLL and EXE files) that contain the required .NET classes.
/bp *bcelpath*	Optional	Use the CLASSPATH environment variable to locate the file.	Specifies the path to the folder that contains the bcel.jar file. Ignored if you use the /nj option.
/cf	Required		Use the ColdFusion software license. If you do not include this option, your proxies are limited to a 30-day trial period.

© 2014 Adobe Systems Incorporated. All rights reserved.

/d *directory*	Optional	The current execution directory.	Specifies the directory in which to write a JAR file with the generated proxies.
/f *classfile*	Optional		Reads the classes from the specified text file, not the command line. For more information, see the JNBridge documentation.
/h	Optional		Lists the options and usage information. Typing the command `jnbproxy` with no options or arguments results in the same information.
/host *hostname*	Required		Specifies the host on which the .NET code is located. This option can be a host name or an IP address. Normally, you specify `localhost`.
/java *javapath*	Optional	Use the first java.exe file found using the system PATH environment variable.	Specifies the path of the directory that contains the java.exe program to use when automatically starting Java. Ignored if you use the `/nj` option.
/jp *jnbcorepath*	Optional	Use, the CLASSPATH environment variable.	Specifies the path of the folder containing the file jnbcore.jar. Ignored if you use the `/nj` option.
/ls	Optional	Generate and list the proxies.	Lists all classes that are generated in support of the specified classes (see *Supporting classes* in Accessing .NET assemblies), but don't generate the proxies.

© 2014 Adobe Systems Incorporated. All rights reserved.

/n *name*	Optional	Create a file named jnbproxies.jar.	Specifies the name of the JAR file in which to place the proxies. Do not specify the .jar extension; the tool automatically adds it.
/nj	Optional	Start Java automatically.	Does not start Java automatically. If you use this option, Java must be running, and the `/bp`, `/java`, `/jp`, and `/wd` options, if present, are ignored.
/ns	Optional	Generate proxies for all supporting classes.	Generates proxies for the classes specified on the command line (or class file) only, not for any supporting classes.
/pd	Required		Specifies the direction in which the proxies operate. Must be `j2n`.
/port *portNum*	Required		Specifies the port on which the .NET side listens when generating the proxies. Must be an integer. Normally this value is 6085.
/pro *protocol*	Required		Specifies the communication mechanism between the .NET and Java sides. The valid values are: • `b`-TCP/binary • `h-`(HTTP/SOAP
/wd *dir*	optional	The system's default working directory.	Specifies the working directory for the JVM.Ignored if the `/nj` option is present.

Classes

A space-separated sequence of fully qualified .NET class names (for example, `CSharpDatatypes.PrimitiveTypes`) for which to generate proxies. The proxies for System.Object and System.Type are always generated, even if they are not listed in the class list.

© 2014 Adobe Systems Incorporated. All rights reserved.

Passing data by reference and value

The proxy generators let you specify whether to pass parameters and return values by reference or by value.

About passing by reference and value

When you pass data by reference, the information transferred between the Java Proxy and the .NET side is a logical pointer to the underlying .NET object, which continues to reside on the .NET side. When you pass data by value, the transferred information contains a copy of the contents of the .NET object, which could or continue to reside on the .NET side after a function call. Passing by reference and value have different advantages.When you pass data by reference, only changed values are passed between the Java proxy and the .NET object directly. All other information is passed as reference to its representation in the corresponding objects. Because the reference is typically much smaller than the actual object, passing by reference is typically fast. Also, because the reference points to a .NET object that continues to exist on the .NET side, if that .NET object is updated, the updates are immediately accessible to the proxy object on the Java side. The disadvantage of reference proxies is that any access to data in the underlying object (for example, field or method accesses) requires a round trip from the Java side to the .NET side (where the information resides) and back to the Java side.When you pass data by value, a copy of the data is passed between .NET and Java. Because the data object itself is typically bigger than a reference, passing an object by value takes longer than passing it by reference. Also, the value that is passed is a snapshot of the object taken at the time that it was passed. The passed object maintains no connection to the underlying .NET object, therefore, the passed value does not reflect any updates to the underlying .NET object that are made after the object is passed. The advantage of passing data by value proxies is that all data in the object is local to the Java side, and field accesses are fast, because they do not require a round trip to the .NET side and back to get the data.The choice of whether to use reference or value proxies depends on the desired semantics of the generated proxies, and on performance.

- In general, use reference proxies (the default), because they maintain the normal parameter-passing semantics of Java and C#.
- In general, use value proxies in any of the following cases:
  - The class functions always must pass parameter values and return values back and forth.
  - The class object contains little data.
  - The object data changes frequently, and the object is either relatively small or the frequency of accesses to data outweighs the time taken to transfer the object.

Specifying the data passing method

When you use the JNBProxy.gui tool to generate proxies, you can designate the proxies that pass by reference and which proxies pass by value. The default proxy type is reference.To set the data passing method for a class, right-click on the class in the Exposed Proxies pane. Select the desired passing method from the list that appears. After you select the passing method, the color of the proxy class changes, to indicate its type: black for reference, or blue for value (public fields/properties style).

Set the passing method for multiple proxy classes simultaneously

1. Select Project > Pass By Reference / Value from the menu bar.
2. The Pass by Reference / Value dialog box lists all proxy classes in the Exposed Proxies pane. Select the classes whose passing value you want to set.
3. Click the Reference or Value (Public fields/properties) button to associate the selected classes to the desired type.
4. Repeat steps 2 and 3 to select multiple groups of classes and set their passing methods.
5. Click OK.

Determining and changing the .NET version

If you get errors when using a .NET object in your application, you could have version issues. For example, many Microsoft system classes were added in .NET Version 2.0, including System.IO.DriveInfo and

© 2014 Adobe Systems Incorporated. All rights reserved.

System.Net.NetworkInformation.Ping. For examples of these classes in applications, see *Using .NET objects* in Using .NET classes and Using a .NET class directly in Example applications, respectively.
Use the following function to get the current .NET version:

```
<cffunction name="GetDotNetVersion" returntype="string">
 <cfset var seClass="">
 <cfobject type=".NET" name="seClass" class="System.Environment">
 <cfreturn seClass.Get_Version().ToString()>
</cffunction>
```

If the function reports that the active version is not the one you require, install or reinstall the correct version of the .NET framework redistributable package on the system that runs ColdFusion. Then reinstall the ColdFusion .NET extension so that it uses the correct .NET version.

Running the .NET extension agent as an application

The ColdFusion .NET extension installer configures the .NET-side extension agent to run automatically as the ColdFusion .NET service. You can also run the .NET extension agent as an application.

Run the .NET extension agent as an application

1. Ensure that you stopped the ColdFusion .NET service, if it was running.
2. Open a command prompt window and navigate to the jnbridge directory. On a stand-alone ColdFusion server configuration, this directory is *installDir*\jnbridge. On a system with a stand-alone .NET extension installation, or a JEE configuration, it is in the *.NETInstallDir*\jnbridge directory, and the default installation directory is C:\ColdFusonDotNetExtension.
3. Enter the following command:

```
JNBDotNetSide
```

#back to top

© 2014 Adobe Systems Incorporated. All rights reserved.

Integrating COM and CORBA Objects in CFML Applications

You can invoke COM (Component Object Model) or DCOM (Distributed Component Object Model) and CORBA (Common Object Request Broker) objects.

© 2014 Adobe Systems Incorporated. All rights reserved.

About COM and CORBA

Adobe ColdFusion supports access to COM and CORBA objects, which can be used in many applications.

About objects

COM and CORBA are two of the *object* technologies supported by ColdFusion. Other object technologies include Java and ColdFusion components. For more information on ColdFusion components see Building and Using ColdFusion Components.

An object is a self-contained module of data and its associated processing. An object is a building block that you can together with other objects and integrate into ColdFusion code to create an application.

A handle, or name, represents an object. Objects have *properties* that represent information. Objects also provide *m ethods* for manipulating the object and getting data from it. The exact terms and rules for using objects vary with the object technology.

You create instances of objects using the `cfobject` tag or the `CreateObject` function. You then use the object and its methods in ColdFusion tags, functions, and expressions. For more information on the ColdFusion syntax for using objects, see Creating and using objects.

About COM and DCOM

COM (Component Object Model) is a specification and a set of services defined by Microsoft to enable component portability, reusability, and versioning. DCOM (Distributed Component Object Model) is an implementation of COM for distributed services, which allows access to components residing on a network.

COM objects can reside locally or on any network node. COM is supported on Microsoft Windows platforms.

For more information on COM, go to the Microsoft COM website, www.microsoft.com/com.

About CORBA

CORBA (Common Object Request Broker Architecture) is a distributed computing model for object-oriented applications defined by the Object Management Group (OMG). In this model, an object is an encapsulated entity whose services are accessed only through well-defined interfaces. The location and implementation of each object is hidden from the client requesting the services. ColdFusion supports CORBA 2.3 on both Windows and UNIX.

CORBA uses an Object Request Broker (ORB) to send requests from applications on one system to objects executing on another system. The ORB allows applications to interact in a distributed environment, independent of the computer platforms on which they run and the languages in which they are implemented. For example, a ColdFusion application running on one system can communicate with an object that is implemented in C++ on another system.

CORBA follows a client-server model. The client invokes operations on objects that the server manages, and the server replies to requests. The ORB manages the communications between the client and the server using the Internet Inter-ORB Protocol (IIOP).

Each CORBA object has an interface that is defined in the CORBA Interface Definition Language (IDL). The CORBA IDL describes the operations that can be performed on the object, and the parameters of those operations. Clients do not have to know anything about how the interface is implemented to make requests.

To request a service from the server, the client application gets a handle to the object from the ORB. It uses the handle to call the methods specified by the IDL interface definition. The ORB passes the requests to the server, which processes the requests and returns the results to the client.

For information about CORBA, see the following OMG website, which is the main web repository for CORBA information: www.omg.com.

© 2014 Adobe Systems Incorporated. All rights reserved.

Creating and using objects

You use the cfobject tag or the CreateObject function to create a named instance of an object. You use other ColdFusion tags, such as `cfset` and `cfoutput`, to invoke the object properties and methods.
Many of the techniques for creating and using objects apply to both COM and CORBA objects. The examples here assume a sample object named "obj", and that the object has a property called "Property", and methods called "Method1", "Method2", and "Method3".

Creating objects

You create, or *instantiate* (create a named instance of) an object in ColdFusion with the `cfobject` tag or `CreateObject` function. The specific attributes or parameters that you use depend on the type of object you use, and are described in detail in Creating and using COM objects and Creating CORBA objects. The following examples use a `cfobject` tag to create a COM object and a `CreateObject` function to create a CORBA object:

```
<cfobject type="COM" action="Create" name="obj" class="sample.MyObject">
obj = CreateObject("CORBA", "d:\temp\tester.ior", "IOR", "Visibroker")
```

ColdFusion releases any object created by `cfobject` or `CreateObject`, or returned by other objects, at the end of the ColdFusion page execution.

Using properties

Use standard ColdFusion statements to access properties as follows:

1. To set a property, use a statement or `cfset`tag, such as the following:

   ```
   <cfset obj.property = "somevalue">
   ```

2. To get a property, use a statement or `cfset`tag, such as the following:

   ```
   <cfset value = obj.property>
   ```

As shown in this example, you do not use parentheses on the right side of the equation to get a property value.

Calling methods

Object methods usually take zero or more arguments. You send In arguments, whose values are not returned to the caller by value. You send Out and In,Out arguments, whose values are returned to the caller, by reference. Arguments sent by reference usually have their value changed by the object. Some methods have return values, while others do not.
Use the following techniques to call methods:

- If the method has no arguments, follow the method name with empty parentheses, as in the following `cfset`tag:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cfset retVal = obj.Method1()>
```

- If the method has one or more arguments, place the arguments in parentheses, separated by commas, as in the following example, which has one integer argument and one string argument:

```
<cfset x = 23>
<cfset retVal = obj.Method1(x, "a string literal")>
```

- If the method has reference (Out or In,Out) arguments, use double quotation marks (") around the name of the variable you are using for these arguments, as shown for the variable x in the following example:

```
<cfset x = 23>
<cfset retVal = obj.Method2("x","a string literal")>
<cfoutput> #x#</cfoutput>
```

  In this example, if the object changes the value of x, it now contains a value other than 23.

Calling nested objects

ColdFusion supports nested (scoped) object calls. For example, if an object method returns another object, and you invoke a property or method on that object, you can use the syntax in either of the following examples:

```
<cfset prop = myObj.X.Property>
```

or

```
<cfset objX = myObj.X>
<cfset prop = objX.Property>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

Getting started with COM and DCOM

ColdFusion is an automation (late-binding) COM client. As a result, the COM object must support the IDispatch interface, and arguments for methods and properties must be standard automation types. Because ColdFusion is a typeless language, it uses the object type information to correctly set up the arguments on call invocations. Any ambiguity in the object data types can lead to unexpected behavior.
In ColdFusion, only use server-side COM objects, which do not have a graphical user interface. If your ColdFusion application invokes an object with a graphical interface in a window, the component could appear on the web server desktop, not on the client desktop. This behavior can take up ColdFusion server threads and prevent further web server requests from being serviced.
ColdFusion can call Inproc, Local, or Remote COM objects. The attributes specified in the `cfobject` tag determine which type of object is called.

COM requirements

To use COM components in your ColdFusion application, you need at least the following items:

- The COM objects (typically DLL or EXE files) that you want to use in your ColdFusion application pages. Ensure that these components implement the IDispatch interface, and therefore allow late binding.
- Microsoft OLE/COM Object Viewer, available from Microsoft. This tool lets you view registered COM objects. Object Viewer lets you view the class information of an object so that you can properly define the `class` attribute for the `cfobject` tag. It also displays the interfaces the object supports, so you can discover the properties and methods (for the IDispatch interface) of the object.

Registering the object

After you acquire an object, register it with Windows for ColdFusion (or any other program) to find it. Some objects have setup programs that register objects automatically, while others require manual registration.
You can register Inproc object servers (.dll or .ocx files) manually by running the regsvr32.exe utility using the following form:

```
regsvr32 c:\path\servername.dll
```

You typically register Local servers (.exe files) either by starting them or by specifying a command-line parameter, such as the following:

```
C:\pathname\servername.exe -register
```

Finding the component ProgID and methods

Your COM object supplier provides documentation that explains each of the component methods and the ProgID. If you do not have documentation, use either the ColdFusion `cfdump` tag or the OLE/COM Object Viewer to view the component interface.

Using the cfdump tag to view COM object interfaces

Effective with ColdFusion, the ColdFusion `cfdump` tag displays the following information about a COM object:

- Public methods
- Put properties
- Get properties

© 2014 Adobe Systems Incorporated. All rights reserved.

The method and property information include the parameter or property types and whether they are in, out, optional, or retval values. The `cfdump` tag output does not include the ProgID of the object.

> **Note**
>
> The dump header indicates the ColdFusion object class, which is coldfusion.runtime.com.ComProxy, and the COM object CLSID.

Using the OLE/COM Object Viewer

The OLE/COM Object Viewer installation installs the executable, by default, as \mstools\bin\oleview.exe. You use the Object Viewer to retrieve a COM object ProgID, as well as its methods and properties.
To find an object in the Object Viewer, it must be registered, as described in *Registering the object*. The Object Viewer retrieves all COM objects and controls from the Registry, and presents the information in a simple format, sorted into groups for easy viewing.
By selecting the category and then the component, you can see the ProgID of a COM object. The Object Viewer also provides access to options for the operation of the object.

To view object properties:

1. Open the Object Viewer and scroll to the object that you want to examine.
2. Select and expand the object in the left pane of the Object Viewer.
3. Right-click the object to view it, including the TypeInfo. If you view the TypeInfo, you see the object methods and properties. Some objects do not have access to the TypeInfo area, which is determined when an object is built and by the language used.

© 2014 Adobe Systems Incorporated. All rights reserved.

Creating and using COM objects

Use the `cfobject` tag or the `CreateObject` function to create an instance of the COM object (component) in ColdFusion before your application pages can invoke any methods or assign any properties in the component. For example, the following code uses the `cfobject` tag to create the Windows CDO (Collaborative Data Objects) for NTS NewMail object to send mail:

```
<cfobject type="COM"
  action="Create"
  name="Mailer"
  class="CDONTS.NewMail">
```

The following line shows how to use the corresponding `CreateObject` function in CFScript:

```
Mailer = CreateObject("COM", "CDONTS.NewMail");
```

Several examples in this documentation use this object.

> **Note**
>
> CDO is installed by default on all Windows NT and 2000 operating systems that have installed the Microsoft SMTP server. In Windows NT Server environments, the SMTP server is part of the Option Pack 4 setup. In Windows 2000 Server and Workstation environments, it is bundled with the operating system.

note. >>>The CDO for NTS NewMail component includes methods and properties to perform a wide range of mail-handling tasks. (In the OLE/COM Object Viewer, methods and properties can be grouped, so you could find it difficult to distinguish between them at first.)
The CDO for NTS NewMail object includes the following properties:

```
Body [ String ]
Cc[ String ]
From[ String ]
Importance[ Long ]
Subject[ String ]
To[ String ]
```

You use these properties to define elements of your mail message. The CDO for NTS NewMail object also includes a `send` method which has optional arguments to send messages.

Connecting to COM objects

The `action` attribute of the cfobject tag provides the following two ways to connect to COM objects:

- **Create method** (`cfobject action="Create"`) Takes a COM object, typically a DLL, and instantiates it before executing methods and assigning properties.
- **Connect method** (`cfobject action="Connect"`) Links to an object, typically an executable, that is

© 2014 Adobe Systems Incorporated. All rights reserved.

already running on the server.You can use the optional `cfobject context` attribute to specify the object context. If you do not specify a context, ColdFusion uses the setting in the Registry. The following table describes the `context` attribute values:

Attribute value	Description
`InProc`	An in-process server object (typically a DLL) that is running in the same process space as the calling process, such as ColdFusion.
`local`	An out-of-process server object (typically an EXE file) that is running outside the ColdFusion process space but running locally on the same server.
`remote`	An out-of-process server object (typically an EXE file) that is running remotely on the network. If you specify `remote`, Also use the `server` attribute to identify where the object resides.

Setting properties and executing methods

The following example, which uses the sample Mailer COM object, shows how to assign properties to your mail message and how to execute component methods to handle mail messages.
In the example, form variables contain the method parameters and properties, such as the name of the recipient, the desired e-mail address, and so on:

```
<!--- First, create the object --->
<cfobject type="COM"
 action="Create"
 name="Mailer"
 class="CDONTS.NewMail">

<!--- Second, use the form variables from the user entry form to populate a number
  of properties necessary to create and send the message. --->
<cfset Mailer.From = "#Form.fromName#">
<cfset Mailer.To = "#Form.to#">
<cfset Mailer.Subject = "#Form.subject#">
<cfset Mailer.Importance = 2>
<cfset Mailer.Body = "#Form.body#">
<cfset Mailer.Cc = "#Form.cc#">

<!--- Last, use the Send() method to send the message.
 Invoking the Send() method destroys the object.--->
<cfset Mailer.Send()>
```

Note

Use the `cftry` and `cfcatch` tags to handle exceptions thrown by COM objects. For more information on exception handling, see Handling runtime exceptions with ColdFusion tags.

Releasing COM objects

© 2014 Adobe Systems Incorporated. All rights reserved.

By default, COM object resources are released when the Java garbage collector cleans them. You can use the `ReleaseCOMObject` function to immediately release resources if an object is no longer needed.
Use the `ReleaseCOMObject` function to release COM objects that are launched as an external process, such as Microsoft Excel. The garbage collector does not always clean these processes in a short time, resulting in multiple external processes running, which drains system resources.
If the COM object has an end method, such as a quit method that terminates the program, call this method before you call the `ReleaseComObject` function. If you use the `ReleaseComObject` function on an object that is in use, the object is prematurely released and your application gets exceptions.

Example

The following example creates a Microsoft Excel application object, uses it, then releases the object when it is no longer needed:

```
<h3>ReleaseComObject Example</h3>
<cfscript>
obj = CreateObject("Com", "excel.application.9");
//code that uses the object goes here
obj.quit();
ReleaseComObject(obj);
</cfscript>
```

General COM object considerations

When you use COM objects, consider the following to prevent and resolve errors:

- Ensuring correct threading
- Using input and output arguments
- Understanding common COM-related error messages

Ensuring correct threading

Improper threading can cause serious problems when using a COM object in ColdFusion. Make sure that the object is *thread-safe*. An object is thread-safe if it can be called from many programming threads simultaneously, without causing errors.
Visual Basic ActiveX DLLs are typically not thread-safe. If you use such a DLL in ColdFusion, you can make it thread-safe by using the OLE/COM Object Viewer to change the threading model of the object to the Apartment model.
If you are planning to store a reference to the COM object in the Application, Session, or Server scope, do not use the Apartment threading model. This threading model is intended to service only a single request. If your application requires you to store the object in any of these scopes, keep the object in the Both threading model, and lock all code that accesses the object, as described in Locking code with cflock.

Change the threading model of a COM Object

1. Open the OLE/COM Object Viewer.
2. Select All Objects under Object Classes in the left pane.
3. Locate your COM object. The left pane lists the objects by name.
4. Select your object.
5. Select the Implementation tab in the right pane.
6. Select the Inproc Server tab, below the App ID field.
7. Select the Threading Model drop-down list and select Apartment or Both, as appropriate.

© 2014 Adobe Systems Incorporated. All rights reserved.

Using input and output arguments

COM object methods in arguments are passed by value. The COM object gets a copy of the variable value, so you can specify a ColdFusion variable without surrounding it with quotation marks.
COM object out method arguments are passed by reference. The COM object modifies the contents of the variable on the calling page, so the calling page can use the resulting value. To pass a variable by reference, surround the name of an existing ColdFusion variable with quotation marks. If the argument is a numeric type, assign the variable a valid number before you make the call. For example:

```
<cfset inStringArg="Hello Object">
<cfset outNumericArg=0>
<cfset result=myCOMObject.calculate(inStringArg, "outNumericArg")>
```

The string "Hello Object" is passed to the object's calculate method as an input argument. The method sets the value of outNumericArg to a numeric value.

Understanding common COM-related error messages

The following table described some error messages you could encounter when using COM objects:

Error	Cause
Error Diagnostic InformationError trying to create object specified in the tag. COM error 0x800401F3. Invalid class string.	The COM object is not registered or does not exist.
Error Diagnostic InformationError trying to create object specified in the tag. COM error 0x80040154. Class not registered.	The COM object is not registered or does not exist. This error usually occurs when an object existed previously, but was removed.
Error Diagnostic InformationFailed attempting to find "SOMEMETHOD" property/method on the object COM error 0x80020006.Unknown name.	The COM object was instantiated correctly, but the method you specified does not exist.

Accessing Complex COM Objects using Java proxies

ColdFusion supports Java proxies to access COM objects. If you do not create Java proxies in advance, ColdFusion must dynamically discover the COM interface. This technique can have two disadvantages:

- Dynamic discovery takes time and can reduce server performance with frequently used complex COM objects.
- Dynamic discovery uses the IDispatcher interface to determine the COM object features, and does not always handle some complex COM interfaces.
  To overcome these problems, ColdFusion includes a utility, com2java.exe, that creates static Java stub proxy classes for COM objects. ColdFusion can use these Java stubs to access COM objects more efficiently than when it creates the proxies dynamically. Additionally, the com2java.exe utility can create stubs for features that the dynamic proxy generator could miss.
  ColdFusion ships with pregenerated stubs for the Windows XP, Windows 2000, and Windows 97 editions of Microsoft Excel, Microsoft Word, and Microsoft Access. ColdFusion is configured to automatically use these stubs.
  If you create Java stub files for a COM object, you continue to use the `cfobject` tag with a `type` attribute

© 2014 Adobe Systems Incorporated. All rights reserved.

value of `COM`, or the `CreateObject` function with a first argument of `COM`, and you access the object properties and methods as you normally do for COM objects in ColdFusion.
Use the following steps to use the com2java.exe utility. This procedure uses Microsoft Outlook as an example.

To create Java stub files for COM objects:

1. Configure your system as follows:
   a. Ensure that a JDK (Java Development Kit) is correctly installed, including proper configuration of the CLASSPATH and the command prompt PATH variable.
   b. Add _CF_root_lib\jintegra.jar to your CLASSPATH.
2. Make a new directory for the Java stub files; for example:

```
mkdir C:\src\outlookXP
```

   This directory can be temporary. You add files from the directory to a ColdFusion JAR file.

3. Run the *CF_root*\Jintegra\bin\com2java.exe program from a command line or the Windows Start Menu. A window appears.
   a. If a COM class implements multiple interfaces that define methods with the same names, click the Options button and clear the Implement interfaces that may conflict option. The generated Java stub classes do not implement the additional, conflicting, interfaces. You can still access the interfaces using the getAs_XXX_ method that is generated. See the generated comments in the Java files.
   b. Click the `Select` button.
   c. Select your COM object's Type Library or DLL. For Microsoft Outlook in Windows XP, it is normally Program Files\Microsoft Office\Office10\MSOUTL.OLB.
   d. Enter a package name (for example, *outlookXP*) in the Java package field in the com2java dialog box. This package will contain all the classes for the Java stubs for the COM object.

> **Note**
>
> Adobe uses a package name that starts with coldfusion.runtime.com.com2java_ for the packages that contain the preinstalled Java stubs for Microsoft Excel, Microsoft Word, and Microsoft Access. For example, the name for the package containing the Microsoft Word XP Java stub classes is_ coldfusion.runtime.com.com2java.wordXP_. This package name hierarchy results in the wordXP classes having a path inside the msapps.jar file of coldfusion\runtime\com\com2java\wordXP\className.class. Although this naming convention is not necessary, consider using a similar package naming convention for clarity, if you use many COM objects._

1. a. Click the Generate Proxies button to display the File browser. Select the directory you created in step 2., and click the file browser OK button to generate the stub files.
   b. Click Close to close the com2java.exe utility.The files generated in your directory include the following:
   - A Java interface and proxy class for each COM interface
   - A Java class for each COM class
   - A Java interface for each ENUM (a set of constant definitions)
2. Compile your Java code. In a command prompt, do the following:
   a. Make the directory that contains the Java stubs (in this example, C:\src\outlookXP) your working directory.
   b. Enter the following line:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
javac -J-mx100m -J-ms100m *.java
```

The compiler switches ensure that you have enough memory to compile all the necessary files.

> **Note**
>
> If you did not place jintegra.jar on your CLASSPATH in step 1b, add the switch `-classpath:/cf_root/lib/jintegra.jar`, where `cf_root` *is the directory where ColdFusion is installed, to the command.*

1. Ensure that the ColdFusion server is not running. To stop the ColdFusion server, open the Services control panel, select ColdFusion application server, and click Stop.
2. Add your .class files to the ColdFusion Microsoft application Java stubs file by doing the following:
   a. In the Windows Command prompt, make the parent directory of the directory that contains your class files your working directory. In this example, make c:\src your working director by entering **cd ..** in the Command prompt from step 4.
   b. Enter the following command:

   ```
   jar -uvf cf_root\lib\msapps.jar directoryName\*.class
   ```

   Where *cf_root* is the directory where ColdFusion is installed and *directoryName*is the name of the directory that contains the class files. For the OutlookXP example, enter the following line:

   ```
   jar -uvf C:\CFusion\lib\msapps.jar outlookXP\*.class
   ```

3. Update the *cf_root*/lib/neo-comobjmap.xml file by appending your object definition to the list. The object definition consists of the following lines:

   ```
   <var name="progID">
   <string>PackageName.mainClass</string>
   </var>
   ```

   Use the following values in these lines:

- **ProgID** The COM object ProgID, as displayed in the OLE/COM object viewer.
- **PackageName** The package name you specified in step 3c.
- **mainClass** The main class of the COM object. The main class contains the methods you invoke. For many Microsoft applications, this class is Application. In general, the largest class file created in step 4 is the main class. For example, to add outlookXP to neo-comobjmap.xml, add the lines in bold text above the `</struct>`end tag:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<var name="access.application.9">
<string>coldfusion.runtime.com.com2java.access2k.Application</string>
</var>
<var name="outlook.application.10">
<string>outlookXP.Application</string>
</var>
</struct>
```

In this example, *outlook.application.10* is the ProgID of the Outlook COM object, *outlookXP* is the package name you specified in step 3c, and *Application* is the main class of the COM object.

1. Restart the ColdFusion server: Open the Services control panel, select ColdFusion application server, and click Start.
2. After you have installed the stubs, you can delete the directory you created in step 2., including all its contents.

Using the Application Scope to improve COM performance

The Java call to create a COM object instance can take substantial time. As a result, creating COM objects in ColdFusion can be substantially slower than in ColdFusion 5. For example, on some systems, creating a Microsoft Word application object could take over one second using ColdFusion, while on the same system, the overhead of creating the Word object could be about 200 milliseconds.
Therefore, in ColdFusion, you can improve COM performance substantially if you can share a single COM object in the Application scope among all pages.
Use this technique only if the following are true:

- The COM object need not be created for every request or session. (For session-specific objects, consider using the technique described here with the Session scope in place of the Application scope.)
- The COM object is designed for sharing.
  Because the object can be accessed from multiple pages and sessions simultaneously, also consider the following threading and locking issues:

- For best performance, make the object multi-threaded. Otherwise, only one request can access the object at a time.
- Lock the code that accesses and modifies common data. In general, you do not have to lock code that modifies a shared object's data, including writable properties or file contents, if multiple requests do not share the data (as opposed to the object) . However, specific locking needs depend on the COM object's semantics, interface, and implementation.
- All `cflock` tags in the application that use an Application scope lock share one lock. Therefore, code that accesses a frequently used COM object inside an Application scope lock can become a bottleneck and reduce throughput if many users request pages that use the object. In some cases, you can avoid some contention by placing code that uses the COM object in named locks. Place the code that creates the object in an Application scope lock.

Note

You can also improve the performance of some COM objects by creating Java stubs, as described in *Accessing Complex COM Objects using Java proxies* above. Using a Java stub does not improve performance as much as sharing the COM object, but the technique works with all COM objects. Also, generate Java stubs to correctly access complex COM objects that do not properly make all their features available through the COM IDispatcher interface. Therefore, to get the greatest performance increase and prevent possible problems, use both techniques.

© 2014 Adobe Systems Incorporated. All rights reserved.

Example 1: Using the FileSystem object

The following example uses the Microsoft FileSystem Scripting object in the Application scope. This code creates a user-defined function that returns a structure that consists of the drive letters and free disk space for all hard drives on the system.

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cfapplication name="comtest" clientmanagement="No" Sessionmanagement="yes">

<!--- Uncomment the following line if you must delete the object from the
Application scope during debugging. Then restore the comments.
This technique is faster than stopping and starting the ColdFusion server. --->
 <!--- <cfset structdelete(Application, "fso")> --->

<!--- The getFixedDriveSpace user-defined function returns a structure with
the drive letters as keys and the drive's free space as data for all fixed
drives on a system. The function does not take any arguments --->

<cffunction name="getFixedDriveSpace" returnType="struct" output=True>
 <!--- If the FileSystemObject does not exist in the Application scope,
 create it. --->
 <!--- For information on the use of initialization variables and locking in
  this code, see "Locking application variables efficiently" in Chapter 15,
  "Using Persistent Data and Locking" --->
 <cfset fso_is_initialized = False>
 <cflock scope="application" type="readonly" timeout="120">
  <cfset fso_is_initialized = StructKeyExists(Application, "fso")>
 </cflock>
 <cfif not fso_is_initialized >
  <cflock scope="Application" type="EXCLUSIVE" timeout="120">
   <cfif NOT StructKeyExists(Application, "fso")>
    <cfobject type="COM" action="create" class="Scripting.FileSystemObject"
     name="Application.fso" server="\\localhost">
   </cfif>
  </cflock>
 </cfif>

 <!--- Get the drives collection and loop through it to populate the
  structure. --->
 <cfset drives=Application.fso.drives()>
 <cfset driveSpace=StructNew()>
 <cfloop collection="#drives#" item="curDrive">
  <!--- A DriveType of 2 indicates a fixed disk --->
  <cfif curDrive.DriveType IS 2>
  <!--- Use dynamic array notation with the drive letter for the struct key
  --->
   <cfset driveSpace["#curDrive.DriveLetter#"]=curDrive.availablespace>
  </cfif>
 </cfloop>
 <cfreturn driveSpace>
</cffunction>

<!--- Test the function. Get the execution time for running the function --->
<cfset start = getTickCount()>
<cfset DriveInfo=getFixedDriveSpace()>
<h3>Getting fixed drive available space</h3>
<cfoutput>Execution Time: #int(getTickCount()-start)#
milliseconds</cfoutput><br><br>
<cfdump label="Drive Free Space" var="#driveInfo#">
```

Example 2: Using the Microsoft Word application object

© 2014 Adobe Systems Incorporated. All rights reserved.

The following example uses the Microsoft Word application COM object in the Application scope to convert a Word document to HTML. This example works with Word 2000 as written. To work with Word 97, change "Val(8)" to "Val(10)".
This example uses an Application scope lock to ensure that no other page interrupts creating the object. Once the Word object exists, the example uses a named lock to prevent simultaneous access to the file that is being converted.

```
<cfapplication name="comtest" clientmanagement="No" Sessionmanagement="yes">
<!--- Uncomment the following line if you must delete the object from the
Application scope --->
<!--- <cfset structdelete(Application, "MyWordObj")> --->

<!--- use the GetTickCount function to get a current time indicator, used for
  displaying the total processing time. --->
<cfset start = GetTickCount()>
<!--- If necessary, create the Word.application object and place it in the
  Application scope --->
<cfset WordObj_is_initialized = False>
<cflock scope="application" type="readonly" timeout=120>
 <cfset WordObj_is_initialized = StructKeyExists(application, "MyWordObj")>
</cflock>
<cfif not WordObj_is_initialized >
 <cflock scope="Application" type="exclusive" timeout="120">
  <cfif not StructKeyExists(application, "MyWordObj")>

<!--- First try to connect to an existing Word object --->
   <cftry>
    <cfobject type="com"
     action="connect"
     class="Word.application"
     name="Application.MyWordobj"
     context="local">
    <cfcatch>
<!--- No object exists, create one --->
     <cfobject type="com"
      action="Create"
      class="Word.application"
      name="Application.MyWordobj"
      context="local">
    </cfcatch>
   </cftry>

   <cfset Application.mywordobj.visible = False>
  </cfif>
 </cflock>
</cfif>

<!--- Convert a Word document in temp.doc to an HTML file in temp.htm. --->
<!--- Because this example uses a fixed filename, multiple pages could try
 to use the file simultaneously. The lock ensures that all actions from
 reading the input file through closing the output file are a single "atomic"
 operation, and the next page cannot access the file until the current page
 completes all processing.
 Use a named lock instead of the Application scope lock to reduce lock contention.
--->
<cflock name="WordObjLock" type="exclusive" timeout="120">
 <cfset docs = application.mywordobj.documents()>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

```
 <cfset docs.open("c:\CFusion\wwwroot\temp.doc")>
 <cfset converteddoc = application.mywordobj.activedocument>
 <!--- Val(8) works with Word 2000. Use Val(10) for Word 97 --->
 <cfset converteddoc.saveas("c:\CFusion\wwwroot\temp.htm",val(8))>
 <cfset converteddoc.close()>
</cflock>

<cfoutput>
 Conversion of temp.htm Complete<br>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

```
 Execution Time: #int(getTickCount()-start)# milliseconds<br>
</cfoutput>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

Getting started with CORBA

The ColdFusion cfobject tag and CreateObject function support CORBA through the Dynamic Invocation Interface (DII). As with COM, the object's type information must be available to ColdFusion. Therefore, an IIOP-compliant Interface Repository (IR) must be running on the network, and the object's Interface Definition Language (IDL) specification must be registered in the IR. If your application uses a naming service to get references to CORBA objects, a naming service must also be running on the network.

ColdFusion loads ORB runtime libraries at startup using a connector, which does not tie ColdFusion customers to a specific ORB vendor. ColdFusion currently includes connectors for the Borland Visibroker 4.5 ORB. The source necessary to write connectors for other ORBs is available under NDA to select third-party candidates and ORB vendors.

configuring and enabling CORBA access in ColdFusion requires several steps. For detailed instructions, see *Installing ColdFusion*.

Note

When you enable CORBA access in ColdFusion, one step requires you to start the Interface Repository using an IDL file. This file must contain the IDL for all the CORBA objects that you invoke in ColdFusion applications on the server.

© 2014 Adobe Systems Incorporated. All rights reserved.

Creating CORBA objects

In ColdFusion, the cfobject tag and CreateObject function create a stub, or proxy object, for the CORBA object on the remote server. You use this stub object to invoke the remote object.
The following table describes the attributes you use in the `cfobject` tag to create a CORBA object:

Attribute	Description
`type`	Must be CORBA. COM is the default.
`context`	Specifies the CORBA binding method, that is, how the object is obtained, as follows: • `IOR`-Uses a file containing the object's unique Interoperable Object Reference. • `NameService`-Uses a naming service.
`class`	Specifies the information required for the binding method to access the object.If you set the `context` att ribute to `IOR`, The `class` attribute must be to the full path of a file containing the string version of the IOR. ColdFusion must be able to read this IOR file at all times, so make it local to the server or locate it on the network in an accessible place.If you set the `context` attribute to `NameService`, The `class` attribute must be a name delimited by forward slashes, such as MyCompany/Department/Dev. You can use period-delimited "kind" identifiers as part of the class attribute; for example, adobe.current/Eng.current/CF"
`name`	Specifies the name (handle) that your application uses to call the object's interface.
`locale`	(Optional) Identifies the connector configuration. You can omit this option if ColdFusion Administrator has only one connector configuration, or if it has multiple connector configurations and you want to use the one that is currently selected in the Administrator. If you specify this attribute, it must be an ORB name you specified in the CORBA Connector ORB Name field when you configured a CORBA connector in ColdFusion Administrator; for example, Visibroker.

For example, use the following CFML to invoke a CORBA object specified by the tester.ior file if you configured your ORB name as Visibroker:

```
<cfobject action = "create" type = "CORBA" context = "IOR"
 class = "d:\temp\tester.ior" name = "handle" locale = "Visibroker">
```

© 2014 Adobe Systems Incorporated. All rights reserved.

When you use the `CreateObject` function to invoke this CORBA object, specify the name as the function return variable, and specify the type, class, context, and locale as arguments. For example, the following line creates the same object as the preceding `cfobject` tag:

```
handle = CreateObject("CORBA", "d:\temp\tester.ior", "IOR", "Visibroker")
```

Using a naming service

Currently, ColdFusion can only resolve objects registered in a CORBA 2.3-compliant naming service.
If you use a naming service, make sure that its naming context is identical to the naming context specified in the property file of the Connector configuration in use, as specified in the ColdFusion Administrator CORBA Connectors page. The property file must contain the line "SVCnameroot=*name*" where *name* is the naming context being used. The server implementing the object must bind to this context, and register the appropriate name.

© 2014 Adobe Systems Incorporated. All rights reserved.

Using CORBA objects in ColdFusion

After you create the object, you can invoke attributes and operations on the object using the syntax described in Creating and using objects.

Using CORBA interface methods in ColdFusion

When you use the cfobject tag or the CreateObject function to create a CORBA object, ColdFusion creates a handle to a CORBA interface: the `cfobject name` attribute or the `CreateObject` function return variable. For example, the following CFML creates a handle named myHandle:

```
<cfobject action = "create" type = "CORBA" context = "IOR"
 class = "d:\temp\tester.ior" name = "myHandle" locale="visibroker">
<cfset myHandle = CreateObject("CORBA", "d:\temp\tester.ior", "IOR", "visibroker")
```

You use the handle name to invoke all of the interface methods, as in the following CFML:

```
<cfset ret=myHandle.method(foo)>
```

Method name case considerations

Method names in IDL are case sensitive. However, ColdFusion is not case sensitive. Therefore, do not use methods that differ only in case in IDL.
For example, the following IDL method declarations correspond to two different methods:

```
testCall(in string a); // method #1
TestCall(in string a); // method #2
```

However, ColdFusion cannot differentiate between the two methods. If you call either method, you cannot be sure which of the two gets invoked.

Passing parameters by value (in parameters)

CORBA in parameters are always passed by value. When calling a CORBA method with a variable in ColdFusion, specify the variable name without quotation marks, as shown in the following example:

IDL	void method(in string a);
CFML	<cfset foo="my string"><cfset ret=handle.method(foo)>

Passing variables by reference (out and inout parameters)

CORBA out and inout parameters are always passed by reference. As a result, if the CORBA object modifies the value of the variable that you pass when you invoke the method, your ColdFusion page gets the modified value.
To pass a parameter by reference in ColdFusion, specify the variable name in double-quotation marks in the CORBA method. The following example shows an IDL line that defines a method with a string variable, b, that is passed in and out of the method by reference. It also shows CFML that calls this method.

© 2014 Adobe Systems Incorporated. All rights reserved.

IDL	void method(in string a, inout string b);
CFML	<cfset foo = "My Initial String"><cfset ret=handle.method(bar, "foo")><cfoutput>#foo#</cfoutput>

In this case, the ColdFusion variable foo corresponds to the inout parameter b. When the CFML executes, the following happens:

1. ColdFusion calls the method, passing it the variable by reference.
2. The CORBA method replaces the value passed in, "My Initial String", with some other value. Because the variable was passed by reference, this action modifies the value of the ColdFusion variable.
3. The `cfoutput` tag prints the new value of the foo variable.

Using methods with return values

Use CORBA methods that return values as you would any ColdFusion function; for example:

IDL	double method(out double a);
CFML	<cfset foo=3.1415><cfset ret=handle.method("foo")><cfoutput>#ret#</cfoutput>

Using IDL types with ColdFusion variables

ColdFusion supports specific CORBA data types and converts between CORBA types and ColdFusion data.

IDL support

The following table shows which CORBA IDL types ColdFusion supports, and whether they can be used as parameters or return variables. (NA means not applicable.)

CORBA IDL type	General support	As parameters	As return value
constants	No	No	No
attributes	Yes (for properties)	NA	NA
enum	Yes (as an integer)	Yes	Yes
union	No	No	No
sequence	Yes	Yes	Yes
array	Yes	Yes	Yes
interface	Yes	Yes	Yes
typedef	Yes	NA	NA

© 2014 Adobe Systems Incorporated. All rights reserved.

struct	Yes	Yes	Yes
module	Yes	NA	NA
exception	Yes	NA	NA
any	No	No	No
boolean	Yes	Yes	Yes
char	Yes	Yes	Yes
wchar	Yes	Yes	Yes
string	Yes	Yes	Yes
wstring	Yes	Yes	Yes
octet	Yes	Yes	Yes
short	Yes	Yes	Yes
long	Yes	Yes	Yes
float	Yes	Yes	Yes
double	Yes	Yes	Yes
unsigned short	Yes	Yes	Yes
unsigned long	Yes	Yes	Yes
longlong	No	No	No
unsigned longlong	No	No	No
void	Yes	NA	Yes

Data type conversion

The following table lists IDL data types and the corresponding ColdFusion data types:

IDL type	ColdFusion type
boolean	Boolean
char	One-character string

© 2014 Adobe Systems Incorporated. All rights reserved.

wchar	One-character string
string	String
wstring	String
octet	One-character string
short	Integer
long	Integer
float	Real number
double	Real number
unsigned short	Integer
unsigned long	Integer
void	Not applicable (returned as an empty string)
struct	Structure
enum	Integer, where 0 corresponds to the first enumerator in the enum type
array	Array (must match the array size specified in the IDL)
sequence	Array
interface	An object reference
module	Not supported (cannot dereference by module name)
exception	ColdFusion throws an exception of type `coldfusion.runtime.corba.CorbaUserException`
attribute	Object reference using dot notation

Boolean data considerations

ColdFusion treats any of the following as Boolean values:

True	"yes", "true", or 1
False	"no", "false", or 0

© 2014 Adobe Systems Incorporated. All rights reserved.

You can use any of these values with CORBA methods that take Boolean parameters, as the following code shows:

IDL	`module Tester`
	`{`
	`interface TManager`
	`{`
	`void testBoolean(in boolean a);`
	`void testOutBoolean(out boolean a);`
	`void testInoutBoolean(inout boolean a);`
	`boolean returnBoolean();`
	`}`
	`}`

CFML	`<cfset handle = CreateObject("CORBA", "d:\temp\tester.ior", "IOR", "") >`
	
	`<cfset ret = handle.testboolean("yes")>`
	`<cfset mybool = True>`
	`<cfset ret = handle.testoutboolean("mybool")>`
	`<cfoutput>#mybool#</cfoutput>`
	`<cfset mybool = 0>`
	`<cfset ret = handle.testinoutboolean("mybool")>`
	
	`<cfoutput>#mybool#</cfoutput>`
	`<cfset ret = handle.returnboolean()>`
	`<cfoutput>#ret#</cfoutput>`

Struct data type considerations

For IDL struct types, use ColdFusion structures. You can prevent errors by using the same case for structure key names in ColdFusion as you do for the corresponding IDL struct field names.

Enum type considerations

ColdFusion treats the enum IDL type as an integer with the index starting at 0. As a result, the first enumerator corresponds to 0, the second to 1, and so on. In the following example, the IDL enumerator a corresponds to 0, b to 1 and c to 2:

© 2014 Adobe Systems Incorporated. All rights reserved.

IDL	`module Tester`
	`{`
	`enum EnumType {a, b, c};`
	`interface TManager`
	`{`
	`void testEnum(in EnumType a);`
	`void testOutEnum(out EnumType a);`
	`void testInoutEnum(inout EnumType a);`
	`EnumType returnEnum();`
	`}`
	`}`

CFML	`<cfset handle = CreateObject("CORBA", "d:\temp\tester.ior", "IOR", "") >`
	
	`<cfset ret = handle.testEnum(1)>`

In this example, the CORBA object gets called with the second (*not* first) entry in the enumerator.

Double-byte character considerations

If you are using an ORB that supports CORBA later than version 2.0, you do not have to do anything to support double-byte characters. Strings and characters in ColdFusion convert appropriately to wstring and wchar data when they are used. However, the CORBA 2.0 IDL specification does not support the wchar and wstring types, and uses the 8-bit Latin-1 character set to represent string data. In this case, you cannot pass parameters containing those characters, however, you can call parameters with char and string types using ColdFusion string data.

© 2014 Adobe Systems Incorporated. All rights reserved.

Handling CORBA object method exceptions

Use the `cftry` and `cfcatch` tags to catch CORBA object method exceptions thrown by the remote server, as follows:

1. Specify `type="coldfusion.runtime.corba.CorbaUserException"` in the `cfcatch` tag to catch CORBA exceptions.
2. Use the `cfcatch.getContents` method to get the contents of the exception object.
   The `cfcatch.getContents` method returns a ColdFusion structure containing the data specified by the IDL for the exception.
   The following code example shows the IDL for a CORBA object that raises an exception defined by the PrimitiveException exception type definition, and the CFML that catches the exception and displays the contents of the object.

IDL	`interface myInterface`
	`{`
	`exception PrimitiveException`
	`{`
	`long l;`
	`string s;`
	`float f;`
	`};`
	`void testPrimitiveException()`
	`raises (PrimitiveException);`
	`}`

CFML	`<cftry>`
	`<cfset ret0 =`
	`handle.testPrimitiveException()>`
	`<cfcatch`
	`type=coldfusion.runtime.corba.Corb`
	`aUserException>`
	`<cfset exceptStruct=`
	`cfcatch.getContents()>`
	`<cfdump var ="#exceptStruct#">`
	`</cfcatch>`
	`</cftry>`

© 2014 Adobe Systems Incorporated. All rights reserved.

CORBA example

The following code shows an example of using a LoanAnalyzer CORBA object. This simplified object determines whether an applicant is approved for a loan based on the information that is supplied.
The LoanAnalyzer CORBA interface has one method, which takes the following two in arguments:

- An Account struct that identifies the applicant's account. It includes a Person struct that represents the account holder, and the applicant's age and income.
- A CreditCards sequence, which corresponds to the set of credit cards the user currently has. A member of the CardType enumerator represents the credit card type. (This example assumes that the applicant has no more than one of any type of card.)
  The object returns a Boolean value indicating whether the application is accepted or rejected.
  The CFML does the following:

1. Initializes the values of the ColdFusion variables that are used in the object method. In a more complete example, the information would come from a form, query, or both. The code for the Person and Account structs is straightforward. The cards variable, which represents the applicant's credit cards, is more complex. The interface IDL uses a sequence of enumerators to represent the cards. ColdFusion represents an IDL sequence as an array, and an enumerator as 0-indexed number indicating the position of the selected item among the items in the enumerator type definition.In this case, the applicant has a Master Card, a Visa card, and a Diners card. Because Master Card (MC) is the first entry in the enumerator type definition, it is represented in ColdFusion by the number 0. Visa is the third entry, so it is represented by 2. Diners is the fifth entry, so it is represented by 4. To represent the sequence, place these numbers in an array. Doing so results in a three-element, one-dimensional array containing 0, 2, and 4.
2. Instantiates the CORBA object.
3. Calls the approve method of the CORBA object and gets the result in the return variable, ret.
4. Displays the value of the ret variable, Yes, or No.

- **IDL**

```
struct Person
{
long pid;
string name;
string middle;
string last_name;
}
struct Account
{
Person person;
short age;
double income;
}
double loanAmountl
enum cardType {AMEX, VISA, MC, DISCOVER, DINERS};
typedef sequence<cardType> CreditCards;
interface LoanAnalyzer
{
boolean approve( in Account, in CreditCards);
}
```

- **CFML**

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<!--- Declare a "person" struct ---->
<cfset p = StructNew()>
<cfif IsStruct(p)>
<cfset p.pid = 1003232>
<cfset p.name = "Eduardo">
<cfset p.middle = "R">
<cfset p.last_name = "Doe">
</cfif>
<!---- Declare an "Account" struct --->
<cfset a = StructNew()>
<cfif IsStruct(a)>
<cfset a.person = p>
<cfset a.age = 34>
<cfset a.income = 150120.50>
</cfif>
<!----- Declare a "CreditCards" sequence --->
<cfset cards = ArrayNew(1)>
<cfset cards[1] = 0> <!--- corresponds to Amex --->
<cfset cards[2] = 2> <!--- corresponds to MC --->
<cfset cards[3] = 4> <!--- corresponds to Diners --->
<!---- Creating a CORBA handle using the Naming Service---->
<cfset handle = CreateObject("CORBA", "FirstBostonBank/MA/Loans",
"NameService") >
<cfset ret=handle.approve(a, cards)>
<cfoutput>Account approval: #ret#</cfoutput>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

Using External Resources

- Sending and Receiving E-Mail
- Interacting with Microsoft Exchange Servers
- Interacting with Remote Servers
- Managing Files on the Server
- Using Event Gateways
- Using the Instant Messaging Event Gateways
- Using the SMS Event Gateway
- Using the FMS event gateway
- Using the Data Services Messaging Event Gateway
- Using the Data Management Event Gateway
- Creating Custom Event Gateways
- Using the ColdFusion Extensions for Eclipse

#back to top

© 2014 Adobe Systems Incorporated. All rights reserved.

Sending and Receiving E-Mail

You can add interactive e-mail features to your Adobe ColdFusion applications by using the cfmail and cfpop tags. This complete two-way interface to mail servers makes the ColdFusion e-mail capability a vital link to your users.

© 2014 Adobe Systems Incorporated. All rights reserved.

Using ColdFusion with mail servers

Adding e-mail to your ColdFusion applications lets you respond automatically to user requests. You can use e-mail in your ColdFusion applications in many different ways, including the following:

- Trigger e-mail messages based on requests or orders from users.
- Allow users to request and receive additional information or documents through e-mail.
- Confirm customer information based on order entries or updates.
- Send invoices or reminders, using information pulled from database queries.
  ColdFusion offers several ways to integrate e-mail into your applications. To send e-mail, you generally use the Simple Mail Transfer Protocol (SMTP). To receive e-mail, you use the Post Office Protocol (POP) to retrieve e-mail from the mail server. To use e-mail messaging in your ColdFusion applications, you must have access to an SMTP server, a valid POP account, or both.
  In your ColdFusion application pages, you use the cfmail and cfpop tags to send and receive e-mail, respectively.

How ColdFusion sends mail

The ColdFusion implementation of SMTP mail uses a spooled architecture. If you select to spool mail on the Mail page in the ColdFusion Administrator, when an application page processes a cfmail tag, the messages that are generated are not sent immediately. Instead, they are spooled to disk and processed in the background. This architecture has two advantages:

- End users of your application are not required to wait for SMTP processing to complete before a page returns to them. This design is especially useful when a user action causes the sending of more than a handful of messages.
- Messages sent using `cfmail` are delivered reliably, even in the presence of unanticipated events like power outages or server crashes.
  You can set how frequently ColdFusion checks for spooled mail messages on the Mail page in the ColdFusion Administrator. If ColdFusion is busy or has a large existing queue of messages, however, delivery can occur after the spool interval.
  Some ColdFusion editions have advanced spooling options that let you fine-tune how ColdFusion sends mail. For more information, see *Configuring and Administering ColdFusion*.

Error logging and undelivered messages

ColdFusion logs all errors that occur during SMTP message processing to the file mail.log in the ColdFusion log directory. The log entries contain the date and time of the error as well as diagnostic information about why the error occurred.
If a message is not delivered because of an error, ColdFusion writes it to this directory:

- In Windows: _CFusion_\Mail\Undelivr
- On UNIX: /opt/*coldfusion*/mail/undelivr
  The error log entry that corresponds to the undelivered message contains the name of the file written to the UnDelivr (or undelivr) directory.

> **Note**
>
> To have ColdFusion try to resend a message that it could not deliver, move the message file from the Undelivr directory to the Spool directory.

For more information about the mail logging settings in the ColdFusion Administrator, see *Configuring and Administering ColdFusion*.

© 2014 Adobe Systems Incorporated. All rights reserved.

Sending e-mail messages

Before you configure ColdFusion to send e-mail messages, you must have access to an SMTP e-mail server. Also, before you run application pages that access the e-mail server, you can configure the ColdFusion Administrator to use the SMTP server. If you must override the ColdFusion Administrator SMTP server setting for any messages, you can specify a new mail server in the `server` attribute of the cfmail tag.

Configure ColdFusion for e-mail

1. In the ColdFusion Administrator, select Server Settings > Mail.
2. In the Mail Server box, enter the name or IP address of your SMTP mail server.
3. (Optional) Change the Server Port and Connection Timeout default settings.
4. Select the Verify Mail Server Connection option to make sure ColdFusion can access your mail server.
5. If your mail server does not use port 25, the default, SMTP port, change the Server Port default settings.
6. Depending on your ColdFusion edition, the Mail page in the Administrator has additional options that you can use to configure and optimize ColdFusion mail behavior. Select these options as appropriate.
7. Click Submit Changes.ColdFusion saves the settings. The page displays a message indicating success or failure for connecting to the server.
   ColdFusion Enterprise edition includes additional mail spooling and delivery features. For more information on these features, and for information on the ColdFusion Administrator mail settings, see *Configuring and Administering ColdFusion*.

Sending SMTP e-mail with the cfmail tag

The `cfmail` tag provides support for sending SMTP e-mail from within ColdFusion applications. The `cfmail` tag is like the cfoutput tag, except that `cfmail` outputs the generated text as an SMTP mail message rather than to a page. The `cfmail` tag supports all the attributes and commands that you use with `cfoutput`, including `query`. The following table describes basic `cfmail` tag attributes that you could use to send a simple e-mail message. For a complete list of attributes, see the cfmail description in the *CFML Reference*.

Attribute	Description
`subject`	The subject of the message.
`from`	The e-mail address of the sender.
`to`	The e-mail address of the recipient. Use a comma-delimited list to specify multiple recipients.
`cc`	(Optional) The e-mail address of a carbon copy recipient. The recipient address is visible to other recipients. Use a comma-delimited list to specify multiple cc recipients.
`bcc`	(Optional) The e-mail address of a blind carbon copy recipient. The recipient address is not visible to other recipients. Use a comma-delimited list to specify multiple bcc recipients.

Send a simple e-mail message

1. Create a ColdFusion page with the following content:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<html>
<head>
 <title>Sending a simple e-mail</title>
</head>
<body>
<h1>Sample e-mail</h1>
<cfmail
 from="Sender@Company.com"
 to="#URL.email#"
 subject="Sample e-mail from ColdFusion">

This is a sample e-mail message to show basic e-mail capability.

</cfmail>
The e-mail was sent.

</body>
</html>
```

2. Save the file as send_mail.cfm in the myapps directory under your *web_root* directory.
3. Open your browser and enter the following URL: *http://localhost:8500/myapps/send_mail.cfm?email=*_myname@mycompany.com_(Replace myname@mycompany.com with your e-mail address.)The page sends the e-mail message to you, through your SMTP server.

Note

If you do not receive an e-mail message, check whether you have configured ColdFusion to work with your SMTP server; for more information, see Sending e-mail messages.

The `cfmail` tag has many options that let you customize your mail or control how it is sent. For a description of all attributes, including options to wrap mail text at a specified column, specify the mail character encoding, and specify the mail server, user name, and password, see the cfmail description in the *CFML Reference*.

Sending HTML e-mail

If you know all the mail recipients use mail applications that are capable of reading and interpreting HTML code in a mail message, you can use the cfmail tag to send an HTML message. The `cfmail` tag `type="HTML"` attribute informs the receiving e-mail client that the message contains embedded HTML tags that must be processed. For an example that sends HTML mail, see *Including images in a message* in Using the cfmailparam tag.

Sending multipart mail messages

The cfmailpart tag lets you create multipart mail messages, with each part having a different MIME type or character set. For example, if you do not know that all recipients can interpret HTML mail messages, you can send your message as a multipart mail with a text part and an HTML part. To do so, use two `cfmailpart` tags, one with the HTML version of the message and one with the plain text message, as shown in the following example. To test this example, replace the `To` attribute value with a valid e-mail address, save and run the page, and check the incoming e-mail at the address you entered.

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<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>
```

Note

In the HTML version of the message, escape any number signs, such as those used to specify colors, by using two # characters; for example, `bgcolor="##C5D9E5"`.

© 2014 Adobe Systems Incorporated. All rights reserved.

Sample uses of the cfmail tag

An application page containing the cfmail tag dynamically generates e-mail messages based on the tag settings. Some of the tasks that you can accomplish with `cfmail` include the following:

- Sending a mail message in which the data the user enters in an HTML form determine the recipient and contents
- Using a query to send a mail message to a database-driven list of recipients
- Using a query to send a customized mail message, such as a billing statement, to a list of recipients that is dynamically populated from a database

Sending form-based e-mail

In the following example, the contents of a customer inquiry form submittal are forwarded to the marketing department. You could also use the same application page to insert the customer inquiry into the database. You include the following code on your form so that it executes when users enter their information and submit the form:

```
<cfmail
 from="#Form.EMailAddress#"
 to="marketing@MyCompany.com,sales@MyCompany.com"
 subject="Customer Inquiry">

A customer inquiry was posted to our website:

Name: #Form.FirstName# #Form.LastName#
Subject: #Form.Subject#

#Form.InquiryText#
</cfmail>
```

Sending query-based e-mail

In the following example, a query (ProductRequests) retrieves a list of the customers who inquired about a product during the previous seven days. ColdFusion sends the list, with an appropriate header and footer, to the marketing department:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cfmail
 query="ProductRequests"
 from="webmaster@MyCompany.com"
 to="marketing@MyCompany.com"
 subject="Widget status report">

Here is a list of people who have inquired about
MyCompany Widgets during the previous seven days:

<cfoutput>
#ProductRequests.FirstName# #ProductRequests.LastName# (#ProductRequests.Company#) -
#ProductRequests.EMailAddress#&##013;
</cfoutput>

Regards,
The webmaster
webmaster@MyCompany.com

</cfmail>
```

Reviewing the code

The following table describes the code:

Code	Description

`<cfoutput>`
`#ProductRequests.FirstName#`
`#ProductRequests.LastName#`
`(#ProductRequests.Company#) -`
`#ProductRequests.EMailAddress#&##013;`
`</cfoutput>`
	Presents a dynamic list embedded within a normal message, repeating for each row in the ProductRequests query. Because the `cfmail` tag specifies a query, the `cfoutput` tag does not use a `query` attribute. The `&##013;` forces a carriage return between output records.

Sending e-mail to multiple recipients

In addition to simply using a comma-delimited list in the `to` attribute of the cfmail tag, you can send e-mail to multiple recipients by using the `query` attribute of the `cfmail` tag. The following examples show how you can send the same message to multiple recipients and how you can customize each message for the recipient.

Sending a simple message to multiple recipients

In the following example, a query (BetaTesters) retrieves a list of people who are beta testing ColdFusion. This query then notifies each beta tester that a new release is available. The contents of the `cfmail` tag body are not dynamic. What is dynamic is the list of e-mail addresses to which the message is sent. Using the variable `#TesterEMail#,` which refers to the TesterEmail column in the Betas table, in the `to` attribute, enables the dynamic list:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cfquery name="BetaTesters" datasource="myDSN">
 SELECT * FROM BETAS
</cfquery>

<cfmail query="BetaTesters"
 from="beta@MyCompany.com"
 to="#BetaTesters.TesterEMail#"
 subject="Widget Beta Four Available">

To all Widget beta testers:

Widget Beta Four is now available
for downloading from the MyCompany site.
The URL for the download is:

http://beta.mycompany.com

Regards,
Widget Technical Support
beta@MyCompany.com

</cfmail>
```

Customizing e-mail for multiple recipients

In the following example, a query (GetCustomers) retrieves the contact information for a list of customers. The query then sends an e-mail to each customer to verify that the contact information is still valid:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cfquery name="GetCustomers" datasource="myDSN">
 SELECT * FROM Customers
</cfquery>

<cfmail query="GetCustomers"
 from="service@MyCompany.com"
 to="#GetCustomers.EMail#"
 subject="Contact Info Verification">

Dear #GetCustomers.FirstName# -

We'd like to verify that our customer
database has the most up-to-date contact
information for your firm. Our current
information is as follows:

Company Name: #GetCustomers.Company#
Contact: #GetCustomers.FirstName# #GetCustomers.LastName#

Address:
 #GetCustomers.Address1#
 #GetCustomers.Address2#
 #GetCustomers.City#, #GetCustomers.State# #GetCustomers.Zip#

Phone: #GetCustomers.Phone#
Fax: #GetCustomers.Fax#
Home Page: #GetCustomers.HomePageURL#

Please let us know if any of this
information has changed, or if we must
get in touch with someone else in your
organization regarding this request.

Thanks,
Customer Service
service@MyCompany.com

</cfmail>
```

Reviewing the code

The following table describes the code and its function:

Code	Description

`<cfquery name="GetCustomers" datasource="myDSN">`
` SELECT * FROM Customers`
`</cfquery>`
	Retrieves all data from the Customers table into a query named GetCustomers.

© 2014 Adobe Systems Incorporated. All rights reserved.

`<cfmail query="GetCustomers"`
` from="service@MyCompany.com"`
` to="#GetCustomers.EMail#"`
` subject="Contact Info`
`Verification">`
	Uses the `to` attribute of `cfmail`, the `#GetCustomers.Email#` query column causes ColdFusion to send one message to the address listed in each row of the query. Therefore, the mail body does not use a `cfoutput` tag.

`Dear #GetCustomers.FirstName# -`
 	
`.....`
 	
`Company Name:`
`#GetCustomers.Company#`
`Contact: #GetCustomers.FirstName#`
`#GetCustomers.LastName#`
 	
`Address:`
` #GetCustomers.Address1#`
` #GetCustomers.Address2#`
` #GetCustomers.City#,`
`#GetCustomers.State#`
`#GetCustomers.Zip#`
 	
`Phone: #GetCustomers.Phone#`
`Fax: #GetCustomers.Fax#`
`Home Page:`
`#GetCustomers.HomePageURL#`
	Uses other query columns (`#GetCustomers.FirstName#`, `#GetCustomers.LastName#`, and so on) within the `cfmail` section to customize the contents of the message for each recipient.

Digitally signing e-mail

To add digital signature to your mail, specify the attributes `sign`, `keystore`, `keystorepassword`, `keyalias`, and `keypassword` as provided in the following example:

```
<cfmail from="Sender@Company.com" server="sendmail.myCo.com" sign="true"
keystore="C:\OpenSSL\bin\hello.jks" keystorepassword="digital"
to="Recepient@Company.com" keyalias="crypto" keypassword="signature" subject="Mail
with Digital Signature">
```

To add digital signature to all the mails you send, instead of adding the attributes to the tag, specify the settings in the Server Settings > Settings page of the ColdFusion Administrator.
The supported keystores are JKS and PKCS12.
Due to import control restrictions in various countries, the policy files (local_policy.jar and US_export_policy.jar) support only limited cryptography. If the key strength exceeds the limit, you might encounter the error suggesting that the keystore cannot be loaded. If you are from an eligible country, you can download the unlimited strength version of the policy files and replace the default cryptography JAR files with them. The files are available on the Java SDK web site.

© 2014 Adobe Systems Incorporated. All rights reserved.

Using the cfmailparam tag

You use the cfmailparam tag to include files in your message or add a custom header to an e-mail message. You can send files as attachments or display them inline in the message. You nest the `cfmailparam` tag within the cfmail tag.

Attaching files to a message

You can use one `cfmailparam` tag for each attachment, as the following example shows:

```
<cfmail from="daniel@MyCompany.com"
 to="jacob@YourCompany.com"
 subject="Requested Files">

Jake,

Here are the files you requested.

Regards,
Dan

<cfmailparam file="c:\widget_launch\photo_01.jpg">
<cfmailparam file="c:\widget_launch\press_release.doc">

</cfmail>
```

Use a fully qualified system path for the `file` attribute of `cfmailparam`. The file must be located on a drive on the ColdFusion server machine (or a location on the local network), not the browser machine.

Including images in a message

You can use the `cfmailparam` to include images from other files in an HTML message, as follows:

1. Place a `cfmailparam` tag for each image following the `cfmail` start tag.
2. In each `cfmailparam`tag, do the following
   - Set the `file` attribute to the location of the image.
   - Specify `disposition="inline"`
   - Set the `contentID` attribute to a unique identifier; for example, myImage1.
3. In the location in your HTML where you want the message included, use an `img`tag such as the following:

   ```
   <img src="cid:myImage1">
   ```

The following example shows a simple mail message with an inline image. In this case, the image is located between paragraphs, but you could include it directly inline with the text. To test this example, replace the `cfmail to` parameter with a valid e-mail address and change the `file` parameter to the path to a valid image.

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cfmail type="HTML"
  to = "Peter@myCo.com"
  from = "Paul@AnotherCo.com"
  subject = "Sample inline image">
 <cfmailparam file="C:\Inetpub\wwwroot\web.gif"
  disposition="inline"
  contentID="image1">
 <P>There should be an image here</p>
 <img src="cid:image1">
 <p> This text follows the picture</p>
</cfmail>
```

Adding a custom header to a message

When the recipient of an e-mail message replies to the message, the reply is sent, by default, to the address specified in the From field of the original message. You can use the cfmailparam tag to provide a Reply-To e-mail address that tells the mail client to override the value in the From field. Using `cfmailparam`, the reply to the following example is addressed to widget_master@YourCompany.com:

```
<cfmail from="jacob@YourCompany.com"
 to="daniel@MyCompany.com"
 subject="Requested Files">
<cfmailparam name="Reply-To" value="widget_master@YourCompany.com">

Dan,
Thanks very much for the sending the widget press release and graphic.
I'm now the company's Widget Master and am accepting e-mail at
widget_master@YourCompany.com.

See you at Widget World 2002!

Jake
</cfmail>
```

Note

You can combine the two uses of `cfmailparam` within the same ColdFusion page. Write a separate `cfmailparam` tag for each header and for each attached file.

© 2014 Adobe Systems Incorporated. All rights reserved.

Receiving e-mail messages

You create ColdFusion pages to access a Post Office Protocol (POP) server to retrieve e-mail message information. ColdFusion can then display the messages (or just header information), write information to a database, or perform other actions.

The cfpop tag lets you add Internet mail client features and e-mail consolidation to applications. Although a conventional mail client provides an adequate interface for personal mail, in many cases an alternative interface to some mailboxes is advantageous. You use `cfpop` to develop targeted mail clients to suit the specific needs of a wide range of applications. The `cfpop` tag does not work with the other major e-mail protocol, Internet Mail Access Protocol (IMAP).

Here are three instances in which implementing POP mail makes sense:

- If your site has generic mailboxes that more than one person reads (*sales@yourcompany.com*), it can be more efficient to construct a ColdFusion mail front end to supplement individual user mail clients.
- In many applications, you can automate mail processing when the mail is formatted to serve a particular purpose; for example, when subscribing to a list server.
- If you want to save e-mail messages to a database.
  Using `cfpop` with your POP server is like running a query on your mailbox contents. You set its `action` attribute to retrieve either headers (using the `GetHeaderOnly` value) or entire messages (using the `GetAll` value) and assign it a `name` value. You use the name to access the recordset that `cfpop` returns, for example, when using the cfoutput tag. To access a POP server, you also must define the `server`, `username`, and `password` attributes.

> **Note**
>
> If the `cfpop` tag encounters an error, such as an improperly formatted e-mail message, when retrieving messages, it tries to ignore the error; it returns empty fields in the result structure and retrieves any available messages.

For more information on the cfpop tag syntax and variables, see the *CFML Reference*.

Using the cfpop tag

Use the following steps to add POP mail to your application.

Implement the cfpop tag in your application

1. Choose the mailboxes to access within your ColdFusion application.
2. Determine which mail message components you must process: message header, message body, attachments, and so on.
3. Decide whether you must store the retrieved messages in a database.
4. Decide whether to delete messages from the POP server after you retrieve them.
5. Incorporate the cfpop tag in your application and create a user interface for accessing a mailbox.
6. Build an application page to handle the output. Retrieved messages can include characters that do not display properly in the browser.
   You use the `cfoutput` tag with the HTMLCodeFormat and HTMLEditFormat functions to control output to the browser. These functions convert characters with special meanings in HTML, such as the less than (<), greater than (>), and ampersand (&) symbols, into HTML-escaped characters, such as <, >, and &. The `HTMLCodeFormat` tag also surrounds the text in a `pre` tag block.

The cfpop query variables

Like any ColdFusion query, each `cfpop` query returns variables that provide information about the record:

- **RecordCount** The total number of records returned by the query.

© 2014 Adobe Systems Incorporated. All rights reserved.

- **ColumnList** A list of the headings of the columns that the query returns.
- **CurrentRow** The current row of the query that `cfoutput`, or `cfloop` in a query-driven loop, is processing.
  The query includes one variable that the `cfquery` tag does not return: the UID variable contains the unique identifier of the e-mail message file.
  You can reference these properties in a `cfoutput` tag by prefixing the query variable with the query name in the `name` attribute of `cfpop`:

```
<cfoutput>
This operation returned #Sample.RecordCount# messages.
</cfoutput>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

Handling POP mail

You use cfpop tag to manage mail. You can specify the messages to act on. You can get message headers, messages, and attachments, and you can delete messages.

Specifying the message or messages

For all cfpop actions, you can tell the tag to perform the action on all messages, or to do it on selected messages. To operate on all messages, for example to get all message headers, do not specify a `messageNumber` or `UID` attribute. To operate on specific messages, for example, to delete three selected messages, specify a `messageNumber` or `UID` attribute with a comma-delimited list of messages to act on.

Retrieving message headers

To retrieve message headers without getting the messages, specify `action="GetHeaderOnly"` in the cfpop tag. Whether you use `cfpop` to retrieve the header or the entire message, ColdFusion returns a query object that contains one row for each message in the specified mailbox. you specify the query object name in the `cfpop` tag `name` attribute. The query has the following fields:

- date
- from
- header (A string with all the mail header fields, including entries that have separate fields in the query object)
- messageNumber (The sequential number of the message in the POP server; identical to the row number of the entry in the query object)
- messageID (The mail header Message-ID field)
- replyTo
- subject
- cc
- to
- UID (The mail header X-UID field)
  The `cfpop` tag with the `getHeaderOnly` attribute retrieves any file attachments if you specify an `attachmentPath` attribute; otherwise, it does not get the attachments, and the attachmentfiles column contains empty strings.

Retrieve only the message header

1. Create a ColdFusion page with the following content:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<html>
<head>
<title>POP Mail Message Header Example</title>
</head>

<body>
<h2>This example retrieves message header information:</h2>

<cfpop server="mail.company.com"
 username=#myusername#
 password=#mypassword#
 action="GetHeaderOnly"
 name="Sample">

<cfoutput query="Sample">
 MessageNumber: #HTMLEditFormat(Sample.messageNumber)# <br>
 To: #HTMLEditFormat(Sample.to)# <br>
 From: #HTMLEditFormat(Sample.from)# <br>
 Subject: #HTMLEditFormat(Sample.subject)# <br>
 Date: #HTMLEditFormat(Sample.date)#<br>
  Cc: #HTMLEditFormat(Sample.cc)# <br>
 ReplyTo: #HTMLEditFormat(Sample.replyTo)# <br><br>
</cfoutput>

</body>
</html>
```

2. Edit the following lines so that they use valid values for your POP mail server, user name, and password:

```
<cfpop server="mail.company.com"
 username=#myusername#
 password=#mypassword#
```

3. Save the file as header_only.cfm in the myapps directory under your *web_root* and view it in your web browser:This code retrieves the message headers and stores them in a `cfpop` recordset called Sample. For more information about working with recordset data, see Using Query of Queries.
   The HTMLCodeFormat function replaces characters that have meaning in HTML, such as the less than (<) and greater than (>) signs that can surround detailed e-mail address information, with escaped characters such as < and >.
   In addition, you can process the date returned by cfpop with the ParseDateTime function, which accepts an argument for converting POP date/time objects into a CFML date-time object.
   You can reference any of these columns in a cfoutput tag, as the following example shows:

```
<cfoutput>
 #ParseDateTime(queryname.date, "POP")#
 #HTMLCodeFormat(queryname.from)#
 #HTMLCodeFormat(queryname.messageNumber)#
</cfoutput>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

Retrieving messages

When you use the cfpop tag with `action="GetAll"`, ColdFusion returns the same columns as with `getheaderonly`, plus the following additional columns:

- attachments (A tab-delimited list of attachment filenames)
- attachmentfiles (A tab-delimited list of paths to the attachment files retrieved to the local server, if any. You get the files only if you specify an `attachmentpath` attribute.)
- body
- htmlbody
- textbody
  If the message is multipart, the htmlbody and textbody fields contain the contents of the HTML and plain text parts, and the body field has the first part in the message. If the message has only one part, the body contains the message, and either the htmlbody or textbody field, depending on the message type, also has a copy of the message.

Retrieve entire messages

1. Create a ColdFusion page with the following content:

```
<html>
<head><title>POP Mail Message Body Example</title></head>

<body>
<h2>This example adds retrieval of the message body:</h2>
<cfpop server="mail.company.com"
 username=#myusername#
 password=#mypassword#
 action="GetAll"
 name="Sample">

<cfoutput query="Sample">
 MessageNumber: #HTMLEditFormat(Sample.messageNumber)# <br>
 To: #Sample.to# <br>
 From: #HTMLEditFormat(Sample.from)# <br>
 Subject: #HTMLEditFormat(Sample.subject)# <br>
 Date: #HTMLEditFormat(Sample.date)#<br>
 Cc: #HTMLEditFormat(Sample.cc)# <br>
 ReplyTo: #HTMLEditFormat(Sample.replyTo)# <br>
 <br>
 Body:<br>
 #Sample.body#<br>
 <br>
 Header:<br>
 #HTMLCodeFormat(Sample.header)#<br>
 <hr>
 </cfoutput>

</body>
</html>
```

2. Edit the following lines so that they use valid values for your POP mail server, user name, and password:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cfpop server="mail.company.com"
 username=#myusername#
 password=#mypassword#
```

3. Save the file as header_body.cfm in the myapps directory under your *web_root* and view it in your web browser:
   This example does not use a CFML function to encode the body contents. As a result, the browser displays the formatted message as you would normally see it in a mail program that supports HTML messages.

Retrieving messages and attachments

When you use the cfpop tag with an `attachmentpath` attribute to specify the directory in which to store attachments, ColdFusion retrieves any attachment files from the POP server and saves them in the specified directory. The `cfpop` tag fills the attachmentfiles field with a tab-separated list of the locations of the attachment files. Use the `cffile` tag to delete these temporary files when they are no longer needed. ColdFusion creates the directory if it does not exist. (ColdFusion must have the appropriate rights on the system to create the directory.) If a message has no attachments, the `attachments` and `attachmentfiles` columns contain empty strings.

> **Note**
>
> CFML does not provide a way to change the name of a mail attachment returned by `cfpop` befor e it tries to save the file. If the attachment name is invalid for the file system on which ColdFusion is running, the attachment cannot be saved.

Retrieve all parts of a message, including attachments

1. Create a ColdFusion page with the following content:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<html>
<head>
<title>POP Mail Message Attachment Example</title>
</head>

<body>
<h2>This example retrieves message header,
body, and all attachments:</h2>

<cfpop server="mail.company.com"
 username=#myusername#
 password=#mypassword#
 action="GetAll"
 attachmentpath="c:\temp\attachments"
 name="Sample">

<cfoutput query="Sample">
 MessageNumber: #HTMLEditFormat(Sample.MessageNumber)# <br>
 To: #HTMLEditFormat(Sample.to)# <br>
 From: #HTMLEditFormat(Sample.from)# <br>
 Subject: #HTMLEditFormat(Sample.subject)# <br>
 Date: #HTMLEditFormat(Sample.date)# <br>
 Cc: #HTMLEditFormat(Sample.cc)# <br>
 ReplyTo: #HTMLEditFormat(Sample.ReplyTo)# <br>
 Attachments: #HTMLEditFormat(Sample.Attachments)# <br>
 Attachment Files: #HTMLEditFormat(Sample.AttachmentFiles)# <br>
 <br>
 Body:<br>
 #Sample.body# <br>
 <br>
 Header:<br>
 HTMLCodeFormat(Sample.header)# <br>
 <hr>
</cfoutput>

</body>
</html>
```

2. Edit the following lines so that they use valid values for your POP mail server, user name, and password:

```
<cfpop server="mail.company.com"
 username=#myusername#
 password=#mypassword#
```

3. Save the file as header_body_att.cfm in the myapps directory under your *web_root* and view it in your web browser:

Note

To avoid duplicate filenames when saving attachments, set the `generateUniqueFilenames` a ttribute of `cfpop` to `Yes`.

© 2014 Adobe Systems Incorporated. All rights reserved.

Deleting messages

Using the cfpop tag to delete a message permanently removes it from the server. By default, retrieved messages remain on the POP mail server. To delete the messages, set the `action` attribute of the `cfpop` tag to Delete. Use the `messagenumber` attribute to specify the messages to delete; omit the attribute to delete all the user's messages from the server.

> **Note**
>
> Message numbers are reassigned at the end of every POP mail server communication that contains a delete action. For example, if you retrieve four messages from a POP mail server, the server returns the message numbers 1,2,3,4. If you delete messages 1 and 2 with a single `cfpo` p tag, messages 3 and 4 are assigned message numbers 1 and 2, respectively.

Delete messages

1. Create a ColdFusion page with the following content:

```
<html>
<head>
<title>POP Mail Message Delete Example</title>
</head>

<body>
<h2>This example deletes messages:</h2>

<cfpop server="mail.company.com"
 username=#username#
 password=#password#
 action="Delete"
 messagenumber="1,2,3">

</body>
</html>
```

2. Edit the following lines so that they use valid values for your POP mail server, user name, and password:

```
<cfpop server="mail.company.com"
 username=#myusername#
 password=#mypassword#
```

3. Save the file as message_delete.cfm in the myapps directory under your *web_root* and view the file in your web browser.
4. Run the header_only.cfm page that you created to confirm that the messages have been deleted.

> When you view this page in your web browser, ColdFusion_ immediately deletes the messages from the POP server.

#back to top

© 2014 Adobe Systems Incorporated. All rights reserved.

Interacting with Microsoft Exchange Servers

You can use Adobe ColdFusion to interact with Microsoft Exchange servers to send, get, and manage mail; and to create, get, and manage calendar events, connections, and tasks.

© 2014 Adobe Systems Incorporated. All rights reserved.

Using ColdFusion with Microsoft Exchange servers

ColdFusion can interact with the Microsoft Exchange server to perform the following actions:

Item	Actions
Mail messages	get, get attachments, get meeting information, move to a different folder, delete, delete attachments, set properties
Calendar events	create, get, get attachments, delete, delete attachments, modify, respond
Contacts	create, get, get attachments, delete, delete attachments, modify
Tasks	create, get, get attachments, delete, delete attachments, modify

To perform these actions, you use the following ColdFusion tags:

Tag	Purpose
cfexchangeconnection	Opens and closes persistent connections between an application and the Exchange server.Gets information about subfolders of the Inbox.
cfexchangecalendar	Creates, gets, and manages calendar events.
cfexchangecontact	Creates, gets, and manages contacts.
cfexchangemail	Gets and manages mail messages. Does not send mail.
cfmail	Sends mail to the exchange server.
cfexchangetask	Creates, gets, and manages tasks.
cfexchangefilter	Specifies the criteria to get specific items. Used only as a child of the `cfexchangecalendar`, `cfexchangecontact`, `cfexchangemail`, and `cfexchangetask` tags that specify the `get` action.

The following list describes a few of the activities you can do using ColdFusion with the Exchange server:

- Build a customized Exchange web client interface.
- View information about upcoming tasks.
- Create mailing lists based on contact entries.
- Automatically add tasks to users' task lists based on new bug reports or customer contacts.
- Schedule meetings and appointments.

© 2014 Adobe Systems Incorporated. All rights reserved.

- Show and manage meeting attendee availability.

© 2014 Adobe Systems Incorporated. All rights reserved.

Managing connections to the Exchange server

To communicate with an Exchange server, establish a connection with the server. The connection can use the HTTP protocol or the HTTPS protocol. By default, ColdFusion connects to the mailbox that belongs to the login user name, but you can also connect to any mailbox whose owner has delegated access rights to the login user name. You can also access the server by using a proxy host.

> **Note**
>
> To establish any connection, the Exchange server must grant the login user Outlook web access. For information on how to enable this access, see *Enabling Outlook web access* below.

Connections to the server can be persistent or transient:

- A *persistent connection* lasts until you explicitly close it. Persistent connections let you use a single connection for multiple tasks, which saves the processing overhead of opening and closing a separate connection for each interaction with the Exchange server.
- A *transient connection* lasts for the duration of the tag that interacts with the Exchange server. Transient connections are a useful technique on ColdFusion pages where you only have to access the Exchange server for a single tag; for example, where you only get a set of contacts.

Enabling access to the Exchange server

To enable access to the Exchange server, ensure the following:

- The Exchange server, Exchange access, and WebDav access are configured in IIS.
- The Exchange server enables Outlook web access to all login users.
- If you are using HTTPS to log into the exchange server, you have a valid client certificate in the JRE certificate store.

Ensure that IIS is configured for access to the Exchange server

1. Open the IIS manager from the Administrative Tools control panel on the machine where the Exchange server is installed.
2. Expand the Web Sites node in the tree on the left pane. If you see Exchange there, the web application is configured for Exchange. If you do not see it, follow the Microsoft instructions for configuring Exchange in the website
3. Click the Web Service Extension node in the tree on the left pane. The right pane shows Web Service Extensions and their status. Make sure that Microsoft Exchange Server and WebDav entries are both allowed. If either entry is prohibited, select it and click the Allow button.

Enabling Outlook web access

To establish any connection, the Exchange server must grant the login user Outlook web access.

Check and grant web access

1. In the Exchange administrator, open Administrative Tools > Active Directory Users and Computers > *your domain name* > users.
2. Right-click the user whose ID you use to establish connections.
3. Select the Exchange Features tab.
4. In the Protocols section, enable the Outlook Web Access entry if it is disabled.

Enabling HTTPS access to the Exchange server

To enable HTTPS access from ColdFusion to the Exchange server you must

© 2014 Adobe Systems Incorporated. All rights reserved.

- Enable SSL on the Exchange server system
- Ensure that the JRE certificate store has a valid client certificate

Enabling SSL on the Exchange server system

Use the following steps to enable SSL on the Exchange server system:

1. On the system where the Exchange server is installed, open the IIS manager from the Administrative Tools control panel.
2. In the tree on the left pane, expand the Web Sites node,
3. Right-click Exchange and ExchWeb in the expanded list and open the Web Site Properties dialog, then click the Directory Security tab.
4. In the Secure Communications section, click Edit to open the Secure Communications dialog. Select the Require secure channel (SSL) option, click OK, and click Apply.
   As an alternative to steps 3 and 4, you could do the following: Right-click Default Web Site. In Secure Communications->Edit, check the Require secure channel (SSL) option, click OK, and Click Apply. Select the nodes (for example Exchange) for which to enable SSL.

Enabling HTTPS access on the ColdFusion server

To use HTTPS to access the exchange server, you must have a valid client certificate in the JRE certificate store. If a known authority did not issue the certificate on the Exchange server, install a certificate. The Java certificate store already contains certificates from some authorities.
You can ask your system administrator to give you a certificate that you can install on the ColdFusion server, or you can do the following:

1. Open Outlook Web Access in Internet Explorer and go to File->Properties.
2. Click the certificates button.
3. Click the Details tab and the 'Copy To File' button on the tab. Then follow the wizard options to save the certificate.
   To install the certificate, run the following command using `keytool.exe`, which is in the jre\bin folder:

```
keytool.exe -importcert -file <path_to_certificate_file> -keystore
..\lib\security\cacerts
```

Note

The `keytool.exe` program requires you to enter a password. The default password is `change it`.

Using persistent connections

To open a persistent connection, you use the `cfexchangeconnection` tag and specify the `open` action, the server IP address or URL, the user name, and the name of the connection (which you use in subsequent tags to specify the connection). You typically also specify a password, and can specify several other attributes, including a proxy host or a delegate mailbox ID. For details, see cfexchangeconnection in the *CFML Reference*.
Persistent connections use HTTP or HTTPS protocol with the `keepAlive` property set to `true`. As a result, the connections do not automatically close at the end of an HTTP request or ColdFusion page. Close the connection when you are done using it. If you do not, ColdFusion retains the connection until an inactivity time-out period elapses, after which, ColdFusion recovers the connection resources.

© 2014 Adobe Systems Incorporated. All rights reserved.

Note

You can store a connection in a persistent scope, such as the Application scope, and reuse it on multiple pages. However, you get no advantage by doing so, because the connections are lightweight and you get no substantial performance gain if you use a persistent scope.

The following example opens a connection, gets all mail sent from spamsource.com, and deletes the messages from the Exchange server:

```
<cfexchangeConnection
 action = "open"
 username = "#user1#"
 password = "#password1#"
 server = "#exchangeServerIP#"
 connection = "conn1">

<cfexchangemail action = "get" name = "spamMail" connection = "conn1">
 <cfexchangefilter name = "fromID" value = "spamsource.com">
</cfexchangemail>

<cfloop query="spamMail">
 <cfexchangemail action = "delete" connection = "conn1"
   uid = "#spamMail.uid#">
</cfloop>

<cfexchangeConnection
 action = "close"
 connection = "conn1">
```

Using transient connections

Transient connections last only as long as the tag that uses them takes to complete processing. To create a transient connection, you specify the connection directly in your action tag (such as `cfexchangetask`) by using the same attributes as you do in the `cfexchangeconnection` tag (except for the connection name).
The following example uses a transient connection to create a single task:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<!--- Create a structure with the task fields. --->
<cfscript>
 stask = StructNew();
 stask.Priority = "high";
 stask.Status = "Not_Started";
 stask.DueDate = "3:00 PM 09/14/2007";
 stask.Subject = "My New Task";
 stask.PercentCompleted = 0;
 Message = "Do this NOW!";
</cfscript>

<!--- Create the task by using a transient connection. --->
<cfexchangetask action = "create"
 username = "#user1#"
 password = "#password1#"
 server = "#exchangeServerIP#"
 task = "#stask#"
 result = "theUID">

<!--- Display the UID to confirm that the action completed. --->
<cfdump var = "#theUID#">
```

Accessing delegated accounts

In Exchange, one user can grant, or delegate, another user access rights to their account. Users can delegate reviewer (read-only), author (read/write), or editor (read-write-delete) rights to any combination of the calendar, contacts, Inbox, or task list.

> **Note**
>
> You cannot use ColdFusion to delegate access rights.

To access the delegator's account as a delegated user, specify the following information:

- Specify the delegated user's user name and password in the `username` and `password` attributes.
- Specify the mailbox name of the account that you are accessing in the `mailboxName` attribute.
  You can access the account in a `cfexchangeconnection` tag that opens a persistent connection, or in a ColdFusion Exchange tag that uses a transient connection.
  For example, if access rights to the docuser3 account are delegated to docuser4, you can use the `cfexchangeconnection` tag as in the following example to open a connection to the docuser3 account by using docuser4's credentials:

```
<cfexchangeconnection action="open"
 connection="theConnection"
 server="myexchangeserver.mycompany.com"
 username="docuser4"
 password="secret"
 mailboxName="docuser3">
```

You can use this connection for any activities that docuser3 has delegated to docuser4. If docuser3, for example,

© 2014 Adobe Systems Incorporated. All rights reserved.

has only delegated reviewer rights to the calendar, you can use this connection only with the `cfexchangecalendar` tag with `get` and `getAttachments` attributes.

© 2014 Adobe Systems Incorporated. All rights reserved.

Creating Exchange items

You can create Exchange events, contacts, and tasks by using the `cfexchangecalendar`, `cfexchangecontact`, or `cfexchangetask` tag, respectively, and specifying an `action` attribute value of `create`. You create mail messages by using the cfmail tag to send the message. For information on sending mail, see Sending and Receiving E-Mail.

When you create a calendar event, contact, or task, you specify the action, the connection information (persistent connection name or transient connection attributes) and an attribute that specifies a structure with the information you are adding. You can also specify a `result` variable that contains the value of the Exchange UID for the entry that you create. You can use this UID to identify the entry in tags that modify or delete the entry.

The name of the attribute that you use to specify the entry information varies with the tag you are using, as follows:

Tag	Attribute
cfexchangecalendar	event
cfexchangecontact	contact
cfexchangetask	task

Enclose in number signs (#) the variable that contains the details of the event, contact, or task data, as in the following example:

```
<cfexchangecalendar action="create" connection="myConn" event="#theEvent#"
 result="resultUID">
```

The contents of the entry information structure depend on the tag. For details of the structure contents, see cfexchangecalendar, cfexchangecontact, and cfexchangetask in the *CFML Reference*.

Note

To create an Exchange calendar appointment, create a calendar event and do not specify any required or optional attendees.

The following example lets a user enter information in a form and creates a contact on the Exchange server with the information:

```
<!--- Create a structure to hold the contact information. --->
<cfset sContact="#StructNew()#">

<!--- A self-submitting form for the contact information --->
<cfform format="flash" width="550" height="460">
 <cfformitem type="html"><b>Name</b></cfformitem>
 <cfformgroup type="horizontal" label="">
  <cfinput type="text" label="First" name="firstName" width="200">
  <cfinput type="text" label="Last" name="lastName" width="200">
 </cfformgroup>
 <cfformgroup type="VBox">
  <cfformitem type="html"><b>Address</b></cfformitem>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

```
    <cfinput type="text" label="Company" name="Company" width="435">
    <cfinput type="text" label="Street" name="street" width="435">
    <cfinput type="text" label="City" name="city" width="200">
    <cfselect name="state" label="State" width="100">
     <option value="CA">CA</option>
     <option value="MA">MA</option>
     <option value="WA">WA</option>
    </cfselect>
    <cfinput type="text" label="Country" name="Country" width="200"
      Value="U.S.A.">
    <cfformitem type="html"><b>Phone</b></cfformitem>
    <cfinput type="text" validate="telephone" label="Business"
      name="businessPhone" width="200">
    <cfinput type="text" validate="telephone" label="Mobile"
      name="cellPhone" width="200">
    <cfinput type="text" validate="telephone" label="Fax" name="fax"
      width="200">
    <cfformitem type="html"><b>Email</b></cfformitem>
    <cfinput type="text" validate="email" name="email" width="200">
   </cfformgroup>

   <cfinput type="Submit" name="submit" value="Submit" >
  </cfform>


  <!--- If the form was submitted, fill the contact structure from it. --->
  <cfif isDefined("Form.Submit")>
   <cfscript>
    sContact.FirstName=Form.firstName;
    sContact.Company=Form.company;
    sContact.LastName=Form.lastName;
    sContact.BusinessAddress.Street=Form.street;
    sContact.BusinessAddress.City=Form.city;
    sContact.BusinessAddress.State=Form.state;
    sContact.BusinessAddress.Country=Form.country;
    sContact.BusinessPhoneNumber=Form.businessPhone;
    sContact.MobilePhoneNumber=Form.cellPhone;
    sContact.BusinessFax=Form.fax;
    sContact.Email1=Form.email;
   </cfscript>

   <!--- Create the contact in Exchange --->
   <cfexchangecontact action="create"
    username ="#user1#"
    password="#password1#"
    server="#exchangeServerIP#"
    contact="#sContact#"
    result="theUID">

   <!--- Display a confirmation that the contact was added. --->
   <cfif isDefined("theUID")>
    <cfoutput>Contact Added. UID is#theUID#</cfoutput>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

```
  </cfif>
 </cfif>
```

For another example of creating items, which creates a task, see *Using transient connections* in Managing connections to the Exchange server.

© 2014 Adobe Systems Incorporated. All rights reserved.

Getting Exchange items and attachments

You can get calendar events, contacts, mail messages, and tasks from the Exchange server. You can also get attachments to these items.
Getting an exchange item and its attachments can require multiple operations.

- To get mail that is not directly in the Inbox, specify the path from the root of the mailbox to the mail folder, and you can get items from only a single mail folder at a time. You can use the cfexchangeconnection tag to get the names, paths, and sizes of all folders in a mailbox, and can use the results to iterate over the folders.
- To get an attachment to an item, you must first get the item, and then use the item UID to get its attachments.
- If an Exchange item contains a message with inline images, the images are available as attachments. You can get the attachments, use the attachment CID to locate the image in the message, and display the image inline.

> **Note**
>
> The `getattachment` action does not always populate the CID field for HTML mail that contains inline attachments, such as images. This problem occurs because some Exchange clients do not always set the CID values if the attachments are sent inline.

Getting and using folder names

To get the names of folders in the mailbox, or the subfolders of a particular folder, use the cfexchangeconnection tag with the `getSubfolders` action. This action returns a query with a row for each subfolder. The query has three columns:

- folder name
- full path from the mailbox to the folder, including the Inbox
- folder size, in bytes
  You can specify the folder whose subfolders you are getting and whether to recursively get all levels of subfolders.
  You can use a folder path from the `getSubfolders` action in the `cfexchangemail` tag `folder` attribute to specify the folder that contains the mail message that requires action. If you do not specify a folder, the `cfexchangemail` tag searches only the top level of the Inbox for the message on which to act.
  To perform operations on mail from multiple folders, including getting mail items or attachments, you can loop over the entries in the query returned by the `getSubfolders` action, as the following example shows. This example generates a report of all declined meeting messages in the Inbox and all its subfolders.

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<!--- Create a connection. --->
<cfexchangeConnection
 action="open"
 username ="#user2#"
 password="#password2#"
 server="#exchangeServerIP#"
 connection="conn1">

<!--- Get the names and paths to all subfolders. --->
<cfexchangeconnection action="getSubfolders" connection="conn1"
 name="folderInfo" folder="Inbox" recurse="yes">

<!--- Get the information from the Inbox top level.
   The getSubfolders results do not include an Inbox row. --->
 <cfexchangemail action="get" connection="conn1"
   name="theResponses">
  <cfexchangefilter name="MessageType" value="Meeting_Response">
 </cfexchangemail>

<!--- Use a query of queries to select only the declined meetings. --->
<!--- You cannot use cfexchangefilter to filter for the meeting response type. --->
 <cfquery dbtype="query" name="theResponses">
  SELECT * FROM theResponses
  WHERE MEETINGRESPONSE = 'Decline'
 </cfquery>

<!--- Loop through the subfolders and get the meeting responses from each
 folder. --->
<cfloop query="folderInfo">
 <cfexchangemail action="get" connection="conn1"
   name="#folderinfo.foldername#">
  <cfexchangefilter name="folder" value="#folderinfo.folderpath#">
  <cfexchangefilter name="MessageType" value="Meeting_Response">
 </cfexchangemail>

 <!--- Use the evaluate function to get the name of the folder. --->
 <cfset meetingData=evaluate(folderinfo.foldername)>
 <!--- Use a query of queries with a UNION clause to add this folder's
   results to the theResponses query. --->
 <cfquery dbtype="query" name="theResponses">
  SELECT * FROM meetingData
  WHERE MEETINGRESPONSE = 'Decline'
  UNION
  SELECT * FROM theResponses
 </cfquery>
</cfloop>

<!--- Close the connection. --->
<cfexchangeConnection
 action="close"
 connection="conn1">

<!--- Display the results. --->
<h3>The Declined Responses:</h3>
<cftable query="theResponses" colheaders="yes" border="yes">
 <cfcol header="From" text="#FROMID#">
 <cfcol header="Subject" text="#SUBJECT#">
 <cfcol header="Message" text="#MESSAGE#">
</cftable>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

Getting items

You get one or more events, contacts, mail messages, or tasks from the Exchange server by using a cfexchangecalendar, cfexchangecontact, cfexchangemail, or cfexchangetask tag, respectively, and specifying an `action` attribute value of `get`. ColdFusion returns the items in a query object that you specify in the `name` attribute. You determine the items to get by specifying selection conditions in {{cfexchange}}filter child tags. The code to get items from the Exchange server has the following pattern:

```
<cfexchange***
  action="get"
  name="results query object name"
  connection information>
 <cfexchangefilter
   name="filter type"
   value"filter value>
 <cfexchangefilter
   name="data/time filter type"
   from="start date/time"
   to="end date/time">
 .
 .
 .
</cfexchange>
```

The following rules determine how you get items:

- You can have zero or more `cfexchangefilter`tags.
  - If you do not specify a `maxrows` field in the structure specified by the `name` attribute, ColdFusion gets a maximum of 100 items. To get more items, specify a `maxrows` field value greater than 100.
  - If you specify multiple `cfexchangefilter` tags with *different* `name` attributes, ColdFusion gets all items that match all of the specified conditions.
  - If you specify multiple `cfexchangefilter` tags with *identical* `name` attributes ColdFusion gets the items that match only the last tag with the duplicate `name` attribute.
- The `name` attributes correspond to field names in the Exchange item records. The valid values for the `name` at tributes depend on the type of item you are getting. For detailed lists of the valid values, see the corresponding tag references in the *CFML Reference*.
- If the `name` attribute specifies a field that takes text or numeric information, you use the `value` attribute to specify the condition.
- If the `name` attribute specifies a field that takes a date, time, or date and time, you use the `from` and `to` attrib utes to specify the range. You can omit one of these attributes to specify an open-ended range, such as all dates up to and including December 1, 2007.
- Date ranges are inclusive. The selected items include ones with the specified `to` or `from` dates.
- You cannot use the empty string as a `value` attribute to search for an empty value. To find entries where a particular field has an empty value, get all entries and use a query of queries to filter the results to include only entries where the field is empty.
- In fields that take text strings such as Message and or Subject, ColdFusion returns items that contain the exact phrase that you specify in the `value` attribute.
- When you use the `cfexchangemail` tag, ColdFusion gets only items a single folder. If you include a filter for a folder, ColdFusion gets items that are directly in the Inbox only and does not search any subfolders. For an example of getting information from multiple folders, see *Getting and using folder names* above.

© 2014 Adobe Systems Incorporated. All rights reserved.

When ColdFusion gets the results, it creates the query object specified in the `name` attribute, if it does not exist, and populates each row with a single item such as a mail message. The query columns depend on the type of item. For example, a mail message has `FromID` and `ToID` fields, and a contact has `FirstName` and `LastName` fields. For detailed information on the returned structures, see the corresponding tag in the *CFML Reference*.

The query results for all types of items have two columns:

- A `UID` column with the unique ID of the item. You use this value to specify the item when you delete, modify, or (for calendar entries) respond to it. You also use the UID value to get the item attachments.
- A `HasAttachments` column with a Boolean value specifying whether the item has any attachments. If this field is true, you can use the `getAttachments` action to get the attachments.

The following example gets the mail messages that were sent during the last week to the docuser1 user from any e-mail address that includes adobe.com. To keep this code short, the example uses the `cfdump` tag to show the results.

```
<cfset rightNow = Now()>
<cfset lastWeek = DateAdd("d","-7", rightNow)>

<cfexchangemail action="get" name="weeksMail"
  username ="#user1#" password="#password1#"
  server="#exchangeServerIP#">
 <cfexchangefilter name="FromID" value="adobe.com">
 <cfexchangefilter name="TimeSent" from="#lastWeek#" to="#rightNow#">
</cfexchangemail>

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

Getting item attachments

To get the attachments to an Exchange contact, event, message, or task, use a ColdFusion Exchange tag with a `getAttachments` action. Also specify the following information in the tag:

- The UID of the message that contains the attachment or attachments.
- The name of the query to hold the information about the returned attachments. When the tag completes processing, the query object contains one record for each retrieved attachment. The query has six columns that contain the filename, complete path to the saved attachment file, MIME type, file size, CID value (or an empty string) and an indicator that shows whether the attachment is a message.
- The path where the attachment is saved. (If you omit the path, ColdFusion does not get the attachments, but does get the information about the attachments.)
- Optionally, whether to create unique filenames by appending numbers to the names when two or more attachments have the same names. (The default is to not create unique filenames.)

The following ColdFusion Exchange tag gets all attachments to the message identified by the `theUID` variable, saves them in the C:/temp/cf_files/attachments directory, and stores information about the attachments in the `attachInfo` structure:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cfexchangemail action="getattachments"
 connection="myconn1"
 uid="#theUID#"
 name="#attachInfo#"
 attachmentPath="C:/temp/cf_files/attachments"
 generateUniqueFilenames="true">
```

To get message attachments, you must have the UID of the message and know that the message has attachments. Use a ColdFusion Exchange tag, such as `cfexchangemail`, with the `get` action to determine this information. When the tag completes processing, the query specified by the `name` attribute includes the following columns:

- The `HasAttachments` field is `true` if a message has one or more attachments
- The `UID` field contains the Exchange UID of the item. The exact UID format depends on the type of item; event, contact, message, or task.
  You can use these fields in your decision logic that determines whether to get attachments for a message and determines the message UID.
  The following example gets the attachments to all mail messages from docuser2 in the last week. It places each messages attachments in a directory whose name is the hexadecimal part of the message UID. For each message with attachments, the application reports subject and date of the message, followed by a table listing the messages attachments. The table includes the attachment name, MIME type, and size.
  Notice that if a message has multiple attachments with the same name, the attachment information query always lists the attachments with their original, duplicate names, even if you specify generateUniqueFilenames="true". The generateUniqueFilenames attribute only affects the names of the files on disk. The `attachmentFilePath` column of the attachment information structure does have the unique filenames, however.

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cfset rightNow = Now()>
<cfset lastWeek = DateAdd("d","-7", rightNow)>

<cfexchangeconnection
 action="open"
 username ="#user1#"
 password="#password1#"
 server="#exchangeServerIP#"
 connection="conn1">

<cfexchangemail action="get" folder="Inbox/MailTest" name="weeksMail"
  connection="conn1">
 <cfexchangefilter name="FromID" value="docuser2">
 <cfexchangefilter name="TimeSent" from="#lastWeek#" to="#rightNow#">
</cfexchangemail>

<cfloop query="weeksMail">
 <cfif weeksmail.HasAttachment>
  <!--- The UID is surrounded in <> characters and has an @ character.
  Extract the hexadecimal number part for use as a directory name. --->
  <cfset atpos=Find('@', weeksMail.UID)>
  <cfset shortUID=Mid(weeksMail.UID, 2, atpos-2)>

  <cfexchangemail action="getAttachments"
   connection="conn1"
   folder="Inbox/MailTest"
   uid="#weeksMail.uid#"
   name="attachData"
   attachmentPath="C:/temp/cf_files/attachments/#shortUID#"
   generateUniqueFilenames="true">

  <cfoutput>
   Directory #shortUID# contains these attachments to the
   following message:<br />
   Subject: #weeksMail.Subject#<br />
   Sent: #dateFormat(weeksmail.TimeSent)#<br />
   <cftable query="attachData" colheaders="true">
    <cfcol header="Filename" text="#attachmentFilename#">
    <cfcol header="Size" text="#size#">
    <cfcol header="MIME type" text="#mimeType#">
   </cftable>
  </cfoutput>

 </cfif>
</cfloop>

<cfexchangeconnection
 action="close"
 connection="conn1">
```

Displaying images inline

If an HTML message includes inline images, the Exchange server saves the images as attachments. Take the following steps to display the images in the retrieved message:

© 2014 Adobe Systems Incorporated. All rights reserved.

1. Use `cfexchangemail` tag `get` action to get the mail message.
2. Use `cfexchangemail` tag `getattachments` action to get the message attachments. Specify the UID of the mail message you got in the previous step. Also specify an attachmentPath attribute value that is under your web root, so that you can access the saved files by using a URL.
3. Search through the `HTMLMessage` field text that you got in step 1 and find the image items. Get the CID (content ID) value for each image.
4. Search the attachments query that you got in step 1. For each row with a `CID` column value that you got in step 3, get the corresponding `attachmentFilePath` column value.
5. Replace every `img` tag `src` attribute value with the `attachmentFilePath` field value that corresponds to the cid value.
6. Display the resulting HTML.
   The following example shows how to display a message with an inline image by retrieving the image from the attachments.

```
<!--- Open the connection to the Exchange server. --->
<cfexchangeconnection
 action="open"
 username = "#user1#"
 password = "#password1#"
 server = "#exchangeServerIP#"
 connection = "testconn">

<!--- Get the mail message. --->
<cfexchangeMail action="get" connection ="testconn" name="getMail">
 <cfexchangeFilter name="Subject" value="sample inline image">
</cfexchangeMail>

<cfdump var="#getMail#">

<!--- The following code assumes we found only one matching message. --->
<cfoutput query="getMail">
 <cfset theUID = #getMail.UID#>
 <cfset htmlmessage = getMail.htmlmessage>
</cfoutput>

<!--- Get the message attachments. --->
<CFExchangeMail action="getAttachments" UID ="#theUID#" connection="testconn"
name="attachments"
attachmentPath="C:\ColdFusion8\wwwroot\My_Stuff\cfexchange\Book\attachments"
generateuniquefilenames="no">

<!--- Extract the image names from the mail message --->
<!--- Initialize the index into the message used in finding --->
<cfset findstart = 1>
<!--- Use an index loop to find all image source entries in the message --->
<!--- This example supports up to 25 inline images --->
<cfloop index="i" from="1" to="25">
 <!--- find a cid: entry --->
 <cfset stringStart[i] = Find('"cid:', htmlmessage, findstart)>

 <!--- Exit the loop if no match was found --->
 <cfif (stringstart[i] EQ 0)>
  <cfbreak>
 </cfif>
 <!--- Increment the string index used in finding images. --->
 <cfset findstart = stringstart[i] +5 >
```

© 2014 Adobe Systems Incorporated. All rights reserved.

```
  <!--- Get text to the right of BADCHARcid:.BADCHAR
   Using a string length of 30 should get more than the image name. --->
  <cfset rightpart[i]=Mid(htmlmessage, findstart, 30)>
  <!--- use the ListFirst function to remove all the text starting
   at the quotation mark. --->
  <cfset imagename[i]=ListFirst(rightpart[i], '"')>

  <!--- Loop over the attachments query and find the CID. --->
  <cfloop query="attachments">
   <!--- Replace the image name with the contents of the attachment --->
   <cfif attachments.CID EQ imagename[i]>
    <cfset htmlmessage = Replace(htmlmessage,"cid:#imagename[i]#",
    "attachments/#attachments.ATTACHMENTFILENAME#")>
   </cfif>
  </cfloop>
 </cfloop>

 <h3>The full mail message</h3>
 <cfoutput>#htmlmessage#</cfoutput>

 <cfexchangeconnection
```

© 2014 Adobe Systems Incorporated. All rights reserved.

```
    action="close"
    connection = "testconn">
```

© 2014 Adobe Systems Incorporated. All rights reserved.

Modifying Exchange items

You can modify any elements of calendar, contact, and task items that you can set in ColdFusion. For mail message, you can change the `Importance`, `Sensitivity`, and `IsRead` flags, and you can move the mail messages between folders.

> **Note**
>
> If an item has attachments and you specify attachments when you modify the item, the new attachments are added to the previous attachments; they do not replace them. Use the `deleteAttachments` action to remove any obsolete or changed attachments.

Modifying calendar, contact, and task items

You can modify calendar, contact, and task items by using the `cfexchangecalendar`, `cfexchangecontact`, or `cfexchangetask` tag with an `action` attribute value of `modify`. You specify a `contact`, `event`, or `task` attribute with a structure that contains the item properties that you want to change, and their new values. You do not have to specify the values for properties that you are not changing. To change the end time of a calendar task, for example, you specify only an `EndTime` field in the `event` attribute structure.

The following example lets you create, and then modify a calendar event. When you first submit the form, ColdFusion creates the calendar event and redisplays the form with the data you entered. Accept the event before you modify the form and resubmit it. When you submit the form a second time, ColdFusion sends the modification information. For information about accepting events, see Working with meetings and appointments.

The following example resends all the event data (to limit the example length), but you could change the example so that it only sends modified data. This example also omits recurrence information to keep the code relatively simple:

```
<!--- Initialize the form.eventID to 0, to indicate a new event. --->
<!--- The EventID field is a hidden field managed by this application. --->
<cfparam name="form.eventID" default="0">

<!--- If the form was submitted, populate an event structure from it. --->
<cfif isDefined("Form.Submit")>
 <cfscript>
  sEvent=StructNew();
  sEvent.AllDayEvent="false";
  sEvent.Subject=Form.subject;
  if (IsDefined("Form.allDay")) {
   sEvent.AllDayEvent="true";
   sEvent.StartTime=createDateTime(Year(Form.date), Month(Form.date),
    Day(Form.date), 8, 0, 0);
  }
  else {
   sEvent.StartTime=createDateTime(Year(Form.date), Month(Form.date),
    Day(Form.date), Hour(Form.startTime), Minute(Form.startTime), 0);
   sEvent.EndTime=createDateTime(Year(Form.date), Month(Form.date),
    Day(Form.date), Hour(Form.endTime), Minute(Form.endTime), 0);
  }
  sEvent.Location=Form.location;
  sEvent.RequiredAttendees=Form.requiredAttendees;
  sEvent.OptionalAttendees=Form.optionalAttendees;
  //sEvent.Resources=Form.resources;
  if (Form.reminder NEQ "") {
   sEvent.Reminder=Form.reminder;
  }
  else {
```

© 2014 Adobe Systems Incorporated. All rights reserved.

```
    sEvent.Reminder=0;
   }
   sEvent.Importance=Form.importance;
   sEvent.Sensitivity=Form.sensitivity;
   sEvent.message=Form.Message;
  </cfscript>

  <!--- If the form is being submitted for the first time,
    create an event. --->
  <cfif form.eventID EQ 0>
  <!--- Create the event in Exchange --->
   <cfexchangecalendar action="create"
    username ="#user1#"
    password="#password1#"
    server="#exchangeServerIP#"
    event="#sEvent#"
    result="theUID">
   <!--- Display the new event UID and set form.eventID to it. --->
   <cfif isDefined("theUID")>
    <cfoutput>Event Added. UID is #theUID#</cfoutput>
    <cfset Form.eventID = theUID >
   </cfif>

  <cfelse>
  <!--- The form is being resubmitted with new data; update the event. --->
   <cfexchangecalendar action="modify"
    username ="#user1#"
    password="#password1#"
    server="#exchangeServerIP#"
    event="#sEvent#"
    uid="#Form.eventID#">
   <cfoutput>Event ID #Form.eventID# Updated.</cfoutput>

  </cfif>
 </cfif>

 <!--- A self-submitting form for the event information --->
 <cfform format="xml" preservedata="true" style="width:500" height="600">
  <cfinput type="text" label="Subject" name="subject" style="width:435">
  <br />
  <cfinput type="checkbox" label="All Day Event" name="allDay">
  <cfinput type="datefield" label="Date" name="date" validate="date"
   style="width:100">
  <cfinput type="text" label="Start Time" name="startTime" validate="time"
   style="width:100">
  <cfinput type="text" label="End Time" name="endTime" validate="time"
   style="width:100"><br />
  <cfinput type="text" label="Location" name="location"
   style="width:435"><br />
  <cfinput type="text" label="Required Attendees" name="requiredAttendees"
   style="width:435"><br />
  <cfinput type="text" label="Optional Attendees" name="optionalAttendees"
   style="width:435"><br />
  <cfinput type="text" label="Resources" name="resources"
   style="width:435"><br />
  <cfinput type="text" label="Reminder (minutes)" validate="integer"
   name="reminder" style="width:200">
  <cfselect name="importance" label="Importance" style="width:100">
   <option value="normal">Normal</option>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

```
 <option value="high">High</option>
 <option value="low">Low</option>
</cfselect>
<cfselect name="sensitivity" label="Sensitivity" style="width:100">
 <option value="normal">Normal</option>
 <option value="company-confidential">Confidential</option>
 <option value="personal">Personal</option>
 <option value="private">Private</option>
</cfselect>
<cfinput type="textarea" label="Message" name="message" style="width:435;
 height:100">
<cfinput type="hidden" name="eventID" value="#Form.EventID#">
```

© 2014 Adobe Systems Incorporated. All rights reserved.

```
 <cfinput type="Submit" name="submit" value="Submit" >
</cfform>
```

Setting mail attributes

To set a mail message's `Importance`, `Sensitivity`, or `IsRead` flag, use the cfexchangemail tag with an `action` attribute value of `set`. Specify only the flags that you are changing in the `message` attribute.
The following example snippet implements a catch-up operation by changing the `IsRead` flag to `true` on all mail messages that are directly in the Inbox and are more than two weeks old. The example does not change the flags on any messages in folders in the Inbox; to do so, use a separate `cfexchangemail` tag for each folder. For information on accessing and using multiple folders, see *Getting and using folder names* in Getting Exchange items and attachments.

```
<!--- Create a structure with a true IsRead field --->
<cfset changeValues.IsRead="true">

<!--- Open the connection. --->
<cfexchangeConnection
 action="open"
 username ="#user1#"
 password="#password1#"
 server="#exchangeServerIP#"
 connection="conn1">

<!--- Get the mail in the Inbox that is at least two weeks old. --->
<cfexchangemail action="get" name="oldMail" connection="conn1">
 <cfexchangefilter name="timeSent" from="01/01/2000"
  to="#DateAdd("d","-14", Now())#">
</cfexchangemail>

<!--- Loop through the resulting oldMail query and set the IsRead indicator
   to true. --->
<cfloop query="oldMail">
 <cfexchangemail action="set"
  connection="conn1"
  message="#changeValues#"
  uid="#oldMail.uid#">
</cfloop>

<!--- Close the connection. --->
<cfexchangeConnection
 action="close"
 connection="conn1">
```

Moving mail between folders

To move a one or more mail messages from one folder to another, use the `cfexchangemail` tag `move` action, as shown in the following code snippet, which moves all messages with the subject "Rams and Ewes" from the Unread folder in the Inbox to the Sheep folder in the inbox.

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cfexchangemail action="move" connection="con1" folder="Inbox/Unread"
  destinationfolder="Inbox/Sheep">
 <cfexchangefilter name="subject" value="Rams and Ewes">
</cfexchangemail>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

Deleting Exchange items and attachments

To delete an exchange item, use the ColdFusion Exchange tag with the `action` attribute of `delete` and specify the item UID. Deleting the exchange item deletes all attachments
To delete only the attachments to an exchange item, use the ColdFusion Exchange tag with the `action` attribute of `deleteAttachments` and specify the item UID,
This example deletes all meeting requests in the Inbox for meetings that have passed, but does not delete any requests in folders in the Inbox. To delete requests in the Inbox, use a separate `cfexchangemail` tag for each folder. For information on accessing and using multiple folders, see *Getting and using folder names* in Getting Exchange items and attachments.

```
<cfexchangeconnection
 action="open"
 username ="#user#"
 password="#password#"
 server="#exchangeServerIP#"
 connection="conn1">

<!--- Get all meeting notifications from the Inbox. --->
<cfexchangemail action="get" name="requests" connection="conn1">
 <cfexchangefilter name="MessageType" value="Meeting">
</cfexchangemail>

<!--- Get the meeting request data and put it in an array. --->
<cfset i=1>
<cfset meetingData=ArrayNew(1)>
<cfloop query="requests">
 <cfexchangemail action="getmeetinginfo" connection="conn1"
  name="meeting" meetinguid="#MeetingUID#" mailUID="#UID#">
 <cfset meetingData[i]=meeting>
 <cfset i=i+1>
</cfloop>

<!--- Loop through the request data array and delete any outdated
meeting messages from the Inbox. --->
<cfloop index="i" from="1" to="#ArrayLen(meetingData)#" >
 <cfif meetingData[i].StartTime LTE now()>
  <cfexchangemail action="delete" connection="conn1"
   UID="#meetingData[i].UID#">
 </cfif>
</cfloop>

<cfexchangeconnection
 action="close"
 connection="conn1">
```

For another example that deletes all mail from a known spam address, see *Using persistent connections* in Managing connections to the Exchange server.

© 2014 Adobe Systems Incorporated. All rights reserved.

Working with meetings and appointments

The following techniques apply specifically to calendar events and the notices about meetings that you get in your mail Inbox:

- How to get detailed information about meeting requests, cancellation notices, and responses to invitations
- How to specify event recurrence

Working with meeting notices and requests

Your mailbox gets a meeting notice when someone takes any of the following actions:

- Sends you a meeting request
- Cancels a meeting in your calendar
- Responds to a meeting request that you sent and tells Exchange to notify you
  The information provided by the `cfexchangemail` tag with the `get` action does not provide detailed information about meeting. It only includes the following meeting-related information:

- The event UID
- The type of message type: a meeting request, response, or cancellation
- If the message is a response to a meeting request, an indication whether the meeting was accepted, declined, or tentatively accepted
  Also, a meeting request does not appear in your calendar (so you cannot get detailed information about it using the `cfexchangecalendar` tag) until you accept it.
  To get detailed information about a meeting message, use the `cfexchangemail` tag with the `getMeetingInfo` action. After getting the information, you can take the necessary action, such as using an `cfexchangecalendar` tag with the `response` action to accept or decline a meeting request.

Get meeting message details and respond to meeting requests

1. Get the mail messages that contain the meeting notifications by using a `cfexchangemail` tag with an `action` attribute value of `get` and a `cfexchangefilter`child tag with the following attributes:
   - A `name` attribute with a value `MessageType`
   - A `value` attribute with a value of `Meeting`, `Meeting_Request`, `Meeting_Response`, or `Meeting_Cancel`. A value of `Meeting` gets all meeting notifications.
     You can use additional `cfexchangefilter` tags to further limit the messages you get.When the `cfexchangemail` tag completes processing, the `MeetingUID` column of the structure specified by the `cfexchangemail` tag `name` attribute contains the UIDs of the meetings.
2. For each meeting, get the information about the meeting by using a `cfexchangemail`tag with the following attributes:
   - An `action` attribute value of `getMeetingInfo.`
   - A `meetingUID` attribute value with the value from the `MeetingUID` column of the structure specified by the `cfexchangemail` tag `name` attribute.
   - (Optional) A `mailuid` attribute with the UID of the message that contained the meeting notification. Use this attribute to identify a specific message if the Inbox contains multiple messages about a single meeting.
3. Use the information returned in step 2 in application-specific logic to determine the required messages and actions. For example, you could display all meeting requests in a form that lets a user submit a response to each message.
4. To respond to a meeting request, use the `cfexchangecalendar` tag with an `action` value of `respond`and set the following the attributes:
   - Set the `uid` attribute to the Meeting UID you received in step 2. Do *not* use the Message UID.
   - Specify a `responseType` value of `accept`, `decline`, or `tentative`.
   - (Optional) Specify a `notify` value of `true` (the default value) or `false` to control whether the event

© 2014 Adobe Systems Incorporated. All rights reserved.

owner receives a meeting response message.

- If the owner receives a notification, you can also specify a `message` attribute with a text message that is included in the response.

  The following example shows how you can use this process. It displays all meeting invitations in the Inbox and lets the user respond to each request and send a message with the response:

```
<cfexchangeconnection
 action="open"
 username ="#user2#"
 password="#password2#"
 server="#exchangeServerIP#"
 connection="conn1">

<cfif isDefined("Form.Submit")>

<!--- When the form has been submitted, send the responses. --->
 <cfloop index="k" from="1" to="#Form.responses#">
  <cfset resp = Form["response" & k] >
  <cfset msg = Form["respMessage" & k] >
  <cfset msguid = Form["UID" & k] >
  <cfexchangecalendar action="respond" connection="conn1"
   uid="#msguid#" responseType="#resp#" message="#msg#">
  <cfoutput><h4>Response #k# sent!</h4></cfoutput>
 </cfloop>

<cfelse>
 <!--- Get all messages with meeting Requests. --->
 <cfexchangemail action="get" name="requests" connection="conn1">
  <cfexchangefilter name="MessageType" value="Meeting_Request">
 </cfexchangemail>

 <!--- Get the meeting request data. --->
 <cfloop query="requests">
  <cfexchangemail action="getmeetinginfo" connection="conn1"
   name="meeting" meetinguid="#MeetingUID#">
  <cfset meetingData[requests.currentrow]=meeting>
 </cfloop>

 <!--- Display the invitation data in a form. --->
 <cfform name="bar">
  <cfloop index="j" from="1" to="#ArrayLen(meetingData)#">
   <cfoutput>
    <h3>Meeting Request #j#</h3>
    Subject: #meetingData[j].Subject# <br />
    Sensitivity: #meetingData[j].Sensitivity# <br />
    Organizer: #meetingData[j].Organizer# <br />
    All Day?: #meetingData[j].AllDayEvent# <br />
    Day: #DateFormat(meetingData[j].StartTime)#  
    Starts: #TimeFormat(meetingData[j].StartTime)#  
    Ends: #TimeFormat(meetingData[j].EndTime)# <br />
    Duration: #meetingData[j].Duration# <br />
    Location: #meetingData[j].Location# <br />
    Message: #meetingData[j].Message# <br />
   </cfoutput>

   <!--- Specify the response to this invitation. --->
   <h4>response:</h4>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

```
    <cfinput type="radio" checked name="response#j#" value="accept">
     Accept
    <cfinput type="radio" name="response#j#" value="decline">Decline
    <cfinput type="radio" name="response#j#" value="tentative">Tentative
    <br />
    <cftextarea name="respMessage#j#" label="Message (optional)"
     width="300" height="200" />
    <cfinput type="hidden" name="UID#j#"
     value="#meetingData[j].MeetingUID#">
    <hr />
   </cfloop>
   <cfinput type="hidden" name="responses"
    value="#ArrayLen(meetingData)#">
   <cfinput type="Submit" name="submit" value="Submit">
  </cfform>

 </cfif>

 <cfexchangeconnection
```

© 2014 Adobe Systems Incorporated. All rights reserved.

```
action="close"
connection="conn1">
```

For an example that gets information about all declined meeting messages in the Inbox and all its subfolders, see the example in *Getting and using folder names* in Getting Exchange items and attachments.

Specifying Calendar recurrence

To create an event that recurs multiple times, you specify the following fields in the `event` attribute structure:

- Set the `IsRecurring` field to `true`.
- Specify a `RecurrenceType` field value of `DAILY`, `WEELY`, `MONTHLY`, or `YEARLY`.
- (Optional) Specify one of the following mutually exclusive fields: `RecurrenceCount`, `RecurrenceEndDate`, or `RecurrenceNoEndDate`.

> **Note**
>
> If you omit all three of these fields, the event is created with no end date, and if you specify a count or end date, the `RecurrenceNoEndDate` value is automatically false; therefore, Specify a `RecurrenceNoEndDate` field only if you are changing an existing event with a recurrence count or end date to one with no end date.

- Specify the recurrence details in additional fields that depend on the recurrence type.
  To change an event recurrence, including to change whether the event recurs, you specify only the fields whose values change. To stop an event from recurring, set the `IsRecurring` field to `false`. To convert an event from nonrecurring to recurring, set the `IsRecurring` field to `true` and set all the necessary recurrence fields.
  The following sections describe how to specify each type of recurrence. For detailed descriptions of the fields that you use, see cfexchangecalendar in the *CFML Reference*.

> **Note**
>
> If you specify a recurrence rule that conflicts with the start date that you specify, the first occurrence of the event is on first day following the start date that conforms to the rule, not on the start date. For example, if you schedule an event for the second Tuesday of the month, and specify a start date of June 2, 2007, the first occurrence of the event is on June 12, 2007.

Specifying daily recurrence

To set a recurrence that is based on days, you do one of the following:

- Define a `RecurrenceFrequency` field to specify the frequency of the event, in days. To schedule a meeting for every third day, for example, specify `RecurrenceFrequency="3"`.
- Specify `RecurEveryWeekDay="true"` to specify a meeting that is held five days a week.
  You cannot use daily recurrence to schedule a single event that occurs a multiple number of times, but only on week days. To schedule such an event, specify a weekly recurrence with multiple recurrence days.
  The following CFScript code sample sets daily recurrence for every three days and sets the event to occur 20 times:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
IsRecurring="true";
RecurrenceType="DAILY";
RecurrenceCount="20";
RecurrenceFrequency="3";
```

Specifying weekly recurrence

You can create an event that always occurs on the same day or days of the week, and occurs every week or every several weeks by specifying `RecurrenceType="WEEKLY"`. You use the following fields to control the frequency:

- Define a `RecurrenceFrequency` field to specify the frequency of the event, in weeks. If you omit this field, the event occurs every week. To schedule a meeting for every fourth week, for example, specify `RecurrenceFrequency="4"`.
- Specify a `RecurrenceDays` field with a comma-delimited list of one of more of the following strings: `MON`, `TUE`, `WED`, `THUR`, `FRI`, `SAT`, `SUN`. If you omit this attribute, the event recurs on the day of the week determined by the `startTime` field value.
  The following CFScript code sample sets an event that occurs on Tuesday and Thursday of every other week until December 3, 2007.

```
IsRecurring="true";
RecurrenceType="WEEKLY";
RecurrenceEndDate="12/13/2007";
RecurrenceFrequency="2";
RecurrenceDays="TUE,THU;
```

Specifying monthly recurrence

You can create an event that always occurs on a monthly basis, or occurs every several months by specifying `RecurrenceType="MONTHLY"`. You can schedule two types of events:

- Events that occur on the same date of each scheduled month, for example, on the tenth day of every three months.
- Events that occur on the same week of the month and the same day of the week, for example, on the second thursday of every month, or on the last Friday of every six months.
  To specify a date-based monthly event, you only specify the recurrence type, and, if the recurrence is not every month, the frequency. ColdFusion schedules the event to occur on the day of the week determined by the `startTime` field value. To schedule a meeting that occurs on the start date every four months, specify the following recurrence fields:

```
IsRecurring="true";
RecurrenceType="MONTHLY";
RecurrenceFrequency="4";
```

To specify an event that occurs on the same day of the week, specify the following fields in addition to `RecurrenceType`:

© 2014 Adobe Systems Incorporated. All rights reserved.

Field	Description
RecurrenceFrequency	The frequency of the event, in months. If you omit this field, the event occurs every month.
RecurrenceWeek	The week of the month on which the event occurs. Valid values are `first`, `second`, `third`, `fourth`, and `last`.
RecurrenceDay	The day of the week on which the event occurs. Valid values are `SUN`, `MON`, `TUE`, `WED`, `THU`, `FRI`, and `SAT`.

The following CFScript code sample sets an event that occurs on the third Thursday of every three months:

```
IsRecurring="true";
RecurrenceType="Monthly";
RecurrenceFrequency="3";
RecurrenceWeek="third";
RecurrenceDay="THU";
```

Specifying yearly recurrence

You can create an event that always occurs on a yearly basis by specifying `RecurrenceType="YEARLY"`. You can schedule two types of events:

- Events that occur on the same date of each year, for example, on every August 10.
- Events that occur on a specific day week and month, for example, on the second Thursday of August.
  To specify a date-based yearly event, you only specify the recurrence type. ColdFusion schedules the event to occur each year on the date determined by the `startTime` field value. To schedule a meeting that occurs on the start date every year, specify the following recurrence fields:

```
IsRecurring="true";
RecurrenceType="YEARLY";
```

To specify an event that occurs on the same day of the week and month each year, specify the following fields in addition to `RecurrenceType`:

Field	Description
RecurrenceMonth	The month of the year which the event occurs. Valid values are `JAN`, `FEB`, `MAR`, `APR`, `MAY`, `JUN`, `JUL`, `AUG`, `SEP`, `OCT`, `NOV`, and `DEC`.
RecurrenceWeek	The week of the month during which the event occurs. Valid values are `first`, `second`, `third`, `fourth`, and `last`.

© 2014 Adobe Systems Incorporated. All rights reserved.

RecurrenceDay	The day of the week on which the event occurs. Valid values are `SUN`, `MON`, `TUE`, `WED`, `THU`, `FRI`, and `SAT`.

The following CFScript code sample sets an event that occurs on the third Thursday of August three months:

```
IsRecurring="true";
RecurrenceType="YEARLY";
RecurrenceMonth="AUG";
RecurrenceWeek="third";
RecurrenceDay="THU";
```

Example: Setting calendar recurrence

The following example lets you create events with all types of recurrence. To limit the code length, it does not prevent you from attempting to create events with invalid field combinations. When you submit the form, if an event is created, the form redisplays, preceded by a dump that shows the field values that were used to create the event, and the event UID. You cannot resubmit the form to modify the event, but you can change some values in the form and create an event.

```
<!--- Create a structure to hold the event information. --->
<cfparam name="form.eventID" default="0">

<!--- If the form was submitted, populate the event structure from it. --->
<cfif isDefined("Form.Submit")>
 <cfscript>
  sEvent.AllDayEvent="false";
  sEvent=StructNew();
  sEvent.Subject=Form.subject;
  if (IsDefined("Form.allDay")) {
   sEvent.AllDayEvent="true";
   sEvent.StartTime=createDateTime(Year(Form.date), Month(Form.date),
    Day(Form.date), 8, 0, 0);
  }
  else {
   sEvent.StartTime=createDateTime(Year(Form.date), Month(Form.date),
    Day(Form.date), Hour(Form.startTime), Minute(Form.startTime), 0);
   sEvent.EndTime=createDateTime(Year(Form.date), Month(Form.date),
    Day(Form.date), Hour(Form.endTime), Minute(Form.endTime), 0);
  }
  sEvent.Location=Form.location;
  sEvent.RequiredAttendees=Form.requiredAttendees;
  sEvent.OptionalAttendees=Form.optionalAttendees;
  //sEvent.Resources=Form.resources;
  if (Form.reminder NEQ "") {
   sEvent.Reminder=Form.reminder;
  }
  else {
   sEvent.Reminder=0;
  }
  sEvent.Importance=Form.importance;
  sEvent.Sensitivity=Form.sensitivity;
  //Recurrence Fields
  if (IsDefined("Form.isRecurring")) {
```

© 2014 Adobe Systems Incorporated. All rights reserved.

```
    sEvent.IsRecurring="true";}
   if (IsDefined("Form.recurrenceNoEndDate")) {
    sEvent.RecurrenceNoEndDate="true";}
   if (Form.recurrenceCount NEQ "") {
   sEvent.RecurrenceCount=Form.recurrenceCount;}
   if (Form.recurrenceEndDate NEQ "") {
   sEvent.RecurrenceEndDate=createDateTime(Year(Form.recurrenceEndDate),
    Month(Form.recurrenceEndDate), Day(Form.recurrenceEndDate), 0, 0,
    0);}
   sEvent.RecurrenceType=Form.recurrenceType;
   if (Form.recurrenceFrequency NEQ "") {
   sEvent.recurrenceFrequency=Form.recurrenceFrequency;}
   if (IsDefined("Form.recurEveryWeekDay")) {
    sEvent.RecurEveryWeekDay="true";}
   if (Form.recurrenceDays NEQ "") {
   sEvent.RecurrenceDays=Form.recurrenceDays;}
   if (Form.recurrenceDay NEQ "") {
   sEvent.RecurrenceDay=Form.recurrenceDay;}
   if (Form.recurrenceWeek NEQ "") {
   sEvent.RecurrenceWeek=Form.recurrenceWeek;}
   if (Form.recurrenceMonth NEQ "") {
   sEvent.RecurrenceMonth=Form.recurrenceMonth;}

   sEvent.message=Form.Message;
  </cfscript>

 <cfdump var="#sEvent#">

 <!--- Create the event in Exchange. --->
  <cfexchangecalendar action="create"
   username ="#user1#"
   password="#password1#"
   server="#exchangeServerIP#"
   event="#sEvent#"
   result="theUID">
  <!--- Output the UID of the new event --->
  <cfif isDefined("theUID")>
   <cfoutput>Event Added. UID is#theUID#</cfoutput>
   <cfset Form.eventID = theUID >
  </cfif>
 </cfif>

 <cfform format="xml" preservedata="true" style="width:500" height="700">
  <cfinput type="text" label="Subject" name="subject" style="width:435">
    <br />
  <cfinput type="checkbox" label="All Day Event" name="allDay">
  <cfinput type="datefield" label="Date" name="date" validate="date"
   style="width:100">
  <cfinput type="text" label="Start Time" name="startTime" validate="time"
   style="width:100">
  <cfinput type="text" label="End Time" name="endTime" validate="time"
   style="width:100"><br />
  <cfinput type="text" label="Location" name="location"
   style="width:435"><br />
  <cfinput type="text" label="Required Attendees" name="requiredAttendees"
   style="width:435"><br />
  <cfinput type="text" label="Optional Attendees" name="optionalAttendees"
   style="width:435"><br />
  <cfinput type="text" label="Resources" name="resources"
```

© 2014 Adobe Systems Incorporated. All rights reserved.

```
 style="width:435"><br />
<cfinput type="text" label="Reminder (minutes)" validate="integer"
 name="reminder" style="width:200">
<cfselect name="importance" label="Importance" style="width:100">
 <option value="normal">Normal</option>
 <option value="high">High</option>
 <option value="low">Low</option>
</cfselect>
<cfselect name="sensitivity" label="Sensitivity" style="width:100">
 <option value="normal">Normal</option>
 <option value="company-confidential">Confidential</option>
 <option value="personal">Personal</option>
 <option value="private">Private</option>
</cfselect>
<hr />
<!--- Recurrence Information --->
<cfinput type="checkbox" label="IsRecurring" name="isRecurring">
<cfinput type="checkbox" label="RecurrenceNoEndDate" name="noEndDate">
<cfinput type="text" label="RecurrenceCount" validate="integer"
 required="false" name="recurrenceCount">
<cfinput type="text" label="RecurrenceEndDate" validate="date"
 required="false" name="recurrenceEndDate">
<cfselect name="RecurrenceType" label="Recurrence Type"
  style="width:100">
 <option value="DAILY">Daily</option>
 <option value="WEEKLY">Weekly</option>
 <option value="MONTHLY">Monthly</option>
 <option value="YEARLY">Yearly</option>
</cfselect>
<cfinput type="text" label="RecurrenceFrequency" validate="integer"
 name="recurrenceFrequency">
<cfinput type="checkbox" label="RecurEveryWeekDay"
 name="recurEveryWeekDay">
<cfinput type="text" label="RecurrenceDays" name="recurrenceDays">
<cfinput type="text" label="RecurrenceDay" name="recurrenceDay">
<cfselect name="RecurrenceWeek" label="RecurrenceWeek" style="width:100">
 <option value=""></option>
 <option value="first">First</option>
 <option value="second">Second</option>
 <option value="third">Third</option>
 <option value="fourth">Fourth</option>
 <option value="last">Last</option>
<cfinput type="text" label="RecurrenceMonth" name="recurrenceMonth">
</cfselect>
<hr />
<cfinput type="textarea" label="Message" name="message" style="width:300;
  height:100">
```

© 2014 Adobe Systems Incorporated. All rights reserved.

```
 <cfinput type="Submit" name="submit" value="Submit" >
</cfform>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

Connect to Microsoft Exchange Server 2010

Support for integration with Microsoft Exchange Server 2010

Adobe ColdFusion can interact with Microsoft Exchange Server 2010 SP1. The enhancements offer support for Microsoft Exchange Web Services (EWS) which brings in efficacy with the following operations:

- Folder operations such as create, modify, or delete.
- Get rooms and roomlist in the exchange organization.
- Information on user availability, that helps effective scheduling.
- Conversation operations such as find conversation details, copy, move, and the status if the conversation is read.

Note

In ColdFusion 9, the protocol support was limited to WEBDAV.

Note

If you are installing the J2EE configuration of ColdFusion, all JARs in the ews folder (residing in cfusion/lib) have to be present in the system classpath while deploying.

Note

This release does not support form-based authentication to Microsoft Exchange Server 2010.

Setting your Microsoft Exchange Server version

At application level

You can specify the Microsoft Exchange Server version at the application level by providing a value for the variable `exchangeServerVersion` as follows:`<cfset this.exchangeserverversion="version">`This corresponds to the attribute with the same name in the tag `cfapplication`.

In cfapplication

A new attribute `exchangeServerVersion` has been added to `cfapplication`.

Syntax

```
<cfapplication name="app_name" exchangeServerVersion="2010">
...
</cfapplication>
```

Attribute

Attribute	Required/Optional	Default	Description

© 2014 Adobe Systems Incorporated. All rights reserved.

exchangeServerVersion	Optional	2007	Specifies the Microsoft Exchange Server version. The values are: • 2003 • 2007 • 2010 If you do not specify the details, 2007 is taken by default.

Usage

Use this attribute to specify the version of Microsoft Exchange Server that ColdFusion interacts with. You can also set the version in the Application.cfc

At tag level in any of the cfexchange tags

New attribute `serverVersion` has been added to the following tags:

- `cfexchangeconnection`
- `cfexchangemail`
- `cfexchangecalendar`
- `cfexchangetask`
- `cfexchangecontact`

Attribute	Required/Optional	Default	Description
serverVersion	Optional	2007	Specifies the Microsoft Exchange Server version. The values are: • 2003 • 2007 • 2010 If you do not specify the details, 2007 is taken by default.The value you specify overrides the value that you specify at the application level.

New attribute folderID added to cfexchangemail

A new attribute `folderID` supports the following actions: `get`, `move`, and `set.` This is the case-sensitive Exchange UID value that uniquely identifies the folder.If not specified, `folder` is used. If either `folder` or `folderID` are not specified, the inbox is used as the default folder to perform the operation.

New actions added to the tag cfexchangecalendar

Added the following three actions:

© 2014 Adobe Systems Incorporated. All rights reserved.

- **`getUserAvailability`**: To effectively schedule meetings and find the availability of users.
- **`getRoomsList`**: To find the list of rooms in an organization.
- **`getRooms`**: To find the list of rooms in a room list.

Modifications to the tag cfexchangecalendar

For all the cfexchangecalendar actions, the value of the attribute `uid` is as follows:

- If `exchangeServerVersion` is set to 2003 or 2007: The `uid` indicates the ID of the appointment in the mailbox of the organizer.
- If `exchangeServerVersion` is set to 2010: The `uid` indicates the ID of the received appointment in the mailbox of the attendee.
  In the case of interaction with Microsoft Exchange server 2003 or 2007, whenever an appointment is created, the UID of the organizer can be used by the attendee for any operation such as responding, deleting, or getting attachments. In the case of Microsoft Exchange server 2010, the behavior is different. If attendees have to perform appointment-related actions, they have to first search for the appointment in their mailbox and then use the UID of that appointment.

#back to top

© 2014 Adobe Systems Incorporated. All rights reserved.

Interacting with Remote Servers

Adobe ColdFusion wraps the complexity of Hypertext Transfer Protocol (HTTP) and File Transfer Protocol (FTP) communications in a simplified tag syntax that lets you extend your site offerings across the web.

© 2014 Adobe Systems Incorporated. All rights reserved.

About interacting with remote servers

Transfer protocols are mechanisms for moving files and information from a source to one or more destinations. Two of the more common protocols are the Hypertext Transfer Protocol (HTTP) and the File Transfer Protocol (FTP). ColdFusion has the `cfhttp` and `cfftp` tags that let you use these protocols to interact with remote servers.

The cfhttp tag lets you receive a web page or web-based file, just as a web browser uses HTTP to transport web pages. When you type a URL into a web browser, you make an HTTP request to a web server. With the `cfhttp` tag , you can display a web page, send variables to a ColdFusion or CGI application, retrieve specialized content from a web page, and create a ColdFusion query from a text file. You can use the Get or Post methods to interact with remote servers.

The `cfftp` tag takes advantage of the main purpose of FTP transporting files. Unlike HTTP, FTP was not designed to interact with other servers for processing and interacting with data. After you establish an FTP connection with the `cfhttp` tag, you can use it to upload, download, and manage files and directories.

© 2014 Adobe Systems Incorporated. All rights reserved.

Using cfhttp to interact with the web

The cfhttp tag, which lets you retrieve information from a remote server, is one of the more powerful tags in the CFML tag set. You can use one of two methods Get or Post to interact with a remote server using the `cfhttp` tag:

- Using the Get method, you can only send information to the remote server in the URL. This method is often used for a one-way transaction in which `cfhttp` retrieves an object.
- Using the Post method, you can pass variables to a ColdFusion page or CGI program, which processes them and returns data to the calling page. The calling page then appears or further processes the data that was received. For example, when you use `cfhttp` to Post to another ColdFusion page, that page does not appear. It processes the request and returns the results to the original ColdFusion page, which then uses the information as appropriate.

Using the cfhttp Get method

You use Get to retrieve files, including text and binary files, from a specified server. The retrieved information is stored in a special variable, `cfhttp.fileContent`. The following examples show several common Get operations.

Retrieve a file and store it in a variable

1. Create a ColdFusion page with the following content:

```
<html>
<head>
 <title>Use Get Method</title>
</head>
<body>
<cfhttp
 method="Get"
 url="http://www.adobe.com"
 resolveurl="Yes">
<cfoutput>
 #cfhttp.FileContent# <br>
</cfoutput>

</body>
</html>
```

2. (Optional) Replace the value of the `url` attribute with another URL.
3. Save the file as get_webpage.cfm in the myapps directory under your *web_root* and view it in the web browser.The browser loads the web page specified in the `url` attribute.

Reviewing the code

The following table describes the code and its function:

Code	Description

© 2014 Adobe Systems Incorporated. All rights reserved.

`<cfhttp method="Get" url="http://www.adobe.com" resolveurl="Yes">`	Get the page specified in the URL and make the links absolute instead of relative so that they appear properly.
`<cfoutput> #cfhttp.FileContent#   </cfoutput>`	Display the page, which is stored in the variable `cfhttp.fileContent`, in the browser.

Get a web page and save it in a file

1. Create a ColdFusion page with the following content:

```
<html>
<head>
 <title>Use Get Method</title>
</head>
<body>

<cfhttp
 method = "Get"
 url="http://www.adobe.com/software"
 path="c:\temp"
 file="adobe_software.htm">
</body>
</html>
```

2. (Optional) Replace the value of the `url` attribute with another URL and change the filename.
3. (Optional) Change the path from C:\temp to a path on your hard drive.
4. Save the page as save_webpage.cfm in the myapps directory under your *web_root* directory.
5. Go to the specified path and view the file that you specified in a text editor (using the values specified in step 1, the path is C:\temp\macr_software.htm).The saved file does not appear properly in your browser because the Get operation saves only the specified web page HTML. It does not save the frame, image, or other files that the page could include.

Reviewing the code

The following table describes the code and its function:

Code	Description

© 2014 Adobe Systems Incorporated. All rights reserved.

`<cfhttp method = "Get" url="http://www.adobe.com/software" path="c:\temp" file="macr_software.htm">`	Get the page specified in the URL and save it in the file specified by the `path` and `file` attributes.When you use the `path` and `file` attributes, ColdFusion ignores any `resolveurl` attribute. As a result, frames and other included files cannot appear when you view the saved page.

Get a binary file and save it

1. Create a ColdFusion page with the following content:

```
<cfhttp
 method="Get"
 url="http://www.adobe.com/adobe/accessibility/images/spotlight.jpg"
 path="c:\temp"
 file="My_SavedBinary.jpg">
<cfoutput>
 #cfhttp.MimeType#
</cfoutput>
```

2. (Optional) Replace the value of the `url` attribute with the URL of a binary file that you want to download.
3. (Optional) Change the path from C:\temp to a path on your hard drive.
4. Save the file as save_binary.cfm in the myapps directory under your *web_root* and open it in the web browser to view the MIME type.
5. (Optional) Verify that the binary file now exists at the location you specified in the `path` attribute.

Reviewing the code

The following table describes the code and its function:

Code	Description
`<cfhttp method="Get" url="http://www.adobe.com/adobe/accessibility/images/spotlight.jpg" path="c:\temp" file="My_SavedBinary.jpg">`	Get a binary file and save it in the `path` and `file` specified.
`<cfoutput> #cfhttp.MimeType# </cfoutput>`	Display the MIME type of the file.

© 2014 Adobe Systems Incorporated. All rights reserved.

Creating a query object from a text file

You can create a query object from a delimited text file by using the cfhttp tag and specifying `method="Get"` and the `name` attribute. This technique is a powerful method for processing and handling text files. After you create the query object, you can easily reference columns in the query and perform other ColdFusion operations on the data. ColdFusion processes text files in the following manner:

- You can specify a field delimiter with the `delimiter` attribute. The default is a comma.
- If data in a field could include the delimiter character, surround the entire field with the text qualifier character, which you can specify with the `textqualifier` attribute. The default text qualifier is the double-quotation mark (").
- The `textqualifier=""` specifies that no text qualifier `exists.` If you use `textqualifier="""""` (four " marks in a row), it explicitly specifies the double-quotation mark as the text qualifier.
- If a text qualifier exists, surround all field values with the text qualifier character.
- To include the text qualifier character in a field, use a double character. For example, if the text qualifier is ", use "" to include a quotation mark in the field.
- The first row of text is always interpreted as column headings, so that row is skipped. You can override the file column heading names by specifying a different set of names in the `columns` attribute. Specify a name for each column. You then use these new names in your CFML code. However, ColdFusion never treats the first row of the file as data.
- When duplicate column heading names are encountered, ColdFusion adds an underscore character to the duplicate column name to make it unique. For example, if two CustomerID columns are found, the second is renamed "CustomerID_".

Create a query from a text file

1. Create a text file with the following content:

```
OrderID,OrderNum,OrderDate,ShipDate,ShipName,ShipAddress
001,001,01/01/01,01/11/01,Mr. Shipper,123 Main Street
002,002,01/01/01,01/28/01,Shipper Skipper,128 Maine Street
```

2. Save the file as text.txt in the myapps directory under your *web_root*.
3. Create a ColdFusion page with the following content:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cfhttp method="Get"
 url="http://127.0.0.1/myapps/text.txt"
 name="juneorders"
 textqualifier="">

<cfoutput query="juneorders">
 OrderID: #OrderID#<br>
 Order Number: #OrderNum#<br>
 Order Date: #OrderDate#<br>
</cfoutput>
<!--- Now substitute different column names --->
<!--- by using the columns attribute --->
<hr>
Now using replacement column names<br>

<cfhttp method="Get"
 url="http://127.0.0.1/myapps/text.txt"
 name="juneorders"
 columns="ID,Number,ODate,SDate,Name,Address"
 textqualifier="">

<cfoutput query="juneorders">
 Order ID: #ID#<br>
 Order Number: #Number#<br>
 Order Date: #SDate#<br>
</cfoutput>
```

4. Save the file as query_textfile.cfm in the myapps directory under your *web_root* and view it in the web browser.

© 2014 Adobe Systems Incorporated. All rights reserved.

Using the cfhttp Post method

Use the Post method to send cookie, form field, CGI, URL, and file variables to a specified ColdFusion page or CGI program for processing. For Post operations, use the cfhttpparam tag for each variable you want to post. The Post method passes data to a specified ColdFusion page or an executable that interprets the variables being sent and returns data.
For example, when you build an HTML form using the Post method, you specify the name of the page to which form data is passed. You use the Post method in cfhttp in a similar way. However, with the `cfhttp` tag, the page that receives the Post does not, itself, display anything.

Pass variables to a ColdFusion page

1. Create a ColdFusion page with the following content:

```
<html>
<head>
 <title>HTTP Post Test</title>
</head>
<body>
<h1>HTTP Post Test</h1>
<cfhttp method="Post"
 url="http://127.0.0.1:8500/myapps/post_test_server.cfm">
 <cfhttpparam type="Cookie"
  value="cookiemonster"
  name="mycookie6">
 <cfhttpparam type="CGI"
  value="cgivar "
  name="mycgi">
 <cfhttpparam type="URL"
  value="theurl"
  name="myurl">
 <cfhttpparam type="Formfield"
  value="twriter@adobe.com"
  name="emailaddress">
 <cfhttpparam type="File"
  name="myfile"
  file="c:\pix\trees.gif">
</cfhttp>
<cfoutput>
File Content:<br>
 #cfhttp.filecontent#<br>
Mime Type:#cfhttp.MimeType#<br>
</cfoutput>
</body>
</html>
```

2. Replace the path to the GIF file to a path on your server (just before the closing `cfhttp` tag).
3. Save the file as post_test.cfm in the myapps directory under your *web_root*.

Note

Write a page to view the variables, as described in next procedure.

Reviewing the code

© 2014 Adobe Systems Incorporated. All rights reserved.

The following table describes the code and its function:

Code	Description
`<cfhttp method="Post" url="http://127.0.0.1:8500/myapps/post_test_server.cfm">`	Post an HTTP request to the specified page.
`<cfhttpparam type="Cookie" value="cookiemonster" name="mycookie6">`	Send a cookie in the request.
`<cfhttpparam type="CGI" value="cgivar " name="mycgi">`	Send a CGI variable in the request.
`<cfhttpparam type="URL" value="theurl" name="myurl">`	Send a URL in the request.
`<cfhttpparam type="Formfield" value="twriter@adobe.com" name="emailaddress">`	Send a Form field in the request.
`<cfhttpparam type="File" name="myfile" file="c"\pix\trees.gif">`	Send a file in the request. The `</>` tag ends the http request.
`<cfoutput>` `File Content: ` `#cfhttp.filecontent# `	Display the contents of the file that the page that is posted to creates by processing the request. In this example, the contents is the output from the `cfoutput` tag in server.cfm.

© 2014 Adobe Systems Incorporated. All rights reserved.

`Mime Type: #cfhttp.MimeType# ` `</cfoutput>`	Display the MIME type of the created file.

View the variables

1. Create a ColdFusion page with the following content:

```
<html>
<head><title>HTTP Post Test</title> </head>
<body>
<h1>HTTP Post Test</h1>
<cffile destination="C:\temp\"
 nameconflict="Overwrite"
 filefield="Form.myfile"
 action="Upload"
 attributes="Normal">
<cfoutput>
 The URL variable is: #URL.myurl# <br>
 The Cookie variable is: #Cookie.mycookie6# <br>
 The CGI variable is: #CGI.mycgi#. <br>
 The Formfield variable is: #Form.emailaddress#. <br>
 The file was uploaded to #File.ServerDirectory#\#File.ServerFile#.
</cfoutput>
</body>
</html>
```

2. Replace C:\temp\ with an appropriate directory path on your hard drive.
3. Save the file as post_test_server.cfm in the myapps directory under your *web_root*.
4. View post_test.cfm in your browser and look for the file in C:\temp\ (or your replacement path).

Reviewing the code

The following table describes the code and its function:

Code	Description

`<cffile destination="C:\temp\"`
`nameconflict="Overwrite"`
`filefield="Form.myfile"`
`action="Upload"`
`attributes="Normal">`
	Write the transferred document to a file on the server. You send the file using the `type="File"` attribute, but the receiving page gets it as a Form variable, not a File variable. This cffile tag creates File variables, as follows.

`<cfoutput>`	Output information. This page does not display the results. They are passed back to the posting page in its `cfhttp.filecontent` variable.

© 2014 Adobe Systems Incorporated. All rights reserved.

`The URL variable is: #URL.myurl#  `	Output the value of the URL variable sent in the HTTP request.
`The Cookie variable is: #Cookie.mycookie#  `	Output the value of the Cookie variable sent in the HTTP request.
`The CGI variable is: #CGI.mycgi#  `	Output the value of the CGI variable sent in the HTTP request.
`The Form variable is: #Form.emailaddress#.  `	Output the Form variable sent in the HTTP request. You send the variable using the `type="formField"` attribute but the receiving page gets it as a Form variable.
`The file was uploaded to #File.ServerDirectory#\#File.ServerFile#. </cfoutput>`	Output the results of the `cffile` tag on this page. This time, the variables really are File variables.

Return results of a CGI program

The following code runs a CGI program search.exe on a website and displays the results, including both the MIME type and length of the response. The search.exe program must expect a "search" parameter.

```
<cfhttp method="Post"
  url="http://www.my_favorite_site.com/search.exe"
  resolveurl="Yes">
 <cfhttpparam type="Formfield"
  name="search"
  value="ColdFusion">
</cfhttp>
<cfoutput>
 Response Mime Type: #cfhttp.MimeType#<br>
 Response Length: #len(cfhttp.filecontent)# <br>
 Response Content: <br>
  #htmlcodeformat(cfhttp.filecontent)#<br>
</cfoutput>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

Performing file operations with cfftp

The `cfftp` tag lets you perform tasks on remote servers using File Transfer Protocol (FTP). You can use `cfftp` to cache connections for batch file transfers when uploading or downloading files.

> **Note**
>
> To use `cfftp`, select the Enable ColdFusion Security option on the Sandbox Security page in the Security area in the ColdFusion Administrator. (In the Standard Edition, select Security > Basic Security.)

For server/browser operations, use the cffile, cfcontent, and cfdirectory tags.
Using `cfftp` involves two major types of operations: connecting, and transferring files. The FTP protocol also provides commands for listing directories and performing other operations. For a complete list of attributes that support FTP operations and additional details on using the cfftp tag, see the *CFML Reference.*

Open an FTP connection and retrieve a file listing

1. Create a ColdFusion page with the following content:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<html>
<head>
 <title>FTP Test</title>
</head>

<body>
<h1>FTP Test</h1>
<!--- Open ftp connection --->
<cfftp connection="Myftp"
 server="MyServer"
 username="MyUserName"
 password="MyPassword"
 action="Open"
 stoponerror="Yes">

<!--- Get the current directory name. --->
<cfftp connection=Myftp
 action="GetCurrentDir"
 stoponerror="Yes">

<!--- output directory name --->
<cfoutput>
 The current directory is:#cfftp.returnvalue#<p>
</cfoutput>

<!--- Get a listing of the directory. --->
<cfftp connection=Myftp
 action="listdir"
 directory="#cfftp.returnvalue#"
 name="dirlist"
 stoponerror="Yes">
<!--- Close the connection.--->
<cfftp action="close" connection="Myftp">
<p>Did the connection close successfully?
 <cfoutput>#cfftp.succeeded#</cfoutput></p>

<!--- output dirlist results --->
<hr>
<p>FTP Directory Listing:</p>

<cftable query="dirlist" colheaders="yes" htmltable>
 <cfcol header="<B>Name</b>" TEXT="#name#">
 <cfcol header="<B>Path</b>" TEXT="#path#">
 <cfcol header="<B>URL</b>" TEXT="#url#">
 <cfcol header="<B>Length</b>" TEXT="#length#">
 <cfcol header="<B>LastModified</b>"
 TEXT="#DateFormat(lastmodified)#">
 <cfcol header="<B>IsDirectory</b>"
  TEXT="#isdirectory#">
</cftable>
```

2. Change `MyServer` to the name of a server for which you have FTP permission.
3. Change `MyUserName` and `MyPassword` to a valid user name and password.To establish an anonymous connection, enter "anonymous" as the user name and an e-mail address (by convention) for the password.

© 2014 Adobe Systems Incorporated. All rights reserved.

4. Save the file as ftp_connect.cfm in the myapps directory under your *web_root* and view it in the web browser.

Reviewing the code

The following table describes the code and its function:

Code	Description
`<cfftp connection="Myftp" server="MyServer" username="MyUserName" password="MyPassword" action="Open" stoponerror="Yes">`	Open an FTP connection to the MyServer server and log on as MyUserName. If an error occurs, stop processing and display an error. You can use this connection in other `cfftp` tags by specifying the Myftp connection.
`<cfftp connection=Myftp action="GetCurrentDir" stoponerror="Yes">` `<cfoutput>` `The current directory is:` `#cfftp.returnvalue#<p>` `</cfoutput>`	Use the Myftp connection to get the name of the current directory; stop processing if an error occurs. Display the current directory.
`<cfftp connection=Myftp action="ListDir" directory="#cfftp.returnvalue#" name="dirlist" stoponerror="Yes">`	Use the Myftp connection to get a directory listing. Use the value returned by the last `cfftp` call (the current directory of the connection) to specify the directory to list. Save the results in a variable named dirlist (a query object). Stop processing if an error occurs.
`<cfftp action="close" connection="Myftp">` `<p>Did the connection close successfully?` `<cfoutput>#cfftp.succeeded#</cfoutput></p>`	Close the connection, and do not stop processing if the operation fails (because you can still use the results). Instead, display the value of the `cfftp.succeeded` variable, which is `Yes` if the connection is closed, and `No` if the operation failed.

© 2014 Adobe Systems Incorporated. All rights reserved.

`<cftable query="dirlist" colheaders="yes" htmltable> <cfcol header="<B>Name</b>" TEXT="#name#"> <cfcol header="<B>Path</b>" TEXT="#path#"> <cfcol header="<B>URL</b>" TEXT="#url#"> <cfcol header="<B>Length</b>" TEXT="#length#"> <cfcol header="<B>LastModified</b>" TEXT="#DateFormat(lastmodified)#">`
 	
`<cfcol header="<B>IsDirectory</b>" TEXT="#isdirectory#"> </cftable>`
	Display a table with the results of the ListDir FTP command.

After you establish a connection with `cfftp`, you can reuse the connection to perform additional FTP operations until either you or the server closes the connection. When you access an already-active FTP connection, you need not respecify the user name, password, or server. In this case, make sure that when you use frames, only one frame uses the connection object.

Note

For a single simple FTP operation, such as GetFile or PutFile, you need not establish a connection. Specify all the necessary login information, including the server and any login and password, in the single `cfftp` request.

Caching connections across multiple pages

The FTP connection established by the cfftp tag is maintained only in the current page unless you explicitly assign the connection to a variable with Application or Session scope.
Assigning a `cfftp` connection to an application variable could cause problems, since multiple users could access the same connection object at the same time. Creating a session variable for a `cfftp` connection makes more sense, because the connection is available to only one client and does not last past the end of the session.

Example: caching a connection

```
<cflock scope="Session" timeout=10>
<cfftp action="Open"
 username="anonymous"
 password="me@home.com"
 server="ftp.eclipse.com"
 connection="Session.myconnection">
</cflock>
```

In this example, the connection cache remains available to other pages within the current session. For this approach to work, enable session variables in your application, and lock code that uses session variables. For more

© 2014 Adobe Systems Incorporated. All rights reserved.

information on locking, see Using Persistent Data and Locking.

> **Note**
>
> Changing connection characteristics, such the `retrycount` or `timeout` values, could require you to re-establish the connection.

Connection actions and attributes

The following table shows the available `cfftp` actions and the attributes they require when you use a named (that is, cached) connection. If you do not specify an existing connection name, specify the `username`, `password`, and `server` attributes.

Action	Attributes	Action	Attributes
Open	none	Rename	existingnew
Close	none	Remove	serveritem
ChangeDir	directory	GetCurrentDir	none
CreateDir	directory	GetCurrentURL	none
ListDir	namedirectory	ExistsDir	directory
RemoveDir	directory	ExistsFile	remotefile
GetFile	localfileremotefile	Exists	item
PutFile	localfileremotefile		

#back to top

© 2014 Adobe Systems Incorporated. All rights reserved.

Managing Files on the Server

The cffile, cfdirectory, and cfcontent tags handle browser and server file management tasks, such as uploading files from a client to the web server, viewing directory information, and changing the content type that is sent to the web browser. To perform server-to-server operations, use the cfftp tag, described in Performing file operations with cfftp.

© 2014 Adobe Systems Incorporated. All rights reserved.

About file management

Adobe ColdFusion lets you access and manage the files and directories on your ColdFusion server. The cffile tag has several attributes for moving, copying, deleting, and renaming files. You use the cfdirectory tag to list, create, delete, and rename directories. The cfcontent tag lets you define the MIME (Multipurpose Internet Mail Extensions) content type that returns to the web browser.

© 2014 Adobe Systems Incorporated. All rights reserved.

Using cffile

You can use the cffile tag to work with files on the server in several ways:

- Upload files from a client to the web server using an HTML form
- Move, rename, copy, or delete files on the server
- Read, write, or append to text files on the server
  You use the `action` attribute to specify any of the following file actions: `upload`, `move`, `rename`, `copy`, `delete`, `read`, `readBinary`, `write`, and `append`. The required attributes depend on the `action` specified. For example, if `action="write"`, ColdFusion expects the attributes associated with writing a text file.

> **Note**
>
> Consider the security and logical structure of directories on the server before allowing users access to them. You can disable the `cffile` tag in the ColdFusion Administrator. Also, to access files that are not located on the local ColdFusion system, ColdFusion services must run using an account with permission to access the remote files and directories.

Uploading files

File uploading requires that you create two files:

- An HTML form to specify file upload information
- An action page containing the file upload code
  The following procedures describe how to create these files.

Create an HTML file to specify file upload information

1. Create a ColdFusion page with the following content:

```
<head><title>Specify File to Upload</title></head>
<body>
<h2>Specify File to Upload</h2>
<!--- the action attribute is the name of the action page --->
<form action="uploadfileaction.cfm"
  enctype="multipart/form-data"
  method="post">
 <p>Enter the complete path and filename of the file to upload:
 <input type="file"
   name="FiletoUpload"
   size="45">
 </p>
 <input  type="submit"
   value="Upload">
</form>
</body>
```

2. Save the file as uploadfileform.cfm in the myapps directory under your *web_root* and view it in the browser.

> **Note**
>
> The form does not work until you write an action page for it (see the next procedure).

© 2014 Adobe Systems Incorporated. All rights reserved.

Reviewing the code

The following table describes the code and its function:

Code	Description
`<form action="uploadfileaction.cfm" enctype="multipart/form-data" method="post">`	Create a form that contains file selection fields for upload by the user. The `action` attribute value specifies the ColdFusion template that processes the submitted form. The `enctype` attribute value tells the server that the form submission contains an uploaded file. The `method` attribute is set to `post` to submit a ColdFusion form.
`<input type="file" name="FiletoUpload" size="45">`	Allow the user to specify the file to upload. The `file` ty pe instructs the browser to prepare to read and transmit a file from the user system to your server. It automatically includes a Browse button to let the user look for the file instead of manually entering the entire path and filename.

The user can enter a file path or browse the system and select a file to send.

1. Create a ColdFusion page with the following content:

```
<html>
<head> <title>Upload File</title> </head>
<body>
<h2>Upload File</h2>

<cffile action="upload"
  destination="c:\temp\"
  nameConflict="overwrite"
  fileField="Form.FiletoUpload">

<cfoutput>
You uploaded #cffile.ClientFileName#.#cffile.ClientFileExt#
   successfully to #cffile.ServerDirectory#.
</cfoutput>

</body>
</html>
```

2. Change the following line to point to an appropriate location on your server:

```
destination="c:\temp\"
```

© 2014 Adobe Systems Incorporated. All rights reserved.

> **Note**
>
> This directory must exist on the server.

1. Save the file as uploadfileaction.cfm in the myapps directory under your *web_root*.
2. View uploadfileform.cfm in the browser, enter a file to upload, and submit the form.The file you specified uploads.

Reviewing the code

The following table describes the code and its function:

Code	Description
`<cffile action="upload"`	Output the name and location of the uploaded file on the client machine.
`destination="c:\temp\"`	Specify the destination of the file.
`nameConflict="overwrite"`	If the file exists, overwrite it.
`fileField="Form.FiletoUpload">`	Specify the name of the file to upload. Do not enclose the variable in number signs.
`You uploaded #cffile.ClientFileName#.#cffile.ClientFileExt# successfully to #cffile.ServerDirectory#.`	Inform the user of the file that was uploaded and its destination. For information on scope variables, see *Evaluating the results of a file upload* below.

> **Note**
>
> This example performs no error checking and does not incorporate any security measures. Before deploying an application that performs file uploads, ensure that you incorporate both error handling and security. For more information, see Securing Applications and Handling Errors.

Resolving conflicting filenames

When you save a file to the server, a file with the same name could exist. To resolve this problem, assign one of

© 2014 Adobe Systems Incorporated. All rights reserved.

these values to the `nameConflict` attribute of the `cffile` tag:

- **Error** (default) ColdFusion stops processing the page and returns an error. The file is not saved.
- **Skip** Allows custom behavior based on file properties. The tag does not save the file or return an error.
- **Overwrite** Overwrites a file that has the same name as the uploaded file.
- **MakeUnique** Generates a unique filename for the uploaded file. The name is stored in the file object variables serverFile and serverFileName. You can use this variable to record the name used when the file was saved. The unique name might not resemble the attempted name. For more information on file upload status variables, see *Evaluating the results of a file upload* below.

Controlling the type of file uploaded

For some applications, you could want to restrict the type of file that is uploaded, for example, to not accept graphic files in a document library.
You use the `accept` attribute to restrict the type of file that you allow in an upload. When an `accept` qualifier is present, the uploaded file MIME content type must match the criteria specified or an error occurs. The `accept` attrib ute takes a comma-separated list of MIME data names, optionally with wildcards.
The browser determines the file MIME type. Common types, such as image/gif and text/plain, are registered in the browser.

> **Note**
>
> Current versions of Microsoft Internet Explorer and Netscape support MIME type associations. Other browsers and earlier versions might ignore these associations.

ColdFusion saves any uploaded file if you omit the `accept` attribute or specify "**/**". You can restrict the file types, as demonstrated in the following examples.
The following `cffile` tag saves an image file only if it is in the GIF format:

```
<cffile action="Upload"
 fileField="Form.FiletoUpload"
 destination="c:\uploads\"
 nameConflict="Overwrite"
 accept="image/gif">
```

The following `cffile` tag saves an image file only if it is in GIF or JPEG format:

```
<cffile action="Upload"
 fileField="Form.FiletoUpload"
 destination="c:\uploads\"
 nameConflict="Overwrite"
 accept="image/gif, image/jpeg">
```

> **Note**
>
> If you receive an error like "The MIME type of the uploaded file (image/jpeg) was not accepted by the server", enter `accept="image/jpeg"` to accept JPEG files.

This `cffile` tag saves any image file, regardless of the format:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cffile action="Upload"
 fileField="Form.FiletoUpload"
 destination="c:\uploads\"
 nameConflict="Overwrite"
 accept="image/*">
```

Setting file and directory attributes

In Windows, you specify file attributes using `attributes` attribute of the cffile tag. In UNIX, you specify file or directory permissions using the `mode` attribute of the `cffile` or cfdirectory tag.

Windows

In Windows, you can set the following file attributes:

- Hidden
- Normal
- ReadOnly
  To specify several attributes in CFML, use a comma-separated list for the `attributes` attribute; for example, `attributes="ReadOnly,Hidden"`. If you do not use the `attributes` attribute, the existing attributes of the file are maintained. If you specify any other attributes in addition to Normal, the additional attribute overrides the Normal setting.

UNIX

In UNIX, you can individually set permissions on files and directories for each of three types of users' owner, group, and other. You use a number for each user type. This number is the sum of the numbers for the individual permissions allowed. Values for the `mode` attribute correspond to octal values for the UNIX `chmod` command:

- 4 = read
- 2 = write
- 1 = execute

You enter permissions values in the `mode` attribute for each type of user: owner, group, and other in that order. For example, use the following code to assign read permissions for everyone:

```
mode=444
```

To give a file or directory owner read/write/execute permissions and read-only permissions for everyone else:

```
mode=744
```

Evaluating the results of a file upload

After a file upload is completed, you can retrieve status information using file upload status variables. This status information includes data about the file, such as its name and the directory where it was saved.
You can access file upload status variables using dot notation, using either file.varname or cffile.varname. Although

© 2014 Adobe Systems Incorporated. All rights reserved.

you can use either the File or cffile prefix for file upload status variables, cffile is preferred; for example, cffile.ClientDirectory. The File prefix is retained for backward compatibility.

> **Note**
>
> File status variables are read only. They are set to the results of the most recent `cffile` operation. If two cffile tags execute, the results of the first are overwritten by the subsequent `cffile` operation.

The following table describes the file upload status variables that are available after an upload:

Variable	Description
attemptedServerFile	Initial name that ColdFusion uses when attempting to save a file; for example, myfile.txt. (see Resolving conflicting filenames).
clientDirectory	Directory on the client system from which the file was uploaded.
clientFile	Full name of the source file on the client system with the filename extension; for example, myfile.txt.
clientFileExt	Extension of the source file on the client system without a period; for example, txt (not .txt).
clientFileName	Name of the source file on the client system without an extension; for example, myfile.
contentType	MIME content type of the saved file; for example, `image` for `image/gif`.
contentSubType	MIME content subtype of the saved file; for example, `gif` for `image/gif`.
dateLastAccessed	Date that the uploaded file was last accessed.
fileExisted	Indicates (Yes or No) whether the file existed with the same path.
fileSize	Size of the uploaded file.
fileWasAppended	Indicates (Yes or No) whether ColdFusion appended the uploaded file to an existing file.
fileWasOverwritten	Indicates (Yes or No) whether ColdFusion overwrote a file.

© 2014 Adobe Systems Incorporated. All rights reserved.

fileWasRenamed	Indicates (Yes or No) whether the uploaded file was renamed to avoid a name conflict.
fileWasSaved	Indicates (Yes or No) whether ColdFusion saved the uploaded file.
oldFileSize	Size of the file that was overwritten in the file upload operation. Empty if no file was overwritten.
serverDirectory	Directory where the file was saved on the server.
serverFile	Full name of the file saved on the server with the filename extension; for example, myfile.txt.
serverFileExt	Extension of the file saved on the server without a period; for example, txt (not .txt).
serverFileName	Name of the file saved on the server without an extension; for example, myfile.
timeCreated	Date and time the uploaded file was created.
timeLastModified	Date and time of the last modification to the uploaded file.

Moving, renaming, copying, and deleting server files

With the cffile tag, you can create application pages to manage files on your web server. You can use the tag to move files from one directory to another, rename files, copy a file, or delete a file.
The examples in the following table show static values for many of the attributes. However, the value of all or part of any attribute in a `cffile` tag can be a dynamic parameter.

Action	Example code
Move a file	`<cffile action="move" source="c:\files\upload\KeyMemo.doc" destination="c:\files\memo\">`
Rename a file	`<cffile action="rename" source="c:\files\memo\KeyMemo.doc" destination="c:\files\memo\OldMemo.doc">`

© 2014 Adobe Systems Incorporated. All rights reserved.

Copy a file	`<cffile action="copy" source="c:\files\upload\KeyMemo.doc" destination="c:\files\backup\">`
Delete a file	`<cffile action="delete" file="c:\files\upload\oldfile.txt" >`

This example sets the ReadOnly flag bit for the uploaded file:

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

Note

Ensure that you include the trailing slash () when you specify the destination directory. Otherwise, ColdFusion treats the last element in the path as a filename. This rule only applies to copy actions.

Reading, writing, and appending to a text file

In addition to managing files on the server, you can use the cffile tag to read, create, and modify text files. As a result, you can do the following things:

- Create log files. (You can also use `cflog` to create and write to log files.)
- Generate static HTML documents.
- Use text files to store information that can be incorporated into web pages.

Reading a text file

You can use the `cffile` tag to read an existing text file. The file is read into a local variable that you can use anywhere in the application page. For example, you could read a text file and then insert its contents into a database, or you could read a text file and then use one of the string replacement functions to modify the contents.

Read a text file

1. Create a ColdFusion page with the following content:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<html>
<head>
 <title>Read a Text File</title>
</head>

<body>
Ready to read the file:<br>
<cffile action="read"
 file="C:\inetpub\wwwroot\mine\message.txt"
 variable="Message">

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

2. Replace C:\inetpub\wwwroot\mine\message.txt with the location and name of a text file on the server.
3. Save the file as readtext.cfm in the myapps directory under your *web_root* and view it in the browser.

Writing a text file on the server

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

Create a form in to capture data for a text file

1. Create a ColdFusion page with the following content:

```
<html>
<head>
 <title>Put Information into a Text File</title>
</head>

<body>
<h2>Put Information into a Text File</h2>

<form action="writetextfileaction.cfm" method="Post">
 <p>Enter your name: <input type="text" name="Name" size="25"></p>
 <p>Enter the name of the file: <input type="text" name="FileName"
size="25">.txt</p>
 <p>Enter your message:
 <textarea name="message"cols=45 rows=6></textarea>
 </p>
 <input type="submit" name="submit" value="Submit">
</form>
</body>
</html>
```

2. Save the file as writetextfileform.cfm in the myapps directory under your *web_root*.

© 2014 Adobe Systems Incorporated. All rights reserved.

Note

The form does not work until you write an action page for it (see the next procedure).

Write a text file

1. Create a ColdFusion page with the following content:

```
<html>
<head>
 <title>Write a Text File</title>
</head>
<body>
<cffile action="write"
 file="C:\inetpub\wwwroot\mine\#Form.FileName#.txt"
 output="Created By: #Form.Name#
#Form.Message# ">
</body>
</html>
```

2. Modify the path C:\inetpub\wwwroot\mine\ to point to a path on your server.
3. Save the file as writetextfileaction.cfm in the myapps directory under your *web_root*.
4. View the file writetextfileform.cfm in the browser, enter values, and submit the form.The text file is written to the location you specified. If the file exists, it is replaced.

Appending a text file

You can use the cffile tag to append additional text to the end of a text file; for example, when you create log files.

Append a text file

1. Open the writetextfileaction.cfm file.
2. Change the value for the `action` attribute from `write` to `append`so that the file appears as follows:

```
<html>
<head>
 <title>Append a Text File</title>
</head>
<body>
<cffile action="append"
 file="C:\inetpub\wwwroot\mine\message.txt"
 output="Appended By: #Form.Name#">
</body>
</html>
```

3. Save the file as writetextfileaction.cfm in the myapps directory under your *web_root*.
4. View the file in the browser, enter values, and submit the form.The appended information displays at the end of the text file.

© 2014 Adobe Systems Incorporated. All rights reserved.

Using cfdirectory

Use the cfdirectory tag to return file information from a specified directory and to create, delete, and rename directories. When listing directory contents or deleting a directory, you can optionally use the `recurse` attribute to access or delete all subdirectories.
As with the `cffile` tag, you can disable `cfdirectory` processing in the ColdFusion Administrator. For details on the syntax of this tag, see the *CFML Reference*.

Returning file information

When you use the `action="list"` attribute setting, the `cfdirectory` returns a query object as specified in the `name` attribute. The `name` attribute is required when you use the `action="list"` attribute setting. This query object contains result columns that you can reference in a `cfoutput` tag, using the value specified in the `name` attribute:

- **name** Directory entry name.
- **directory** Directory containing the entry.
- **size** Directory entry size.
- **type** File type: File or Dir.
- **dateLastModified** Date an entry was last modified.
- **attributes** (Windows only) File attributes, if applicable.
- **mode**(UNIX only) The octal value representing the permissions setting for the specified directory.

> **Note**
>
> ColdFusion supports the ReadOnly and Hidden values for the `attributes` attribute for `cfdirectory` sorting.

Depending on whether your server is on a UNIX system or a Windows system, either the Attributes column or the Mode column is empty. Also, you can specify a filename in the `filter` attribute to get information on a single file.The following procedure describes how to create a ColdFusion page in which to view directory information.

View directory information

1. Create a ColdFusion page with the following content:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<html>
<head>
 <title>List Directory Information</title>
</head>

<body>
<h3>List Directory Information</h3>
<cfdirectory
 directory="c:\inetpub\wwwroot\mine"
 name="mydirectory"
 sort="size ASC, name DESC, datelastmodified">

<table cellspacing=1 cellpadding=10>
<tr>
 <th>Name</th>
 <th>Size</th>
 <th>Type</th>
 <th>Modified</th>
 <th>Attributes</th>
 <th>Mode</th>
</tr>
<cfoutput query="mydirectory">
<tr>
 <td>#mydirectory.name#</td>
 <td>#mydirectory.size#</td>
 <td>#mydirectory.type#</td>
 <td>#mydirectory.dateLastModified#</td>
 <td>#mydirectory.attributes#</td>
 <td>#mydirectory.mode#</td>
</tr>
</cfoutput>
</table>

</body>
</html>
```

2. Modify the path C:\inetpub\wwwroot\mine so that it points to a directory on your server.
3. Save the file as directoryinfo.cfm in the myapps directory under your *web_root* and view it in the browser.

© 2014 Adobe Systems Incorporated. All rights reserved.

Using cfcontent

The cfcontent tag downloads files from the server to the client. You can use this tag to set the MIME type of the content returned by a ColdFusion page and, optionally, define the name of a file for the current page to download. By default, ColdFusion returns a MIME content type of text/html so that a web browser renders your template text as a web page.
As with the `cffile` and `cfdirectory` tags{{,}} you can disable processing in the ColdFusion Administrator.

About MIME types

A *MIME type* is a label that identifies the contents of a file. the browser uses the MIME type specification to determine how to interact with the file. For example, the browser could open a spreadsheet program when it encounters a file identified by its MIME content type as a spreadsheet file.
A MIME content type consists of `"type/subtype"` format. The following are common MIME content types:

- text/html
- image/gif
- application/pdf

Changing the MIME content type with cfcontent

You use the `cfcontent` tag to change the MIME content type that returns to the browser along with the content generated from your ColdFusion page.
The `cfcontent` tag has one required attribute, `type`, which defines the MIME content type returned by the current page.

Change the MIME content type with cfcontent

1. Create an HTML page with the following content:

```
<h1>cfcontent_message.htm</h1>

<p>This is a <em>test message</em> written in HTML.</p>
<p>This is the <em>second paragraph</em> of the test message.
As you might expect, it is also written in HTML.</p>
```

2. Save the file as cfcontent_message.htm in the myapps directory under your *web_root*.The ColdFusion file that you write in steps 3 through 7 calls this file.
3. Create a ColdFusion page with the following content:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<html>
<head>
<title>cfcontent Example</title>
</head>

<body>
<h3>cfcontent Example</h3>

<cfcontent
 type = "text/html"
 file = "C:\CFusion\wwwroot\myapps\cfcontent_message.htm"
 deleteFile = "No">
</body>
</html>
```

4. If necessary, edit the `file =` line to point to your myapps directory.
5. Save the file as cfcontent.cfm in the myapps directory under your *web_root* and view it in the browser.The text of the called file (cfcontent_message.htm) displays as normal HTML.
6. In cfcontent.cfm, change `type = "text/html"` to `type = "text/plain"`.
7. Save the file and view it in the browser (refresh it if necessary).The text displays as unformatted text, in which HTML tags are treated as text.
   The following example shows how the cfcontent tag can create an Excel spreadsheet that contains your data.

Create an Excel spreadsheet with cfcontent

1. Create a ColdFusion page with the following content:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<!--- Use cfsetting to block output of HTML
outside cfoutput tags. --->
<cfsetting enablecfoutputonly="Yes">

<!--- Get employee info. --->
<cfquery name="GetEmps" datasource="cfdocexamples">
 SELECT * FROM Employee
</cfquery>

<!--- Set content type. --->
<cfcontent type="application/msexcel">

<!--- Suggest default name for XLS file. --->
<!--- "Content-Disposition" in cfheader also ensures
relatively correct Internet Explorer behavior. --->
<cfheader name="Content-Disposition" value="filename=Employees.xls">

<!--- Format data using cfoutput and a table.
  Excel converts the table to a spreadsheet.
  The cfoutput tags around the table tags force output of the HTML when
  using cfsetting enablecfoutputonly="Yes" --->
<cfoutput>
 <table cols="4">
  <cfloop query="GetEmps">
   <tr>
    <td>#Emp_ID#</td>
    <td>#FirstName#</td>
    <td>#LastName#</td>
   </tr>
  </cfloop>
 </table>
</cfoutput>
```

2. Save the file as employees_to_excel.cfm in the myapps directory under your *web_root* and view it in the browser.The data appears in an Excel spreadsheet.

#back to top

© 2014 Adobe Systems Incorporated. All rights reserved.

Using Event Gateways

Adobe ColdFusion provides event gateways, which you can use when writing applications. You configure an event gateway for an application and deploy the application.

To use event gateways, you should have a thorough knowledge of ColdFusion development concepts and practices, including ColdFusion components (CFCs). To write applications for custom gateways that are not provided in ColdFusion, you must also know the details of the event gateway you are using, including its requirements.

© 2014 Adobe Systems Incorporated. All rights reserved.

About event gateways

ColdFusion event gateways are ColdFusion elements that let ColdFusion react to or generate external events or messages in an asynchronous manner. Event gateways let a ColdFusion application handle information that does not come through an HTTP request. For example, you can use event gateways to handle instant messages, short messages from mobile devices, or messages sent to a TCP/IP port.
The event gateway mechanism has the following major features:

- ColdFusion event gateways do not require HTTP requests. ColdFusion developers can write ColdFusion gateway applications without using any CFM pages (just CFCs).
- ColdFusion CFCs can use event gateways to listen for and respond directly to external events.
- Event gateways operate asynchronously. A gateway typically gets a message and dispatches it for processing, without requiring or waiting for a response.
- ColdFusion developers can create event gateways to handle any type event that a Java application can receive.
  ColdFusion includes several product-level event gateways, such as a gateway for the XMPP (Extensible Messaging and Presence Protocol) instant messaging protocol. Adobe also provides the source for several example gateways, such as a generalized socket gateway, that you can extend to handle your specific needs. You can also write your own gateways in Java to handle other event or messaging technologies supported by the Java runtime or by third-party providers, such as gateways for additional instant messaging protocols, gateways for specific ERP systems, or other protocols, such as NNTP.

Using event gateways

Because event gateways provide a generalized asynchronous messaging mechanism, you can use them with many kinds of event or messaging resources. For example, ColdFusion includes gateways (either product quality, or lighter weight example gateways) for communicating between ColdFusion applications and the following types of resources:

- Mobile phones and other devices that support short messaging services (SMS)
- XMPP or IBM Sametime Instant message clients
- Java Sockets (which let your ColdFusion application communicate with TCP/IP-based devices and programs, such as Telnet terminal clients).
- Java Messaging Service (JMS) resources, such as storefront sales order handling systems.
  Event gateways are not limited to sending or receiving information using communications protocols. For example, ColdFusion includes an example event gateway that monitors changes to a directory and invokes a CFC method whenever the directory changes. ColdFusion also includes an event gateway that lets a CFML application "call" a CFC asynchronously and continue processing without getting a response from the CFC.
  Just as you can create event gateways that serve many different event or messaging based technologies, you can write many kinds of applications that use them. Just a few examples of possible gateway uses include the following.

Server to client push examples

- An application that sends an instant message (IM) or SMS text message to a person who can approve a purchase order, get a response, and mark the purchase order as approved or denied.
- A bot that notifies users through their preferred messaging method (mobile phone, instant messaging, or even e-mail) when watch list stock goes up, and offers to buy or sell the stock immediately.
- An application that authenticates web users by sending them an SMS message that includes code that they must to enter into the browser to proceed.

Client to server examples

- A menu-based SMS application that lets users get information from any of several web service data providers. ColdFusion includes an SMS menuing example int the gateways/cfc directory.
- An instant messaging application that takes messages from users to technical support and assigns and

© 2014 Adobe Systems Incorporated. All rights reserved.

directs the messages to the most available support staff member. The application could also log the user ID and session, and you could use ColdFusion to generate usage reports.
- A directory lookup robot IM "buddy" that responds to messages chat contain an employee name with the employee's phone number or buddy ID.

Server to serve examples

- A JMS subsystem that publishes status updates that business intelligence systems consume.
- A system that monitors and publishes download events from a website.

Event gateway terms and concepts

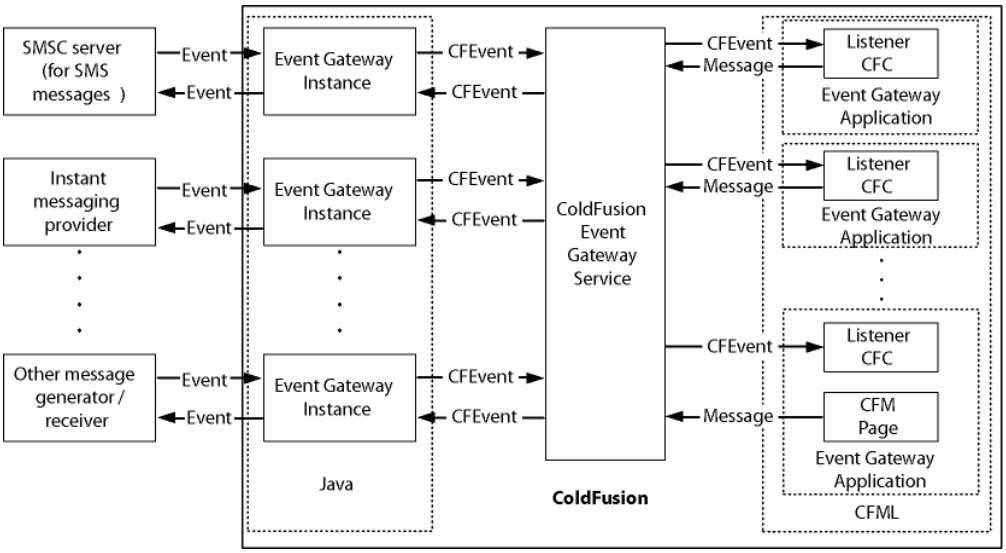
This document uses the following terms when referring to event gateways:

- **Event** A trigger that ColdFusion can receive from an external source. ColdFusion event gateways receive events.
- **Message** The information provided by an event. In ColdFusion, a message is the data structure that the event gateway receives when an event is triggered.
- **Event gateway** Java code that receives events and sends them to and from ColdFusion application code. This document uses the term *event gateway*, without the word type or instance, for the general concept of a ColdFusion event gateway. Where the context makes the meaning obvious, the term can also mean event gateway type or event gateway instance.
- **Event gateway type** A specific *event gateway* implementation, represented by a Java class. Each event gateway type handles messages belonging to a particular a communications method or protocol, such as short message service (SMS), an instant messaging protocol, or Sockets. You generally have one event gateway type per communication protocol. You configure each event gateway type on the Gateway Types page in the Event Gateways area in the ColdFusion Administrator.
- **Event gateway instance** A specific instance of an *event gateway type* class. You configure each event gateway instance on the ColdFusion Gateway Instances page by specifying the event gateway type, an ID, the path to the *event gateway application* CFC that uses this instance, and a configuration file (if needed for the selected event gateway type). You can have multiple event gateway instances per event gateway type, for example, for different event gateway applications.
- **Event gateway application** One or more CFCs and any supporting CFM pages that handle events from an *e vent gateway instance* and send messages using the event gateway instance. The event gateway application is not part of an event gateway instance, but the code that is responsible for processing event messages to and from the instance.
- **Event gateway listener** Code in an *event gateway* that receives *events* from an event source and passes them to the ColdFusion_ gateway service_ for delivery to a CFML *listener CFC*.
- **Listener CFC** A CFC that contains one or more methods that respond to incoming messages from one or more event *gateway instances*. Part of an *event gateway application.*
- **ColdFusion gateway service** The part of ColdFusion that provides underlying support for *event gateways*, including a path between an *event gateway instance* and *listener CFCs*.

How event gateway applications work

The following diagram shows the architecture of ColdFusion event gateway applications:

© 2014 Adobe Systems Incorporated. All rights reserved.



How event gateways interact

Typically, a ColdFusion event gateway instance, a Java object, listens for events coming from an external provider. For example, a general socket event gateway listens for messages on an IP socket, and an SMS event gateway receives messages from an SMSC server.

Each event gateway instance communicates with one or more listener CFCs through the ColdFusion event gateway service. The listener CFCs receive CFEvent object instances that contain the messages, process them, and can send responses back to the event gateway, which can send the messages to the external resources.

Alternatively, a ColdFusion application can initiate a message by calling a ColdFusion function that sends the message to the event gateway. The event gateway then forwards the message to an external resource, such as an instant messaging server. A CFC in the application listens for any responses to the sent message.

Some event gateways can be one way: they listen for a specific event and send it to a CFC, or they get messages from a ColdFusion function and dispatch it, but they do not do both. The example *DirectoryWatcherGateway* discussed in Example event gateways listens for events only, and the asynchronous CFML event gateway receives messages from CFML only. (You could even say that the directory watcher gateway doesn't listen for events; it creates its own events internally by periodically checking the directory state.) For information on the asynchronous CFML event gateway, see Using the CFML event gateway for asynchronous CFCs.

Event gateway structure

Java programmers develop ColdFusion event gateways by writing Java classes that implement the coldfusion.eventgateway.Gateway interface. ColdFusion event gateways normally consist of one or more threads that listen for events from an event provider, such as a Socket, an SMSC server, or some other source. The event gateway sends event messages to the ColdFusion event gateway service message queue, and provides a method that gets called when an event gateway application CFC or CFM page sends an outgoing message.

The event gateway class can also do the following:

- Provide the ColdFusion application with access to a helper class that provides event gateway-specific services, such as buddy-list management or connection management.
- Use a file that specifies configuration information, such as IP addresses and ports, passwords, and other ID information, internal time-out values, and so on.

About developing event gateway applications

ColdFusion application developers write applications that use event gateways. The person or company that provides the event gateway supplies gateway-specific information to the ColdFusion developer. This information must include the structure and contents of the messages that the ColdFusion application receives and sends to the event gateway, plus any information about configuration files or helper methods that the ColdFusion application could use.

© 2014 Adobe Systems Incorporated. All rights reserved.

The ColdFusion developer writes a CFC that listens for messages. Many event gateway types send messages to a listener CFC method named `onIncomingMessage`. A minimal event gateway application could implement only this single method. More complex event gateway types can require multiple CFC listener methods. For example, the ColdFusion XMPP IM event gateway sends user messages to the `onIncomingMessage` CFC method, but sends requests to add buddies to the `onAddBuddyRequest` CFC method.

Depending on the event gateway and application types, the event gateway application could include CFM pages or CFC methods to initiate outgoing messages. The application also could use an event gateway-specific GatewayHelper object to do tasks such as getting buddy lists in IM applications or getting the status of a messaging server.

The ColdFusion application developer also configures an event gateway instance in the ColdFusion Administrator, and possibly in a configuration file. The ColdFusion Administrator configuration information specifies the listener CFC that handles the messages from the event gateway and other standard event gateway configuration details. The configuration file, if necessary, contains event gateway type-specific configuration information.

© 2014 Adobe Systems Incorporated. All rights reserved.

Event gateway facilities and tools

ColdFusion provides several features and tools for developing and deploying event-handling applications, these including the following:

- Standard event gateways.
- Development tools and example code.
- A gateway directory structure configured for use by custom event gateways and event gateway applications. This directory also contains the example code.
- An event gateway-specific log file
- Three pages in the ColdFusion Administrator for managing event gateways.

Standard event gateways

Adobe provides several event gateways as part of ColdFusion. These event gateways support the following messaging protocols:

- **SMS (Short Message Service)** A system designed for exchanging short, often text, messages with wireless devices, such as mobile phones or pagers. For detailed information on using the SMS event gateway, see Using the SMS Event Gateway.
- *XMPP (Extensible Messaging and Presence Protocol): * An open, XML-based protocol for instant messaging. For detailed information on using the XMPP event gateway, see Using the Instant Messaging Event Gateways.
- **IBM Lotus Instant Messaging** (commonly referred to as Lotus Sametime):The IBM product for real-time collaboration. For detailed information on using the Lotus Sametime event gateway, see Using the Instant Messaging Event Gateways. ColdFusion also provides an event gateway, the CFML asynchronous event gateway, that lets a CFML application invoke a CFC method asynchronously. This event gateway does not follow the model of providing a mechanism for exchanging messages with resources outside ColdFusion. Instead, it provides a one-way path for invoking CFCs when an application does not require (indeed, cannot receive) a return value from the CFC. For detailed information on using the CFML asynchronous event gateway, see Using the CFML event gateway for asynchronous CFCs.

Development tools and example code

ColdFusion provides the following tools and example code for developing your own event gateways and event gateway applications:

- An SMS client (phone simulator) and a short message service center (SMSC) server simulator, for developing SMS applications without requiring an external SMS provider.
- Four sample event gateways with source code:
  - A template for an empty event gateway that contains a skeleton on which you can build your own event gateways
  - A TCP/IP socket event gateway that listens on a TCP/IP port
  - A directory watcher event gateway that monitors changes to the contents of a directory
  - A Java Messaging Service (JMS) gateway that acts as a JMS consumer or producer.
- Several sample applications, including the following:
  - A menu application that uses an inquiry-response drill-down menu to provide services such as weather reports and stock quotes.
  - A simple echo application that sends back the messages that it receives.
  - A temperature converter, an asynchronous logging application.
  - An application that returns employee phone number and other information.
  The chapters in this document use these example applications.
- JavaDoc documentation for the Java interfaces and classes that you use to create gateways.
  For more information on these examples, see Using the example event gateways and gateway applications.

© 2014 Adobe Systems Incorporated. All rights reserved.

The ColdFusion gateway directory

The ColdFusion installation includes a *cf_root*\WEB-INF\cfusion\gateway directory on J2EE configurations, or *cf_root*\gateway directory on server configurations. This directory contains all the code for ColdFusion example event gateways and example event gateway applications, and example configuration files for use by standard ColdFusion event gateways. You do not have to place your event gateways, event gateway application CFCs, or event gateway configuration files in this directory, but ColdFusion is configured to find event gateways and CFCs that you place there.

The following table lists the event gateway directory subdirectories, their purpose, and their initial contents. For more information on using the example event gateways and applications, see Using the example event gateways and gateway applications.

Directory	Purpose
cfc	Event gateway application CFCs. ColdFusion is installed with an Administrator Mapping between /gateway and this cfc directory.
cfc/examples	Code for the ColdFusion sample applications.
config	Configuration files for all ColdFusion event gateways, including standard ColdFusion event gateways, such as SMS, and example event gateways, such as the directory watcher event gateway.
doc/api	JavaDoc for the Gateway, and GatewayHelper interfaces, and the CFEvent, GatewayServices, and GenericGateway classes that gateway developer use when writing gateways. This documentation is a subset of the information in Gateway development interfaces and classes in the *CFML Reference*.
lib	Executable code for example and user-developed event gateway classes. The ColdFusion class loader includes this directory on its classpath and includes any JAR files that are in that directory on the class path. The examples.jar file in this directory contains the class files for the DirectoryWatcherGateway, EmptyGateway, and SocketGateway classes.
src/examples	Source code for the example event gateway classes that Adobe provides. Includes the EmptyGateway.java file and the following subdirectories: • socket: Socket gateway source files • watcher: directory watcher gateway source files • JMS: JMS gateway source files

The eventgateway.log file

Event gateways provided with ColdFusion log event gateway errors and events to the *cf_root*\WEB-INF\cfusion\logs\eventgateway.log file on J2EE configurations, or the *cf_root*\logs\eventgateway.log file on server configurations.

© 2014 Adobe Systems Incorporated. All rights reserved.

ColdFusion includes methods that let any event gateway use this file. This log file can be useful in debugging event gateways and event gateway applications.

ColdFusion Administrator event gateway pages

The ColdFusion Administrator includes a Gateways section with three pages for managing event gateways:

- Settings
- Gateway types
- Gateway Instances

  The Settings page lets you enable and disable support for event gateways, specify the number of threads that ColdFusion can devote to processing events, specify the maximum number events that ColdFusion can hold in its event queue (which holds events that are waiting to be processed) and start the SMS test server.

  The Gateway Types page lets you add, remove, and configure event gateway types by specifying a name, a Java class, and startup time-out behavior.

> **Note**
>
> The gateway type name in the ColdFusion Administrator does not have to be the same as the gateway type that is used in the gateway Java code and the `CFEvent` data structure; however, use the same name in both places for consistency.

The Gateway Instances page lets you add, remove, configure, start, and stop individual event gateway instances. You configure an event gateway instance by specifying a unique ID, the gateway type, one or more listener CFC paths, a configuration file (not required for all gateway types), and a startup mode (manual, automatic, or disabled).

© 2014 Adobe Systems Incorporated. All rights reserved.

Structure of an event gateway application

To develop an event gateway application, you create and use some or all of the following elements:

- One or more listener CFCs that handle any incoming messages and send any necessary responses.
- In some applications, ColdFusion pages that generate outgoing messages directly.
- An event gateway instance configuration in the ColdFusion Administrator. This configuration could require a separate event gateway configuration file.
- In some applications, a GatewayHelper object to provide access to additional features of the protocol or technology; for example, to manage instant messaging buddy lists.

The role of the listener CFC

All incoming event messages must be handled by one or more listener CFCs. You specify the listener CFCs when you configure an event gateway in the ColdFusion Administrator. Specify at least one CFC in the administrator. Some gateway types can use more than one CFC. By default, the ColdFusion event gateway service delivers events by calling the `onIncomingMessage` method of the CFC.
The event gateway developer must inform the event gateway application developer of methods that the listener CFC must implement (could be only the `onIncomingMessage` method) and of the structure and contents of the event message data, contained in the CFEvent instance, that the listener CFC must handle. Outgoing messages have the same event message data structure as incoming messages.
Many gateways let the listener CFCs send a response by calling the `cfreturn` function, but ColdFusion does not require a return value. Listener CFCs can also use the `SendGatewayMessage` function, which provides more flexibility than the `cfreturn` tag.

The role of ColdFusion pages

ColdFusion CFM pages cannot receive event messages. However, they can send messages using an event gateway. Therefore, an event gateway application that initiates outgoing messages could use one or more `SendGatewayMessage` functions to send the messages. An application that sends an SMS message to notify users when a package ships, for example, could use the `SendGatewayMessage` function to send the notification.

The role of the ColdFusion Administrator

The Gateway Instances page in the ColdFusion Administrator associates a specific event gateway instance with one or more listener CFCs that processes messages from the event gateway. It tells the ColdFusion event gateway service to send messages received by the event gateway to the listener CFC. It also lets you specify a configuration file for the event gateway instance and whether to start the event gateway instance (and therefore any responder application) when ColdFusion starts. For more information on using the Administrator, see the ColdFusion Administrator online Help.

The role of the GatewayHelper object

A ColdFusion event gateway provides an information conduit: at its most basic, it receives and dispatches event messages. In some cases, however, an event gateway must provide additional functionality. An instant messaging event gateway, for example, must provide such services as managing buddies and providing status information. To support such use, an event gateway can enable access to a GatewayHelper object. The event gateway developer writes a Java class that provides the necessary utility routines as Java methods, and ColdFusion application developers can get an instance of the class by calling the CFML `GetGatewayHelper` method. The application calls the GatewayHelper object methods using normal ColdFusion object access techniques. The ColdFusion instant messaging event gateways and the example socket event gateway provide GatewayHelper objects.

© 2014 Adobe Systems Incorporated. All rights reserved.

Configuring an event gateway instance

Before you develop or deploy an event gateway application, use the ColdFusion Administrator to configure an event gateway instance that handles the event messages. You specify the following information:

- An event gateway ID to identify the specific event gateway instance. You use this value in the CFML `GetGatewayHelper` and `SendGatewayMessage` functions.
- The event gateway type, which you select from the available event gateway types, such as SMS or Socket.
- The absolute path to the listener CFC or CFCs that handles incoming messages. If you have multiple listener CFCs, enter the paths separated by commas. Specify absolute file paths, even if you place the CFCs in the ColdFusion gateway\cfc directory.
- A configuration file, if necessary for this event gateway type or instance.
- The event gateway start-up status; one of the following:

- **Automatic** Start the event gateway when ColdFusion starts.
- **Manual** Do not start the event gateway with ColdFusion, but allow starting it from the ColdFusion Administrator Event Gateways list.
- **Disabled** Do not allow the event gateway to start.

© 2014 Adobe Systems Incorporated. All rights reserved.

Developing an event gateway application

All event gateway applications handle information. They exchange event messages, and possibly other types of information, with other resources. Event gateway applications require a listener CFC to handle events that are sent to the event gateway. Event gateway applications can also use the following code elements:

- `SendGatewayMessage` CFML functions to send messages from outside the listener CFC (or, optionally, from the CFC)
- GatewayHelper objects
- The eventgateway log file

Event gateway application models

Event gateway applications follow one or both of the following models:

- **Responder applications** Where event messages from external sources initiate a response from a ColdFusion listener CFC
- **Initiator applications** Where a ColdFusion application generates event messages to send out using the event gateway Unlike other ColdFusion applications, responder applications are request-free. They do not have CFM pages, just CFCs, and they do not respond to HTTP requests. Instead, ColdFusion the event gateway service deliver the event messages directly to the listener CFC, and the CFC listener method returns any response directly to the event gateway service. Applications that allow mobile phone owners to get a news feed, check for text messages, or request other forms of information follow this model.
  Initiator applications are like most ColdFusion applications. At some point, ColdFusion executes a CFM page in response to a request. (The ColdFusion Administrator Scheduled Tasks page can initiate the request.) ColdFusion sends a message to the event gateway when the application calls a CFML `SendGatewayMessage` function. An application that uses SMS to notify customers when orders have been shipped follows this model.

Sending information to the event gateway

A ColdFusion application can send an outgoing message to the event gateway in either of the following ways:

- In a `cfreturn` tag in the listener CFC listener method
- By calling the ColdFusion `SendGatewayMessage` function
  The first method is useful to automatically respond to incoming messages. Some complex applications that respond to incoming messages could use the `SendGatewayMessage` function either in place or in addition to the return value.
  Some event gateway types also use a GatewayHelper object to send information to external resources. For example, the ColdFusion XMPP and Lotus Sametime instant messaging event gateways provide a GatewayHelper object that can manage buddy lists, and set configuration and status information on the instant messaging server. For more information on the GatewayHelper object, see Using the GatewayHelper object. For more information on the instant messaging GatewayHelper object, see Sample IM message handling application.
  The example code in *Example event gateway CFC* shows the use of a listener return value, and indicates how event gateways can require different data in the return structure to send equivalent messages.

Developing event gateway listener CFCs

The listener CFC responds to event gateway messages. The listener CFC uses, at a minimum, the following basic software elements:

- One or more listener methods
- CFEvent structures that contain the messages
  Listener CFCs can use ColdFusion persistent scopes to store data that must be preserved over multiple CFC invocations or shared with other CFML elements.

© 2014 Adobe Systems Incorporated. All rights reserved.

Listener methods

The ColdFusion event gateway service calls one or more listener methods in the CFC to process incoming messages. The number of listener methods that you must write and their names depends on the event gateway. For example, the ColdFusion SMS event gateway requires a single listener method, which is typically named `onIncomingMessage`. (You can change the SMS event gateway listener method name in the event gateway configuration file.) The ColdFusion XMPP IM event gateway expects the listener CFC to have five methods: `onIncomingMessage`, `onAddBuddyRequest`, `onAddBuddyResponse`, `onBuddyStatus`, and `onIMServerMessage`. By default, if the event gateway does not specify the method name, ColdFusion calls the `onIncomingMessage` method of the listener CFC. For the sake of consistency, Adobe recommends you to use the `onIncomingMessage` method for any event gateway with a single listener method.
The listener method does the following:

1. Takes a single parameter, a CFEvent structure.
2. Processes the contents of the instance as required by the application.
3. Optionally, returns an outgoing message to the event gateway in a `cfreturn` tag. It can also send a message back to the event gateway by calling the ColdFusion `SendGatewayMessage` function.
   The following code shows a listener CFC with an `onIncomingMessage` method that echoes a message back to the Socket event gateway that sent it. It contains the minimum code required to process an incoming message and respond to the sender using the socket gateway.

```
<cfcomponent displayname="echo" hint="echo messages from the event gateway">
 <cffunction name="onIncomingMessage" output="no">
  <cfargument name="CFEvent" type="struct" required="yes">
  <!--- Create a return structure that contains the message. --->
  <cfset retValue = structNew()>
  <cfset retValue.DestinationID = arguments.CFEvent.OriginatorID>
  <cfset retValue.MESSAGE = "Echo: " & arguments.CFEvent.Data.MESSAGE>
  <!--- Send the return message back. --->
  <cfreturn retValue>
 </cffunction>
</cfcomponent>
```

Other event gateways require different fields in the return structure. For example, to echo a message using the SMS event gateway, you use the following lines to specify the return value:

```
<cfset retValue.command = "submit">
<cfset retValue.sourceAddress = arguments.CFEVENT.gatewayid>
<cfset retValue.destAddress = arguments.CFEVENT.originatorid>
<cfset retValue.ShortMessage = "Echo: " & arguments.CFEvent.Data.MESSAGE>
```

The CFEvent structure

The ColdFusion event gateway service passes a CFEvent structure with information about the message event to the listener method. The following table describes the structure fields:

Field	Description

© 2014 Adobe Systems Incorporated. All rights reserved.

GatewayID	The event gateway that sent the event; the value is the ID of an event gateway instance configured on the ColdFusion Administrator Gateway Instances page. If the application calls the `SendGatewayMessage` functi on to respond to the event gateway, it uses this ID as the function's first parameter.
Data	A structure containing the event data, including the message. The `Data` structure contents depend on the event gateway type.
OriginatorID	The originator of the message. The value depends on the protocol or event gateway type. Many event gateways require this value in response messages to identify the destination of the response. Identifies the sender of the message.
GatewayType	The type of event gateway, such as SMS. An application that can process messages from multiple event gateway types can use this field. This value is the gateway type name that the event Gateway class specifies. It is not necessarily the same as the gateway type name in the ColdFusion Administrator.
CFCPath	The location of the listener CFC. The listener CFC does not require this field.
CFCMethod	The listener method that ColdFusion invokes to process the event. The listener CFC does not require this field.
CFCTimeout	The time-out, in seconds, for the listener CFC to process the event request. The listener CFC does not require this field.

When a ColdFusion application responds to an event gateway message, or sends a message independently, it does not use a CFEvent structure. However, the ColdFusion event gateway service creates a Java CFEvent instance with the message data before calling the `outgoingMessage` method of the event gateway.

Using persistent scopes in listener CFCs

ColdFusion listener CFCs can use the Application, Client, and Session persistent scopes.
Because incoming event gateway messages are not associated with HTTP requests, ColdFusion uses different session and client IDs for interactions initiated by these events than for CFM Page requests, as follows:

Identifier	Structure
Session ID	*gatewayType_gatewayID_originatorID*
cfid	*originatorID*

© 2014 Adobe Systems Incorporated. All rights reserved.

cftoken	*gatewayType_gatewayID*

The *gatewayID* value is the event gateway ID that you set in the ColdFusion Administrator, and *gatewayType* and *originatorID* are the values that the event gateway sets in the CFEvent instance for an incoming message.

Application scope

The Application scope lets the CFC share data with any ColdFusion page or CFC that uses the same application name. This way, a listener CFC can use the same Application scope as CFML pages that send messages. Also, you can place multiple listener CFCs in a single directory and have them share an Application.cfc or Application.cfm file and application name.
As with all ColdFusion code, use the Application.cfc This.name variable or the cfapplication tag to set the application name. The listener CFC can use an Application.cfc or Application.cfm file if the CFC is in a directory that is in or under one of the following places:

- the ColdFusion web root
- a directory that is in the ColdFusion Administrator Mappings list.
  The ColdFusion installer creates a mapping in the ColdFusion Administrator for the gateway\cfc directory.

Client scope

The Client scope can store long-term information associated with the ID of the message sender. For example, it can store information about an IM buddy.
To use Client variables across multiple connections, your gateway type must use the same client ID for all interactions with a particular client. For many technologies and gateways, such as the IM and SMS gateways, this requirement is not an issue.

> **Note**
>
> To use Client scope variables with gateways, you must store the Client scope variables in a data source or the registry. You cannot store the variables in cookies, because gateways do not use cookies.

Session scope

The Session scope can store information required across multiple interactions. For example, an interactive IM or SMS application that uses a drill-down menu to select a service can store the information about the menu selections in the Session scope.
Event gateway sessions terminate when they time out. Because the identifiers for event sessions and clients differ from request-driven session and client identifiers, you cannot use the same Session or Client scope on a standard CFM page that sends an outgoing message and in a listener CFC that handles an incoming response to that message.
For an example of using the Session scope, see the example Menu application in the gateway\cfc\examples\menu directory.

> **Note**
>
> ColdFusion cannot create a session if an initiator application uses a `SendGatewayMessage` method to start an interaction with a client, such as an SMS user. In this case, the sending code must keep track (for example, in a database) of the messages it sends and their destinations. When a response event arrives, it can look up the originatorID to determine whether it was in response to an outgoing message.

© 2014 Adobe Systems Incorporated. All rights reserved.

Debugging event gateway CFCs

When an event gateway CFC responds to an event, it cannot display debugging information in the response page, as CFM pages do. As a result, many of the normal ColdFusion debugging techniques, including the `cfdump` tag, are not available. When you develop event gateway CFCs, consider the following debugging techniques:

- Place trace variables in the Application scope. These variables persist, and you can specify an application name for your CFC (see *Application scope*). You can inspect the Application scope contents, including your trace variables, in any CFML page that has the same application name as your CFC.
- Use cflog tags to help you trace any errors by logging significant events to a file. Also, carefully inspect the eventgateway.log and exceptions.log files that ColdFusion maintains. For more information on using the eventgateway.log file, see *The eventgateway.log file* below.
- You can simulate responses from CFCs to the event gateway by using the `SendGatewayMessage` function in a CFM page. The function's *message* parameter should contain the information that the CFC would place in its return variable.
- If you run ColdFusion from the command line, you can use the Java `System.out.println`method to write messages to the console window, as the following code shows:

```
<cfscript>
 sys = createObject("java", "java.lang.System");
 sys.out.println("Debugging message goes here");
</cfscript>
```

Note

You do not have to restart the event gateway instance when you change a CFC. ColdFusion automatically uses the updated CFC when the next event occurs.

Example event gateway CFC

The following code shows a temperature scale converter tool that can work with any of several event gateways: SMS, XMPP, Lotus Sametime, or the example Socket event gateway. Users enter a string that consists of the temperature scale (F, Fahrenheit, C, or Celsius), a comma, and a temperature on their device. The CFC converts Celsius to Fahrenheit or Fahrenheit to Celsius, and returns the result.
This example shows how a responder event gateway application can work, and illustrates how different event gateway types require different outgoing message formats:

```
<cfcomponent displayname="tempconverter" hint="Convert temperatures between
 Celsius and Fahrenheit">

<cffunction name="onIncomingMessage" output="no">
 <cfargument name="CFEvent" type="struct" required="yes">
 <!--- Standard error message giving the correct input format. --->
 <cfset var errormsg = "Please enter scale, integer where scale is F or C,
  for example:F, 32">

 <!--- Get the message. --->
 <cfset data=cfevent.DATA>
 <cfset message="#data.message#">
 <!--- Where did it come from? --->
 <cfset orig="#CFEvent.originatorID#">
```

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<!--- Process the input, generate a message with the new temperature. --->
<!--- Input format is: degrees, temperature. --->
<cfif listlen(message) eq 2>
 <cfif (listgetat(message,1) IS "F") OR
   (listgetat(message,1) IS "Fahrenheit") OR
   (listgetat(message,1) IS "C") OR
   (listgetat(message,1) IS "Celsius")>
  <cfset scale=listgetat(message,1)>
  <cfif isNumeric(listgetat(message,2))>
   <cfset temperature=listgetat(message,2)>
   <cfswitch expression="#scale#">
    <cfcase value="F, Fahrenheit">
     <cfset retmsg = temperature & " degrees Fahrenheit is "
      & (temperature-32.0) * (5.0/9.0) & " degrees Celsius">
    </cfcase>
    <cfcase value="C, Celsius">
     <cfset retmsg = temperature & " degrees Celsius is "
      &(temperature * 10.0/5.0) + 32 & " degrees Fahrenheit">
    </cfcase>
   </cfswitch>
  <cfelse>
   <cfset retmsg=errormsg>
  </cfif>
 <cfelse>
  <cfset retmsg=errormsg>
 </cfif>
<cfelse>
 <cfset retmsg=errormsg>
</cfif>

<!--- Fill the return value as required for the event gateway type. --->
<cfif arguments.CFEVENT.GatewayType is "Socket">
 <cfset retValue = structNew()>
 <cfset retValue.MESSAGE = retmsg>
 <cfset retValue.originatorID = orig>
<cfelseif (arguments.CFEVENT.GatewayType is "Sametime") OR
  (arguments.CFEVENT.GatewayType is "XMPP")>
 <cfset retValue = structNew()>
 <cfset retValue.MESSAGE = retmsg>
 <cfset retValue.BuddyID = arguments.CFEVENT.DATA.SENDER>
 <cfset retValue.originatorID = orig>
<cfelseif arguments.CFEVENT.GatewayType is "SMS">
 <cfset retValue = structNew()>
 <cfset retValue.command = "submit">
 <cfset retValue.sourceAddress = arguments.CFEVENT.gatewayid>
 <cfset retValue.destAddress = arguments.CFEVENT.originatorid>
 <cfset retValue.shortMessage = retmsg>
</cfif>

<!--- Send the return message back. --->
<cfreturn retValue>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

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

Sending a message using the SendGatewayMessage function

The `SendGatewayMessage` function has the following format:

```
SendGatewayMessage(gatewayID, messageStruct)
```

- The *gatewayID* parameter must be the gateway ID specified in the ColdFusion Administrator for the event gateway instance that sends the message.
- The *messageStruct* parameter is a structure whose contents depends on the requirements of the `outgoingMessage` method of the event gateway, and possibly the recipient application. For example, in addition to any message, the structure could include a destination identifier.
  The CFEvent instance passed to the event gateway contains these two parameters in the GatewayID and Data fields; the remaining fields are empty.
  The following example sends a message to a logging CFC, which logs information to a file. If the `SendGatewayMessage` function returns "OK", the example code displays a message. The code uses an instance of the asynchronous CFML event gateway named Asynch Logger. The props variable used in the *messageStruct* parameter has two entries, the destination file and the message to log.

```
<cfscript>
 status = "No";
 props = structNew();
 props.Message = "Replace me with a variable with data to log";
 status = SendGatewayMessage("Asynch Logger", props);
 if (status IS "OK") WriteOutput("Event Message ""#props.Message#"" has been
sent.");
</cfscript>
```

Note

To see the code for the CFC that logs the information, see Using the CFML event gateway for asynchronous CFCs.

Using the GatewayHelper object

The ColdFusion `GetGatewayHelper` function tells ColdFusion to create and initialize a Java GatewayHelper object that provides event gateway-specific helper methods and properties. To use this function, the event gateway must implement a GatewayHelper class. For example, an instant messaging event gateway could make buddy list management methods available in a GatewayHelper object.
The ColdFusion `GetGatewayHelper` function takes a single parameter, the ID of the event gateway instance that provides the helper, and returns a GatewayHelper Java object. The parameter value must be the gateway ID for the instance that is specified in the ColdFusion Administrator. If you do not want to hard-code an ID value in the application (for example, if your listener CFC can respond to multiple event gateway instances), get the gateway ID from the CFEvent structure of the first incoming message.
The CFML code accesses the GatewayHelper object's methods and properties using standard ColdFusion Java object access techniques (see Integrating JEE and Java Elements in CFML Applications). For example, if an event

© 2014 Adobe Systems Incorporated. All rights reserved.

gateway's GatewayHelper class includes an `addBuddy` method that takes a single String parameter, you could use the following code to get the ColdFusion XMPP or Sametime gateway GatewayHelper object and add a buddy to the buddies list:

```
<cfscript>
 myHelper = GetGatewayHelper(myGatewayID);
 status = myHelper.addBuddy("jsmith23", "Jim Smith", "support");
</cfscript>
```

Using the event gateway error log file

When a standard ColdFusion event gateway encounters an error that does not prevent the event gateway from continuing to process, it logs it to the eventgateway.log file in the ColdFusion logs directory. Other event gateways can also to log information in this file, or to other application-specific files in the logs directory.
The standard ColdFusion event gateways log errors in interaction with any messaging server, errors in messages sent by the ColdFusion application, and recoverable errors in event gateway operation. The event gateways also log informational status messages for significant normal events, including event gateway initialization and restarts.
ColdFusion event gateway messages in the eventgateway.log file normally have the following format:

```
gatewayType (gatewayID) message body
```

When you are developing an event gateway application, you can use the ColdFusion Log viewer to inspect the eventgateway.log file and filter the display by using the gateway type and possibly the gateway ID as keywords. By selecting different severity levels, you can get a good understanding of errors and possible inefficiencies in your application and event gateway operation.

© 2014 Adobe Systems Incorporated. All rights reserved.

Deploying event gateways and applications

To deploy an event gateway application in a ColdFusion server, install your listener CFC and configure a gateway instance that uses the CFC.

Deploy an event gateway application

1. Ensure that the ColdFusion Administrator is configured with the required event gateway type. If it is not, deploy the event gateway type (see Deploying an event gateway).
2. If the event gateway type requires a configuration file, ensure that a valid file exists in the gateway\config directory. Some event gateways could be designed to let multiple event gateway instances share a configuration file. Others could require a separate file for each event gateway instance.
3. Install the event gateway application listener CFC and any other application components. ColdFusion provides a *cf_root*\gateways\cfc directory as a convenient location for these CFCs, and includes a mapping in the ColdFusion Administrator page for that directory. However, ColdFusion does not require you to install the listener CFC in this directory.
4. Configure an event gateway instance on the Gateway Instances page of the Event Gateways section in the ColdFusion Administrator (see Configuring an event gateway instance)

© 2014 Adobe Systems Incorporated. All rights reserved.

Using the CFML event gateway for asynchronous CFCs

The ColdFusion CFML event gateway lets CFML code send a message to CFC methods asynchronously. This event gateway lets you initiate processing by a CFC method without waiting for it to complete or return a value. Possible uses for asynchronous CFCs that you access using this event gateway include the following:

- Reindexing a Solr collection with new information without delaying an application, for example, when a user uploads a new file
- Logging information, particularly if significant amount of data must be logged
- Running batch processes that could take a substantial amount of time to complete
  Because asynchronous CFCs run independently of a request, they do not provide feedback to the user. Save any results or error information to a file, data source, or other external resource.
  By default, ColdFusion delivers the message to a CFC method named onIncomingMessage. You can specify any method name, however, in the SendGatewayMessage method's *data* parameter.

CFML event gateway data structure

The structure that you use in the CFML `SendGatewayMessage` function can include two types of fields:

- The structure can include any number of fields with arbitrary contents for use in by the CFC.
- Several optional fields can configure how the gateway delivers the information to the CFC.
  The CFML gateway looks for the following optional fields, and, if they exist, uses them to determine how it delivers the message. Do not use these field names for data that you send to your CFC method.

Field	Use
cfcpath	Overrides the CFC path specified in the ColdFusion Administrator. This field lets you use a single gateway configuration in the ColdFusion Administrator multiple CFCs.
method	Sets the name of the method to invoke in the CFC. The default method is `onIncomingMessage`. This field lets you use a single gateway configuration in the ColdFusion Administrator for a CFC that has several methods.
originatorID	Sets the originatorID field of the CFEvent object that ColdFusion delivers to the CFC. The default value is CFMLGateway.
timeout	Sets the time-out, in seconds, during which the listener CFC must process the event request and return before ColdFusion gateway services terminates the request. The default value is the Timeout Request value set on the Server Settings page in the ColdFusion Administrator. Set this value if a request could validly take longer to process than the default timeout; for example, if the request involves a long processing time.

Using the CFML gateway

The following procedure describes how to use an asynchronous CFC that has a single, `onIncomingMessage` meth

© 2014 Adobe Systems Incorporated. All rights reserved.

od.

Use an asynchronous CFC

1. Create a CFC with an `onIncomingMessage` method. Place the CFC in an appropriate directory for your application. For example, you can place it in the *cf_root*\WEB-INF\cfusion\gateway\cfc directory on J2EE configurations, in the *cf_root*\gateway\cfc directory on server configurations, or in a subdirectory of these directories. ColdFusion is installed with mappings to these cfc gateway directories.The `onIncomingMessage` method must take a CFEvent structure that contains input information in its Data field, and processes the contents of the Data field as needed.
2. Use the Gateway Instances page in the ColdFusion Administrator to add an instance of the CFML event gateway type. Specify the following:
   - A unique Gateway ID.
   - The path to the CFC that you created in step 1.
   - The startup mode. Select Automatic startup mode to start the event gateway when ColdFusion starts up.
   - Do *not* specify a configuration file.
3. Start the event gateway instance.
4. Write CFML code that uses `SendGatewayMessage` functions to send messages in structures to the event gateway instance ID that you specified in step 2. The `SendGatewayMessage` function returns true if the gateway successfully queues the message in the ColdFusion Gateway Service; false, otherwise. It does not ensure that the CFC receives or processes the message.
5. Run your CFML application.

Example: logging messages

The following asynchronous CFML event gateway CFC uses the `cflog` tag to log a message to a file in the ColdFusion logs directory. The CFC takes a message with the following fields:

- file:The name of the file in which to place the message. The default value is defaultEventLog.
- type:The `cflog` type attribute to use. The default value is info.
- message:The message text.

```
<cfcomponent>
 <cffunction name="onIncomingMessage" output="no">
  <cfargument name="CFEvent" type="struct" required="yes">
  <cfscript>
   if (NOT IsDefined("CFEvent.Data.file")) {
     CFEvent.Data.file="defaultEventLog"; }
   if (NOT IsDefined("CFEvent.Data.type")) {
    CFEvent.Data.type="info"; }
  </cfscript>
  <cflog text="#CFEvent.Data.message#"
    file="#CFEvent.Data.file#"
   type="#CFEvent.Data.type#"
   thread="yes"
   date="yes"
   time="yes"
   application="yes">
 </cffunction>
</cfcomponent>
```

The following minimal CFML page tests the event gateway:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
Sending an event to the CFML event gateway that is registered in the
 ColdFusion Administrator as Asynch Logger.<br>
<cfscript>
 status = false;
 props = structNew();
 props.Message = "Replace me with a variable with data to log";
 status = SendGatewayMessage("Asynch Logger", props);
 if (status IS True) WriteOutput('Event Message "#props.Message#" has been sent.');
</cfscript>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

Using the example event gateways and gateway applications

ColdFusion provides several example event gateways and applications in the *cf_root*\WEB-INF\cfusion\gateway directory on J2EE configurations or the *cf_root*\gateway directory on server configurations. These gateways provide example code that you can examine or use in developing your gateways. They are intended as examples only, and are not complete, product-quality, implementations.

Example event gateways

The gateway\src\examples directory and its subdirectories include the sources for the example event gateways. Compiled versions are located in the gateway\lib\examples.jar file. This document briefly describes the event gateways and their functions. For more detailed information, see the code and comments in the files.

EmptyGateway

The EmptyGateway.java file contains an event gateway template that you can use as a skeleton for creating your own event gateway. For more information on this class, and on creating new event gateways, see Creating Custom Event Gateways

SocketGateway

The SocketGateway event gateway listens on a TCP/IP port. Therefore, you can use this gateway for applications that send and respond to messages using TCP/IP-based protocols such as Telnet, or for applications that send messages between sockets. For example, a simple gateway application that responds to messages from a Telnet terminal client without supporting the full Telnet protocol.

> **Note**
>
> The ColdFusion Administrator uses Socket as the gateway type name for the SocketGateway class.

The SocketGateway.java file defines two classes: SocketGateway, the event gateway, and SocketHelper, a GatewayHelper class. The Source file is located in the gateway\src\examples\socket directory.

- **SocketGateway**Listens on a TCP/IP port. This event gateway is multi-threaded and can handle multiple clients simultaneously. It can send outgoing messages to existing clients, but cannot establish a link itself.By default, the SocketGateway class listens on port 4445, but you can specify the port number in a configuration file. The file should contain a single line in the following format:

  ```
  port=portNumber
  ```

- **SocketHelper** A GatewayHelper class with the following methods: `getSocketIDs()` returns an array containing the socket IDs of all Java sockets that are open. The event gateway opens a socket for each remote client.`killSocket(String` *socketid*{{)}}removes the specified socket. Returns a Boolean success indicator.

DirectoryWatcherGateway

The DirectoryWatcherGateway event gateway sends events to the listener CFC when a file is created, deleted, or modified in a directory. The watcher runs in a thread that sleeps for an interval specified in the configuration file, and when the interval has passed, checks for changes since the last time it was awake. If it finds added, deleted, or

© 2014 Adobe Systems Incorporated. All rights reserved.

changed files, it sends a message to a listener CFC. You can configure separate CFCs for add, delete, and change events, or use a single CFC for all events. The source for this event gateway is located in the gateway/src/examples/watcher directory.

> **Note**
>
> The ColdFusion Administrator uses DirectoryWatcher as the gateway type name for the DirectoryWatcherGateway class.

Configuration file

This event gateway requires a configuration file, consisting of lines in the following format:

```
directory=C:/temp
```

> **Note**
>
> If you use backward slash characters () as directory separators in Windows the file paths, escape them by using double slashes, as in C:
> temp. You can use forward slashes ✅ as the directory separator on all operating systems, including Windows.

> **Note**
>
> When you specify filename extensions in the Directory Watcher configuration file, do not include the period, instead use a comma, for example, *`doc,txt`*.

The configuration file can have comment lines, preceded by a number sign (#). If you omit a property or comment it out, ColdFusion uses the default value. If you specify a property with no value, ColdFusion sets an empty property. The configuration file can define the following values:

Property	Req/Opt	Description
`directory`	Required	Path to the directory to watch.
`recurse`	Optional	Whether to check subdirectories. The default value is no.
`extensions`	Optional	Comma-delimited list of extensions to watch. The event gateway logs only changed files with these extensions. An asterisk (*) indicates all files. The default value is all files.
`interval`	Optional	Number of milliseconds between the times that the event gateway checks the directory. The default value is 60 seconds.

© 2014 Adobe Systems Incorporated. All rights reserved.

`addFunction`	Optional	Name of the function to call when a file is added. The default value is `onAdd`.
`changeFunction`	Optional	Name of the function to call when a file is changed. The default value is `onChange`.
`deleteFunction`	Optional	Name of the function to call when a file is deleted. The default value is `onDelete`.

An example configuration file is located in the gateway\config\directory-watcher.cfg file.

CFC methods

When the directory contents change, the event gateway calls one of the following CFC listener methods, unless you change the names in the configuration file:

- `onAdd`
- `onChange`
- `onDelete`

  The CFEvent.Data field sent to the listener methods includes the following fields:

Field	Description
TYPE	Event type, one of ADD, CHANGE, DELETE.
FILENAME	Absolute path from the system directory root to the file that was added, deleted, or changed.
LASTMODIFIED	The date and time that the file was created or modified. This field is not included if the file was deleted.

The event gateway supports multiple listener CFCs and sends the event messages to all listeners. The event gateway is one way; it watches for events and dispatches event information to a CFC, but it does not accept return values from the CFC or input from `SendGatewayMessage` functions.The directory watcher logs errors to the watcher.log file in the ColdFusion logs directory.

Example DirectoryWatcher application

The following code shows a simple directory watcher application. It enters a line in a log file each time a file is added, deleted, or changed in the directory specified in the configuration file. ColdFusion includes the date and time that a log entry is made. However, if the directory watcher monitors changes infrequently, for example once every minute or more, the time in the log entry could differ from the time a file was added or changed, so the information includes the time (but not date) for these actions.

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cfcomponent>
<cffunction name="onAdd" output="no">
 <cfargument name="CFEvent" type="struct" required="yes">
 <cfset data=CFEvent.data>
 <cflog file="MydirWatcher" application="No"
  text="ACTION: #data.type#;FILE: #data.filename#;
  TIME: #timeFormat(data.lastmodified)#">
</cffunction>

<cffunction name="onDelete" output="no">
<cfargument name="CFEvent" type="struct" required="yes">
<cfset data=CFEvent.data>
<cflog file="MydirWatcher" application="No"
 text=" ACTION: #data.type#;FILE: #data.filename#">
</cffunction>

<cffunction name="onChange" output="no">
<cfargument name="CFEvent" type="struct" required="yes">
<cfset data=CFEvent.data>
<cflog file="MydirWatcher" application="No"
  text=" ACTION: #data.type#;FILE: #data.filename#;
  TIME: #timeFormat(data.lastmodified)#">
</cffunction>
</cfcomponent>
```

JMSGateway

The JMSGateway class acts as a Java Messaging Service consumer or producer. The source for this event gateway is located in gateway/src/examples/JMS. The gateway requires a configuration file, which is in gateway/config/jmsgateway.cfg. For full documentation of the configuration options, See the configuration file. The ColdFusion Administrator lists the compiled gateway (which is included in the gateway\lib\examples.jar file) on the Gateway Types page.

> **Note**
>
> The ColdFusion Administrator uses JMS as the gateway type name for the JMSGateway class.

Using the JMS Gateway as a consumer

The JMSGateway class creates a subscriber to the topic specified in the configuration file. The gateway consumes the following types of messages:

- TextMessage
- BytesMessage containing raw UTF-8 text
  The gateway passes the contents of the message to the configured CFC in the event structure, as follows:

Field	Contents
data.id	Message correlation ID
data.msg	Text of the message

© 2014 Adobe Systems Incorporated. All rights reserved.

gatewayType	Gateway type: JMS
originatorID	Topic name from which the message was consumed

The listener CFC method must be named `onIncomingMessage`. If the CFC method does not send a message in response, it should return a structure containing a status field with a value of OK or EXCEPTION. (In this case, The gateway checks the return status field, but does not process these return values further.) To send a message, the CFC method must return a structure as documented in the following section.

Using the JMS Gateway as a producer

To send a JMS message, the return value of your CFC method or the second, *messageStruct*, parameter to the `SendGatewayMessage` function must be a structure with the following fields:

Field	Contents
status	Must be SEND.
topic	Name of the topic to publish the message to.
id	(Optional) The JMS correlation ID to associate with the message. The default is null.
message	Text of the message to publish.
asBytes	(Optional) How to publish the message: • If omitted, no, or false, send the message as text. • If any other value, send the message as byte-encoded UTF-8.

If you send the message in a `SendGatewayMessage` function, the function returns OK if the gateway sends the message, or EXCEPTION if it fails to send the message.

ActiveMQ JMS event gateway

Apache ActiveMQ is a message broker that implements JMS. The source for this event gateway is located in gateway/src/examples/ActiveMQ. For information about using the ActiveMQ JMS event gateway, see the gateway\docs\ActiveMQDeveloperGuide.pdf file.

Menu example application

ColdFusion is installed with a menu-based responder application. The menu application is written to work with any of the standard ColdFusion event gateways (SMS, XMPP, and Sametime) and with the Socket example event gateway, and ColdFusion is preconfigured with an instance of the application that uses SMS, as follows:

- The Gateway Instances page in the ColdFusion Administrator includes a gateway instance for this application that uses the SMS gateway type.
- The gateway/cfc/examples/menu directory and its subdirectories include the CFML for the application
- The gateway/config/sms-test.cfg file is configured to use this application with the SMS client (phone simulator), and short message service center (SMSC) server simulator that are provided with ColdFusion. The application presents users with a drill-down menu of tools that they can use, including a weather report,

© 2014 Adobe Systems Incorporated. All rights reserved.

stock information, status and configuration information, and language tools such as a dictionary.
The code for this application is relatively complex and is distributed among 13 files. The following brief description provides an overview of how it works. To get a full understanding of how the application works, see the source code.

- The top level, menu, directory contains two files: Application.cfm and main.cfc.
- The Application.cfm file consists of a single `cfapplication` tag that enables session management and names the application. Session variables maintain the current state information of the session, such as the active menu, and so on.
- The main.cfc file contains the master CFC; the event gateway configuration in ColdFusion Administrator uses it as the listener CFC. The main CFC file processes CFEvent structures from the event gateway. It does the following:
  1. Inspects the gatewayType field to determine the rest of the structure contents. This check is necessary because different event gateways place the message in fields with different names.
  2. If a Session.menu variable does not exist, initializes the menu system. To do so, it calls methods in two other CFCs: menu and menunode. These two CFCs contain the menu system code.
  3. Calls the session.menu.process method to process the user input. This method can dispatch a message to an individual application for processing, if appropriate.
- The apps directory contains several CFCs. Each file contains the code for a single application, such as the weather report or dictionary lookup (definition.cfc).

Use the menu application with the Socket event gateway

1. On the Gateway Settings page in the ColdFusion Administrator, click the Start SMS Test Server button.
2. On the Gateway Instances page in the ColdFusion Administrator, start the SMS Menu App - 5551212 event gateway by clicking the green play button (third button from the left in the Actions column). If the Status does not say Running after a few seconds, click Refresh to check that the server started.
3. In the *cf_root*\WEB-INF\cfusion\bin directory on J2EE configurations or the *cf_root*\bin directory on server configurations, run the SMSClient.bat file (on Windows) or SMSClient.sh file (on UNIX or Linux) to start the SMS phone simulator. The simulator is preconfigured by default to "call" the default SMS event gateway configuration.
4. Enter any character by typing or by using the mouse to click the simulator keypad, and press Enter on your keyboard or click Send on the simulator.
5. The menu application responds with the top-level menu. Enter L for language tools such as a dictionary and thesaurus, S to get stock quotes or weather forecasts, or C to get information about the server. Press Enter on your keyboard or click Send on the simulator.
6. The application displays a submenu. For example, if you select S in step 5, the options are Q for a stock quote, W for weather, or B to go back to the previous menu. Enter your selection.
7. The application requests information such as a Zip code for the weather, stock symbol for a price, word for the dictionary, and so on. Enter and send the required information (or enter B to go back to the menu).
8. The application gets and displays the requested information. Depending on the application, you could also be prompted to enter M to get more. Enter M (if more information is available), another term, or B to return to the previous menu.
9. Continue by entering menu items and detailed information requests.
10. To exit, select File > Exit from the menu bar.

#back to top

© 2014 Adobe Systems Incorporated. All rights reserved.

Using the Instant Messaging Event Gateways

You can develop an application that uses either of two instant message (IM) event gateway types provided with Adobe ColdFusion: an IBM Lotus Sametime gateway, and an Extensible Messaging and Presence Protocol (XMPP) gateway.

Before you use the IM event gateways, become familiar with ColdFusion event gateway principles and programming techniques (see Using Event Gateways).

© 2014 Adobe Systems Incorporated. All rights reserved.

About ColdFusion and instant messages

ColdFusion includes two instant messaging gateway types: one for messaging using the XMPP protocol, and one for IBM Lotus Instant Messaging (Sametime). These gateway types use identical interfaces for sending and receiving messages and for managing the IM presence information and infrastructure. This documentation, therefore, refers to IM gateways, and only describes the two types where differences exist.
The ColdFusion IM gateways act as IM clients and let you do the following:

- Send and receive instant messages.
- Send and respond to buddy or friend requests and manage buddy/friend information.
- Set and get status and other information.
- Receive and handle messages from the IM server.

About XMPP

XMPP (Extensible Messaging and Presence Protocol) is an open, XML-based protocol for instant messaging. It is the core protocol of the Jabber Instant Messaging and Presence technology that the Jabber Software Foundation develops. As of November 2004, four Internet Engineering Task Force (IETF) specifications (RFCs) defined XMPP, numbers 3920-3923. RFC 3920 covers the XMPP core, and 3921 covers instant messaging and presence. Numerous XMPP servers and clients are available. ColdFusion supports the IETF XMPP protocol.
The following websites provide additional information about the XMPP protocol:

- Jabber Software Foundation: www.jabber.org/. This site includes information on available XMPP servers and clients.
- IETF has copies of the Internet standards for XMPP: www.ietf.org/rfc.html.
- The xmpp.org website was under development as of December 2004; at that time it included several useful links, including links to relevant specifications: www.xmpp.org/.

About IBM Lotus Instant Messaging (Sametime)

IBM Lotus Instant Messaging, commonly referred to as Lotus Sametime, is the IBM product for real-time collaboration. For more information about this product, see www.lotus.com/sametime.

> **Note**
>
> In the Enterprise Edition, to use the Lotus Sametime event gateway, disable FIPS-140 Compliant Strong Cryptography by adding the following to the JVM arguments in the ColdFusion Administrator:

-Dcoldfusion.disablejsafe=false

About IM application development and deployment

The following information introduces the ColdFusion IM application development tools and process, and discuss IM messaging providers.

ColdFusion IM gateway classes

ColdFusion provides the following instant messaging gateway classes:

- **XMPPGateway** The class for the XMPP event gateway type
- **SAMETIMEGateway** The class for the IBM Lotus Instant Messaging event gateway You implement your IM application by configuring a gateway instance in ColdFusion Administrator that uses one of these gateway classes and creating a ColdFusion application that uses the gateway instance to communicate with an instant messaging server.

© 2014 Adobe Systems Incorporated. All rights reserved.

Application development and deployment process

The following is a typical process for developing and deploying an IM application:

1. Design your application.
2. Configure an IM event gateway instance to use an available XMPP or Lotus Sametime server.
3. Write your CFCs, CFM pages, and any other application elements.
4. Test your application using your XMPP or Lotus Sametime server and an appropriate client application.
5. Deploy the application (see Deploying event gateways and applications).

How the IM event gateway and provider interact

Each IM event gateway instance has a single instant messaging ID. Establish the ID and its related password on the IM server using server-specific tools, such as a standard instant messaging client. In ColdFusion, you set the ID, password, and other gateway-specific information in a gateway configuration file, and you create a gateway instance that uses this file.
When you start the gateway, it logs on to the IM server with the ID and password, and receives and sends the messages for the ID. The gateway sends incoming messages to a CFC, which you specify when you configure the gateway instance in the ColdFusion Administrator. The gateway passes outgoing messages from this CFC and from other CFML code to the IM server.
The IM event gateway also provides several helper methods for managing the gateway and its configuration information.

Incoming message handling

You write the following ColdFusion CFC methods to handle incoming messages and requests from the IM event gateway. These CFCs receive messages from the IM server and can respond to them by setting a return value.

CFC method	Message type
`onIncomingMessage`	Standard message from IM users.
`onAddBuddyRequest`	Requests from others to add the gateway ID to their buddy list.
`onAddBuddyResponse`	Responses from others to requests from your gateway to add them to your buddy lists. Also used by buddies to ask to be removed from your list.
`onBuddyStatus`	Presence status messages from other users.
`onIMServerMessage`	Error and status messages from the IM server.

For more information on these methods, see Handling incoming messages.

Outgoing message handling

Applications send outgoing instant messages using the CFML `SendGatewayMessage` method. Incoming message-handling CFC methods can also send messages, including responses to requests from others to add the ColdFusion gateway's ID to their buddy list. For more information on sending messages, see Sending outgoing messages.

IMGatewayHelper methods

© 2014 Adobe Systems Incorporated. All rights reserved.

The ColdFusion IM gateway provides the IMGatewayHelper class, a gateway helper that you can access by calling the CFML `GetGatewayHelper` function. The IMGatewayHelper class has methods that let you do the following:

- Get and set gateway configuration information and get gateway statistics
- Get and set the gateway online presence status
- Manage the gateway's buddy list
- Manage permissions for others to get information about the gateway status.For more information on using GatewayHelper methods, including lists of all the methods, see Using the GatewayHelper object.

© 2014 Adobe Systems Incorporated. All rights reserved.

Configuring an IM event gateway

You provide IM-specific configuration information to the IM event gateway in a configuration file. You specify the configuration file location when you configure the IM event gateway instance in the ColdFusion Administrator. ColdFusion provides sample XMPP and Lotus Sametime event gateway configuration files in the *cf_root*\WEB-INF\cfusion\gateway\config directory on J2EE configurations, and *cf_root*\gateway\config directory on server configurations. The configuration file can have the following information.

> **Note**
>
> The default value in the table is the value the gateway uses if the configuration file omits the property, not the value in the default configuration files.

Property	Default value	Description
userID	none	(Required) The IM user ID to use to connect to the IM server.
password	none	(Required) Password for the user.
secureprotocol	none	XMPP only.Required if you set `securerequirement` to true.The protocol to use for secure communications. The following values are valid: • TSL • SSL
securerequirement	false	XMPP only.Specifies whether the gateway must use secure communications. The following values are valid: • true • false If this value is true, specify a `secureprotocol` value, and connections succeed only if a secure connection is established.
serverip	XMPP: jabber.org Sametime: stdemo3.dfw.ibm.com	Address of XMPP or Lotus Sametime server to which to send messages. Can be a server name or IP address.
serverport	XMPP: 5222Sametime:1533	Port on the server to which to send the messages.If the XMPP `secure protocol` parameter is set to SSL, specify 5223.

© 2014 Adobe Systems Incorporated. All rights reserved.

retries	-1	Integer number of times to retry connecting to the IM server on gateway startup or if the Gateway gets disconnected.0 = do not to retry-1 = try forever
retryinterval	5	Real number of seconds to wait between connection attempts. The minimum is 1 second.
onIncomingMessageFunction	onIncomingMessage	Name of CFC method to call to handle an incoming message. If you specify the property without a value, such as "onIncomingMessageFunction=", the gateway does not send this event to a CFC.
onAddBuddyRequestFunction	onAddBuddyRequest	Name of CFC method to call to handle an incoming buddy request. If you specify the property without a value, the gateway does not send this event to a CFC.
onAddBuddyResponseFunction	onAddBuddyResponse	Name of CFC method to call to handle an incoming response to a buddy request sent by ColdFusion. If you specify the property without a value, the gateway does not send this event to a CFC.
onBuddyStatusFunction	onBuddyStatus	Name of CFC method to call to handle an incoming buddy status message, such as If you specify the property without a value, the gateway does not send this event to a CFC.
onIMServerMessageFunction	onIMServerMessage	Name of CFC method to call to handle an incoming message method. If you specify the property without a value, the gateway does not send this event to a CFC.

Note

If you do not have a CFC method to handle any of the event types, specify the corresponding property without a value. Use the following entry in the configuration file, for example, if you do not have a method to handle IMServerMessage events: onIMServerMessageFunction=

© 2014 Adobe Systems Incorporated. All rights reserved.

Handling incoming messages

The IM event gateway handles five types of messages, and your CFC must implement a listener method for each message type. The following table describes the message-handling CFC methods and the messages they handle. It lists the default CFC method names; however, you can change the names in the gateway configuration file.

CFC method	Description
onIncomingMessage	Standard message from an IM user. The application processes the message body appropriately; for example, it could display the message in an interface window. This method can return a response message to the sender.
onAddBuddyRequest	Request from another IM user to add your application's IM ID to their buddy list. The CFC must determine whether to accept or reject the request, or to take no action. An action is not always appropriate in cases where the request must be reviewed offline for approval and responses are sent at a later time.The CFC returns a message with the decision as a command value and optionally a text message specifying the reason. If you accept the request, the requestor automatically gets added to the list of IDs that can get status information for the gateway. If you specify no action, ColdFusion does not respond.
onAddBuddyResponse	Response from another IM user to a request from the gatewaybeing added to their buddy list. The response message is accept or decline. Your application can handle this response as appropriate; for example, to add or remove the ID from a list of message recipients. This method does not return a value.
onBuddyStatus	Message indicating a gateway buddy's status. Received when a buddy's status changes; for example, from OFFLINE to ONLINE. This method does not return a value.
onIMServerMessage	Status messages from the IM server, such as warning or error messages. The messages you can receive depend on the IM server that sends them. For information on the server messages, see the documentation for the IM server that your gateway instance uses. This method does not return a value.

For detailed information on each method, including examples of their use, see IM Gateway CFC incoming message methods in the *CFML Reference*. For an example that uses these functions, see Sample IM message handling application.

© 2014 Adobe Systems Incorporated. All rights reserved.

Sending outgoing messages

You use the `SendGatewayMessage` CFML function or the return value of a CFC listener method to send outgoing messages. The ColdFusion IM gateway accepts the following outgoing message commands:

Command	Description
submit	(Default) Sends a normal message to another IM user.
accept	Accepts an add buddy request. Adds the buddy to the list of IDs that get your presence information and sends an acceptance message to the buddy ID.
decline	Declines an add buddy request and sends a rejection message to the buddy ID.
noact	Tells the gateway to take no action. The gateway logs a message that indicates that it took no action, and contains the gateway type, gateway ID, and buddy ID.

The message structure that you return in the gateway listener CFC function or use as the second parameter in the CFML `SendGatewayMessage` function can have the following fields. The table lists the fields and the commands in which they are used, and describes each field's use.

Field	Commands	Description
buddyID	All	The destination user ID.
command	All	The command; if omitted, ColdFusion treats the message as a submit command.
message	submit	A text message to send to the destination user.
reason	accept, decline	A text description of the reason for the action or other message to send to the add buddy requestor.

In typical use, a ColdFusion application uses the accept, decline, and noact commands in the return value of the `onAddBuddyRequest` method, and uses the submit command (or no command, because submit is the default command) in `SendGatewayMessage` CFML functions and the return value of the `onIncomingMessage` CFC method.
The `SendGatewayMessage` CFML function can send any command, and can be used to send an accept or decline message. One possible use is in an application where someone must review all buddy requests before they are added. In this case, the `onAddBuddyRequest` CFC method could initially send a noact command in its return value, and save the request information in a database. Administrators could use a separate ColdFusion application to review the request information. This application could use the `SendGatewayMessage` function with an accept or

© 2014 Adobe Systems Incorporated. All rights reserved.

decline command to act on the request and inform the requestor.

The following example `onIncomingMessage` method of a listener CFC echoes incoming IM messages to the message originator:

```
<cffunction name="onIncomingMessage" output="no">
 <cfargument name="CFEvent" type="struct" required="yes">
 <cfset retValue.MESSAGE = "echoing: " & CFEvent.DATA.message>
 <cfset retValue.BuddyID = arguments.CFEVENT.DATA.SENDER>
 <cfreturn retValue>
</cffunction>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

Sample IM message handling application

The application described here consists of two CFCs: an employee phone directory lookup CFC that responds to an onIncomingMessage event, and a Gateway management CFC that responds to all other events. This example shows how an application can respond to events and send outgoing messages.
You can configure a gateway to use both CFCs by entering the paths to the two CFCs, separated by a comma, in the CFC Path field of the Add/Edit ColdFusion Event Gateways form on the Gateway Instances page in the ColdFusion Administrator.

Phone directory lookup CFC

The following CFC implements a simple employee phone directory lookup application. The user sends an instant message containing some part of the name to looked up (a space requests all names). The `onIncomingMessage` r esponse depends on the number matches.

- If there is no match, the `onIncomingMessage` function returns a message indicating that there are no matches.
- If there is one match, the function returns the name, department, and phone number.
- If there are up to ten matches, the function returns a list of the names preceded by a number that the user can enter to get the detailed information.
- If there are over ten matches, the function returns a list of only the first ten names. A more complex application can let the user get multiple lists of messages to provide access to all names.
- If the user enters a number, and previously got a multiple-match list, the application returns the information for the name that corresponds to the number.
  The following listing shows the CFC code:

```
<cfcomponent>
 <cffunction name="onIncomingMessage">
  <cfargument name="CFEvent" type="struct" required="YES">
  <!--- Remove any extra white space from the message. --->
  <cfset message =Trim(arguments.CFEvent.data.MESSAGE)>
  <!--- If the message is numeric, a previous search probably returned a
   list of names. Get the name to search for from the name list stored in
   the Session scope. --->
  <cfif isNumeric(message)>
   <cfscript>
    if (structKeyExists(session.users, val(message))) {
     message = session.users[val(message)];
    }
   </cfscript>
  </cfif>

  <!--- Search the database for the requested name. --->
  <cfquery name="employees" datasource="cfdocexamples">
   select FirstName, LastName, Department, Phone
   from Employees
   where 0 = 0
   <!--- A space indicates the user entered a first and last name. --->
   <cfif listlen(message, " ") eq 2>
    and FirstName like '#listFirst(message, " ")#%'
    and LastName like '#listlast(message, " ")#%'
   <!--- No space: the user entered a first or a last name. --->
   <cfelse>
    and (FirstName like '#listFirst(message, " ")#%'
    or LastName like '#listFirst(message, " ")#%')
   </cfif>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

```
  </cfquery>

  <!--- Generate andreturn the message.--->
  <cfscript>
   retrunVal = structNew();
   retrunVal.command = "submit";
   retrunVal.buddyID = arguments.CFEvent.data.SENDER;

   //No records were found.
   if (employees.recordCount eq 0) {
    retrunVal.message = "No records found for '#message#'";
   }
   //One record was found.
   else if (employees.recordCount eq 1) {
   // Whitespace in the message text results in bad formatting,
   // so the source cannot be indented.
    retrunVal.message = "Requested information:
#employees.firstName# #employees.lastName#
#employees.Department#
#employees.Phone#";
   }
   //Multiple possibilities were found.
   else if (employees.recordCount gt 1) {
    //If more than ten were found, return only the first ten.
    if (employees.recordCount gt 10)
    {
     retrunVal.message = "First 10 of #employees.recordCount# records";
    }else{
     retrunVal.message = "Records found: #employees.recordCount#";
    }
    // The session.users structure contains the found names.
    // The record key is a number that is also returned in front of the
    // name in the message.
    session.users = structNew();
    for(i=1; i lte min(10, employees.recordCount); i=i+1)
    {
     // These two lines are formatted to prevent extra white space.
     retrunVal.message = retrunVal.message & "
#i# - #employees.firstName[i]# #employees.lastName[i]#";
     // The following two lines must be a single line in the source
     session.users[i]="#employees.firstName[i]#
      #employees.lastName[i]#";
    }
   }
   return retrunVal;
  </cfscript>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

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

Status and request-handling CFC

The following CFC handles all IM events, except onIncomingMessage. It maintains an Application scope buddyStatus structure that contains information on the gateway buddies. This structure limits the interactions that are needed with the IM server to get buddy and status information. The application also logs significant events, such as requests to add buddies and error messages from the IM server. In particular, it does the following:

- The `onBuddyStatus` function updates the Application scope buddy status structure when the gateway gets an event message indicating that a buddy's status has changed.
- The `onAddBuddyRequest` function searches for the requested buddy's name in a data source. If it finds a single instance of the name, it adds the buddy and updates the status in the Application scope buddyStatus structure. If it doesn't find name, it declines the buddy request. If it finds multiple instances of the name, it tells the gateway to take no action. It also logs all actions.
- The `onAddBuddyResponse` function adds the buddy to the Application scope buddy status structure if the buddy request is accepted, and sets the current status. It logs all responses.
- The `onIMServerMessage` function logs all messages that it receives.This example uses the IM_ID column of the Employees database of the cfdocexamples database that is included with ColdFusion. The entries in this column assume that you use an XMPP server "company." To run this exampleconfigure an XMPP server with this name and with clients with names in this database, or change the database entries to match IM server clients. Also, configure a gateway instance in the ColdFusion Administrator that uses this server. The following listing shows the CFC code:

```
<cfcomponent>

<cffunction name="onBuddyStatus">
 <cfargument name="CFEvent" type="struct" required="YES">
 <cflock scope="APPLICATION" timeout="10" type="EXCLUSIVE">
  <cfscript>
  // Create the status structures if they dont exist.
   if (NOT StructKeyExists(Application, "buddyStatus")) {
    Application.buddyStatus=StructNew();
   }
   if (NOT StructKeyExists(Application.buddyStatus, CFEvent.Data.BUDDYNAME)) {
    Application.buddyStatus[#CFEvent.Data.BUDDYNAME#]=StructNew();
   }
   // Save the buddy status and timestamp.

Application.buddyStatus[#CFEvent.Data.BUDDYNAME#].status=CFEvent.Data.BUDDYSTATUS;

Application.buddyStatus[#CFEvent.Data.BUDDYNAME#].timeStamp=CFEvent.Data.TIMESTAMP;
  </cfscript>
 </cflock>
</cffunction>

<cffunction name="onAddBuddyRequest">
 <cfargument name="CFEvent" type="struct" required="YES">
 <cfquery name="buddysearch" datasource="cfdocexamples">
  select IM_ID
  from Employees
  where IM_ID = '#CFEvent.Data.SENDER#'
 </cfquery>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

```
  <cflock scope="APPLICATION" timeout="10" type="EXCLUSIVE">
   <cfscript>
    // If the name is in the DB once, accept; if it is missing, decline.
    // If it is in the DB multiple times, take no action.
    if (buddysearch.RecordCount IS 0) {
     action="decline";
     reason="Invalid ID";
    }
    else if (buddysearch.RecordCount IS 1) {
     action="accept";
     reason="Valid ID";
     //Add the buddy to the buddy status structure only if accepted.
     if (NOT StructKeyExists(Application,
       "buddyStatus")) {
      Application.buddyStatus=StructNew();
     }
     if (NOT StructKeyExists(Application.buddyStatus,
       CFEvent.Data.SENDER)) {
      Application.buddyStatus[#CFEvent.Data.SENDER#]=StructNew();
     }
     Application.buddyStatus[#CFEvent.Data.SENDER#].status=
      "Accepted Buddy Request";
     Application.buddyStatus[#CFEvent.Data.SENDER#].timeStamp=
      CFEvent.Data.TIMESTAMP;
     Application.buddyStatus[#CFEvent.Data.SENDER#].message=
      CFEvent.Data.MESSAGE;
    }
    else {
     action="noact";
     reason="Duplicate ID";
    }
   </cfscript>
  </cflock>
  <!--- Log the request and decision information. --->
  <cflog file="#CFEvent.GatewayID#Status"
    text="onAddBuddyRequest; SENDER: #CFEvent.Data.SENDER# MESSAGE:
#CFEvent.Data.MESSAGE# TIMESTAMP: #CFEvent.Data.TIMESTAMP# ACTION: #action#">
  <!--- Return the action decision. --->
  <cfset retValue = structNew()>
  <cfset retValue.command = action>
  <cfset retValue.BuddyID = CFEvent.DATA.SENDER>
  <cfset retValue.Reason = reason>
  <cfreturn retValue>
</cffunction>


<cffunction name="onAddBuddyResponse">
  <cfargument name="CFEvent" type="struct" required="YES">
  <cflock scope="APPLICATION" timeout="10" type="EXCLUSIVE">
   <cfscript>
    //Do the following only if the buddy accepted the request.
    if (NOT StructKeyExists(Application, "buddyStatus")) {
     Application.buddyStatus=StructNew();
     }
    if (#CFEVENT.Data.MESSAGE# IS "accept") {
     //Create a new entry in the buddyStatus record for the buddy.
     if (NOT StructKeyExists(Application.buddyStatus,
       CFEvent.Data.SENDER)) {
      Application.buddyStatus[#CFEvent.Data.SENDER#]=StructNew();
```

© 2014 Adobe Systems Incorporated. All rights reserved.

```
      }
      //Set the buddy status information to indicate buddy was added.
      Application.buddyStatus[#CFEvent.Data.SENDER#].status=
       "Buddy accepted us";
      Application.buddyStatus[#CFEvent.Data.SENDER#].timeStamp=
       CFEvent.Data.TIMESTAMP;
      Application.buddyStatus[#CFEvent.Data.SENDER#].message=
       CFEvent.Data.MESSAGE;
     }
    </cfscript>
   </cflock>
   <!--- Log the information for all responses. --->
   <cflog file="#CFEvent.GatewayID#Status"
    text="onAddBuddyResponse; BUDDY: #CFEvent.Data.SENDER# RESPONSE:
  #CFEvent.Data.MESSAGE# TIMESTAMP: #CFEvent.Data.TIMESTAMP#">
  </cffunction>


  <cffunction name="onIMServerMessage">
   <!--- This function just logs the message. --->
   <cfargument name="CFEvent" type="struct" required="YES">
   <cflog file="#CFEvent.GatewayID#Status"
    text="onIMServerMEssage; SENDER: #CFEvent.OriginatorID# MESSAGE:
  #CFEvent.Data.MESSAGE# TIMESTAMP: #CFEvent.Data.TIMESTAMP#">
```

© 2014 Adobe Systems Incorporated. All rights reserved.

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

© 2014 Adobe Systems Incorporated. All rights reserved.

Using the GatewayHelper object

The CFML `GetGatewayHelper` function returns a GatewayHelper object with several methods that manage your gateway and buddy list. The GatewayHelper methods let you do the following:

- Get and set gateway configuration information and get gateway statistics.
- Get and set the gateway online status.
- Manage the gateway's buddy list
- Manage permissions for others to get information about the gateway status.
  The following sections briefly describe the class methods. For detailed information about each method, see I M Gateway GatewayHelper class methods in the *CFML Reference*.

Gateway configuration information and statistics methods

The following table describes the methods that you can use to get and set configuration information and get gateway statistics:

Method	Description
getName	Returns the gateway's user name.
getNickName	Returns the gateway's display name (nickname).
getProtocolName	Returns the name of the instant messaging protocol (JABBER for XMPP, or SAMETIME).
numberOfMessagesReceived	Returns the number of messages received by the gateway since it was started.
numberOfMessagesSent	Returns the number of messages sent by the gateway since it was started.
setNickName	Sets the gateway's display name (nickname).

Gateway online status methods

The following table describes the methods that you can use to get and set the gateway's online availability status (presence information):

Method	Description
getCustomAwayMessage	Returns the gateway's custom away message if the `setStatus` method set it.
getStatusAsString	Returns the online status of the gateway.
getStatusTimeStamp	Returns the date/time that the gateway changed its online status.

© 2014 Adobe Systems Incorporated. All rights reserved.

isOnline	Returns True if the gateway is connected to the IM server; otherwise, returns false.
setStatus	Changes the gateway's online status; for example, to away or idle.

Gateway buddy management methods

The following table describes the methods that you can use to manage the gateway's buddy list:

Method	Description
addBuddy	Adds a buddy to the gateway's buddy list and tells the IM server to send the gateway messages with the buddy's online state.
getBuddyInfo	Gets information about the specified user from the buddy list, deny list, and permit list.
getBuddyList	Returns the gateway's buddy list.
removeBuddy	Removes the specified user name from the gateway's buddy list and tells the IM server to stop sending the gateway messages with the user's online state.

Gateway permission management methods

The IM gateways can manage the information that other users can get about the gateway's online status.

> **Note**
>
> XMPP permission management is included in the XMPP 1.0 specification, but several XMPP servers that were available at the time of the ColdFusion release do not support permission management.

The following table describes the gateway permission management methods:

Method	Description
addDeny	Tells the IM server to add the specified user to the gateway's deny list. If the `permitMode` is DENY_SOME, these users cannot receive messages on the gateway's state.
addPermit	Tells the IM server to add the specified user to the server's permit list. If the `permitMode` is PERMIT_SOME, these users receive messages on the gateway's state.

© 2014 Adobe Systems Incorporated. All rights reserved.

getDenyList	Returns the list of users that the server has been told not to send state information to.
getPermitList	Returns the list of users that the server has been told to send state information to.
getPermitMode	Gets the gateway's permit mode from the IM server. The permit mode determines whether all users can get the gateway's online state information, or whether the server uses a permit list or a deny list to control which users get state information.
removeDeny	Removes the user from the gateway's deny list.
removePermit	Removes the user from the gateway's permit list.
setPermitMode	Sets the gateway's permit mode on the IM server.

GatewayHelper example

This example lets you use the XMPP or SameTime GatewayHelper class to get and set status and other information, including managing buddy lists and view permissions lists.

```
<cfapplication name="gateway_tool" sessionmanagement="yes">

<!--- Set the gateway buddy name to default values.--->
<cfparam name="session.gwid" default="XMPP Buddy Manager">
<cfparam name="session.buddyid" default="hlichtin2@mousemail">

<!--- Reset gateway and buddy ID if form was submitted. --->
<cfif isdefined("form.submitbuddy")>
 <cfset session.buddyid=form.buddyid>
 <cfset session.gwid=form.gwid>
</cfif>

<!--- Display the current gateway and buddy ID. --->
<h3>Using the GatewayHelper</h3>
<!--- Form to display and reset gateway and Buddy ID. --->
<cfform action="#cgi.script_name#" method="post" name="changeIDs">
 Current buddy ID: <cfinput type="text" name="buddyid"
value="#session.buddyid#"><br>
 Current gateway ID: <cfinput type="text" name="gwid" value="#session.gwid#"><br>
 <cfinput name="submitbuddy" value="Change gateway/buddy" type="submit">
</cfform>

<!--- When a buddy is set, display the links and forms to get and set
  information, and so on, Where form input is required, the form uses a GET method
  so a url.cmd variable represents each selection. --->

<cfoutput>
<h3>Select one of the following to get or set.</h3>
<ul>
 <li><a href="#cgi.script_name#?cmd=buddyinfo">buddyinfo</a>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

```
 <li>LIST: <a href="#cgi.script_name#?cmd=buddylist">buddylist</a> |
  <a href="#cgi.script_name#?cmd=permitlist">permitlist</a> |
  <a href="#cgi.script_name#?cmd=denylist">denylist</a>
 <li>ADD: <a href="#cgi.script_name#?cmd=addbuddy">addbuddy</a> |
  <a href="#cgi.script_name#?cmd=addpermit">addpermit</a> |
  <a href="#cgi.script_name#?cmd=adddeny">adddeny</a>
 <li>REMOVE: <a href="#cgi.script_name#?cmd=removebuddy">removebuddy</a> |
  <a href="#cgi.script_name#?cmd=removepermit">removepermit</a> |
   <a href="#cgi.script_name#?cmd=removedeny">removedeny</a>
 <!--- NOTE: This list does not include OFFLINE because the gateway resets itself to
online. --->
 <li>setStatus (XMPP):
  <cfloop list="ONLINE,AWAY,DND,NA,FREE_TO_CHAT" index="e">
   <a href="#cgi.script_name#?cmd=setstatus&status=#e#">#e#</a> |
  </cfloop>
 <li>setStatus (Sametime):
  <cfloop list="ONLINE,AWAY,DND,IDLE" index="e">
   <a href="#cgi.script_name#?cmd=setstatus&status=#e#">#e#</a> |
  </cfloop>
 <li>
  <form action="#cgi.script_name#" method="get">
   setStatus with CustomAwayMessage:
   <input type="hidden" name="cmd" value="setstatus2">
   <select name="status">
    <cfloop
list="ONLINE,OFFLINE,AWAY,DND,IDLE,INVISIBLE,NA,OCCUPIED,FREE_TO_CHAT,ONPHONE,ATLUNC
H,BUSY,NOT_AT_HOME,NOT_AT_DESK,NOT_IN_OFFICE,ON_VACATION,STEPPED_OUT,CUSTOM_AWAY"
index="e">
     <option value="#e#">#e#</option>
    </cfloop>
   </select>
   <input type="text" name="custommsg" value="(custom away massage)" size="30"/>
   <input type="submit"/>
  </form>
 <li>
  <form action="#cgi.script_name#" method="get">
   setNickName:
   <input type="hidden" name="cmd" value="setnickname">
   <input type="text" name="nickname" value="(enter nickname)">
   <input type="submit">
  </form>
--->
 <li>INFO: <a href="#cgi.script_name#?cmd=getname">getname</a> |
  <a href="#cgi.script_name#?cmd=getnickname">getnickname</a> |
  <a href="#cgi.script_name#?cmd=getcustomawaymessage">getcustomawaymessage</a> |
  <a href="#cgi.script_name#?cmd=getprotocolname">getprotocolname</a> |
  <a href="#cgi.script_name#?cmd=getstatusasstring">getstatusasstring</a> |
  <a href="#cgi.script_name#?cmd=isonline">isonline</a>
 <li>MESSAGE COUNT:
  <a
href="#cgi.script_name#?cmd=numberofmessagesreceived">numberofmessagesreceived</a> |

  <a href="#cgi.script_name#?cmd=numberofmessagessent">numberofmessagessent</a>
 <li>RUNNING TIME: <a
href="#cgi.script_name#?cmd=getsignontimestamp">getsignontimestamp</a> |
  <a href="#cgi.script_name#?cmd=getstatustimestamp">getstatustimestamp</a>
 <li>setPermitMode:
  <cfloop
list="PERMIT_ALL,DENY_ALL,PERMIT_SOME,DENY_SOME,IGNORE_IN_LIST,IGNORE_NOT_IN_LIST"
```

© 2014 Adobe Systems Incorporated. All rights reserved.

```
    index="e"><a href="#cgi.script_name#?cmd=setpermitmode&mode=#e#">#e#</a> |
   </cfloop> <span class="note">doesn't work for XMPP</span>
  <li><a href="#cgi.script_name#?cmd=getpermitmode">getpermitmode</a>
  <li>setPlainTextMode:
   <cfloop list="PLAIN_TEXT,RICH_TEXT" index="e">
    <a href="#cgi.script_name#?cmd=setplaintextmode&mode=#e#">#e#</a> |
   </cfloop>
  <li><a href="#cgi.script_name#?cmd=getplaintextmode">getplaintextmode</a>
 </ul>
 </cfoutput>

 <!--- The url.cmd value exists if one of the previous links or forms has been
 submitted, and identifies the type of request. --->
 <cfoutput>
 <cfif isdefined("url.cmd")>
  <!--- Get the GatewayHelper for the gateway. --->
  <cfset helper = getGatewayHelper(session.gwid)>
  <!--- Get the buddy list if the list or full buddy information was requested. --->
  <cfswitch expression="#LCase(url.cmd)#">
   <cfcase value="buddylist,buddyinfo">
    <cfset ret=helper.getBuddyList()>
   </cfcase>
   <cfcase value="denylist">
    <cfset ret=helper.getDenyList()>
   </cfcase>
   <cfcase value="permitlist">
    <cfset ret=helper.getPermitList()>
   </cfcase>
   <cfcase value="addbuddy">
    <cfset ret=helper.addBuddy("#session.buddyid#",
     "#session.buddyid#", "")>
   </cfcase>
   <cfcase value="addpermit">
    <cfset ret=helper.addPermit("#session.buddyid#",
     "#session.buddyid#", "")>
   </cfcase>
   <cfcase value="adddeny">
    <cfset ret=helper.addDeny("#session.buddyid#",
     "#session.buddyid#", "")>
   </cfcase>
   <cfcase value="removebuddy">
    <cfset ret=helper.removeBuddy("#session.buddyid#", "")>
   </cfcase>
   <cfcase value="removepermit">
    <cfset ret=helper.removePermit("#session.buddyid#", "")>
   </cfcase>
   <cfcase value="removedeny">
    <cfset ret=helper.removeDeny("#session.buddyid#", "")>
   </cfcase>
   <cfcase value="setstatus">
    <cfset ret=helper.setStatus(url.status, "")>
   </cfcase>
   <cfcase value="setstatus2">
    <cfset ret=helper.setStatus(url.status, url.custommsg)>
   </cfcase>
   <cfcase value="getcustomawaymessage">
    <cfset ret=helper.getCustomAwayMessage()>
   </cfcase>
   <cfcase value="getname">
```

© 2014 Adobe Systems Incorporated. All rights reserved.

```
    <cfset ret=helper.getName()>
   </cfcase>
   <cfcase value="getnickname">
    <cfset ret=helper.getNickname()>
   </cfcase>
   <cfcase value="getprotocolname">
    <cfset ret=helper.getProtocolName()>
   </cfcase>
   <cfcase value="getsignontimestamp">
    <cfset ret=helper.getSignOnTimeStamp()>
   </cfcase>
   <cfcase value="getstatusasstring">
    <cfset ret=helper.getStatusAsString()>
   </cfcase>
   <cfcase value="getstatustimestamp">
    <cfset ret=helper.getStatusTimeStamp()>
   </cfcase>
   <cfcase value="isonline">
    <cfset ret=helper.isOnline()>
   </cfcase>
   <cfcase value="numberofmessagesreceived">
    <cfset ret=helper.numberOfMessagesReceived()>
   </cfcase>
   <cfcase value="numberofmessagessent">
    <cfset ret=helper.numberOfMessagesSent()>
   </cfcase>
   <cfcase value="setnickname">
    <cfset ret=helper.setNickName(url.nickname)>
   </cfcase>
   <cfcase value="setpermitmode">
    <cfset ret=helper.setPermitMode(url.mode)>
   </cfcase>
   <cfcase value="getpermitmode">
    <cfset ret=helper.getPermitMode()>
   </cfcase>
   <cfcase value="setplaintextmode">
    <cfset ret=helper.setPlainTextMode(url.mode)>
   </cfcase>
   <cfcase value="getplaintextmode">
    <cfset ret=helper.getPlainTextMode()>
   </cfcase>
   <cfdefaultcase>
    <cfset ret[1]="Error; Invalid command. You shouldn't get this.">
   </cfdefaultcase>
  </cfswitch>
  <br>
  <!--- Display the results returned by the called GatewayHelper method. --->
  <strong>#url.cmd#</strong><br>
  <cfdump var="#ret#">
  <br>
  <!--- If buddy information was requested, loop through buddy list to get
   information for each buddy and display it. --->
  <cfif comparenocase(url.cmd, "buddyinfo") is 0 and arraylen(ret) gt 0>
  <b>Buddy info for all buddies</b><br>
   <cfloop index="i" from="1" to="#arraylen(ret)#">
    <cfdump var="#helper.getBuddyInfo(ret[i])#" label="#ret[i]#"></cfloop>
  </cfif>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

```
</cfif>
</cfoutput>
```

#back to top

© 2014 Adobe Systems Incorporated. All rights reserved.

Using the SMS Event Gateway

You can develop an application that uses the short message service (SMS) event gateway type provided with Adobe ColdFusion. ColdFusion provides tools for developing SMS applications.
Before you use the SMS event gateway, become familiar with ColdFusion event gateway principals and programming techniques (see Using Event Gateways). Although not required, a basic knowledge of SMS is helpful.

© 2014 Adobe Systems Incorporated. All rights reserved.

About SMS and ColdFusion

Short Message Service (SMS) is a system designed for sending short, often text, messages to and from wireless devices, such as mobile phones or pagers. SMS is widely used in Europe and Asia and is becoming increasingly popular in the United States and elsewhere. Some uses for SMS include the following:

- Performing banking transactions
- Sending authentication codes, for example, to be used to access web resources
- Voting, such as popularity voting for reality television shows
- Initiating an action (such as a server restart) and getting a response
- Notifying users of events such as package shipments or restaurant table availability, or providing stock or weather alerts
- Sending person-to-person text messages
- Presenting interactive text-based menus on a mobile phone
- Providing cellular phone updates, such as direct download of logos
- Providing telematics and mobile or remote wireless device applications, such as soda machines, vehicle tracking, smart gas pumps, and so on

SMS protocol features include, but are not limited to, the following:

- Authentication verification is built in.
- Communications can be secure.
- Store and forward communication is performed in near real time.
- Communications can be two-way and session-aware.
- Mobile devices such as mobile phones already include support; you do not install software on the client.

About SMS

The following discussion simplifies SMS technology and describes only a typical use with a ColdFusion application. For a more complete discussion of SMS, see the publicly available literature, including the several books that discuss SMS.

In a ColdFusion SMS application, a mobile device such as a mobile phone communicates (via intermediate steps) with a message center, such as a short message service center (SMSC). For example, a mobile phone user calls a telephone number that the SMS provider has associated with your account; the SMSC gets the messages that are sent to this number. The SMSC can store and forward messages. A ColdFusion application can initiate messages to wireless devices, or it can respond to incoming messages from the devices.

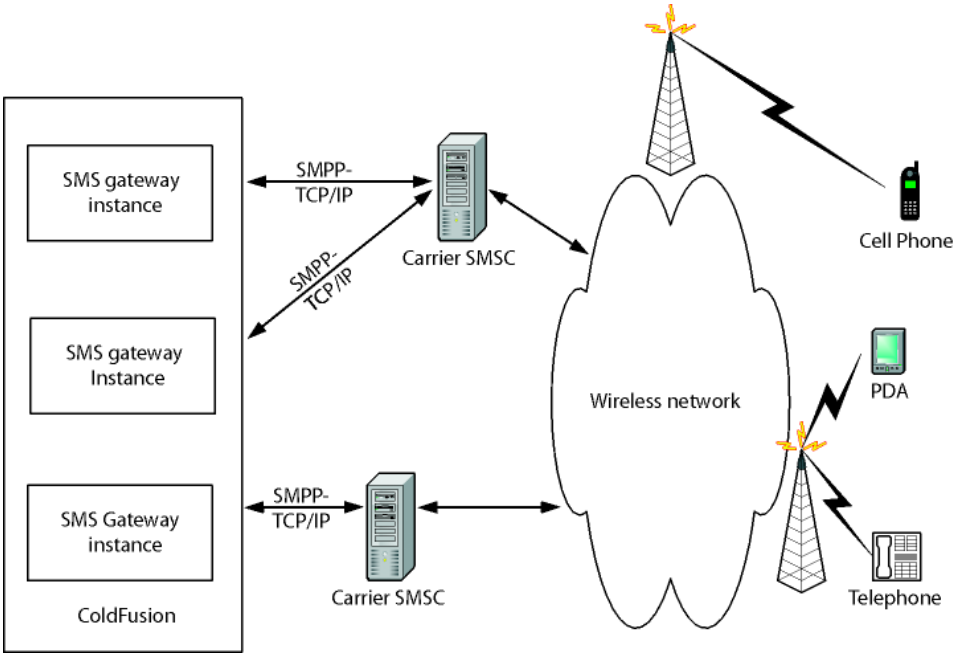
The SMSC communicates with a ColdFusion SMS event gateway using short message peer-to-peer protocol (SMPP) over TCP/IP. Information is transferred by exchanging Protocol Data Units (PDUs) with structures that depend on the type of transaction, such as a normal message submission, a binary data submission, or a message intended for multiple recipients.

Because the SMSC is a store-and-forward server, it can hold messages that cannot be immediately delivered and try to deliver them when the receiving device is available. The SMSC provider configures the time that a message is held on the server for delivery. For example, AT&T Wireless saves messages for 72 hours; after that time, any undelivered messages are deleted. Your messages can request a different time-out (by specifying a `ValidityPeriod` field). The message can also use a `registeredDelivery` field to tell the SMSC to inform you about whether and when the message is delivered.

SMS communication can be secure. Voice and data communications, including SMS message traffic between the SMSC and the mobile device is encrypted as part of the GSM standard. The SMSC authenticates the mobile user's identity before the encrypted communication session begins. Secure the communications between ColdFusion and the SMSC. Typically, you use a secure hardware or software VPN connection around the SMPP connection.

The following image shows the SMS path between mobile devices and ColdFusion gateways:

© 2014 Adobe Systems Incorporated. All rights reserved.



Using the SMS event gateway, ColdFusion establishes a two-way (transceiver) connection to the SMSC of the telecommunications carrier or other SMPP account provider. The SMPP provider assigns an address for your account, and you associate an event gateway instance with the address. Addresses are normally telephone numbers, but carriers often support "short code" addresses that do not require a full 10-digit number. You also configure the gateway instance to communicate with the provider's specified TCP/IP address using a system ID and a password.

> **Note**
>
> The ColdFusion SMS event gateway conforms to the SMPP 3.4 specification, which you can download from the SMS Forum at www.smsforum.net.

A ColdFusion application can initiate and send messages to SMS-enabled devices by specifying the destination mobile device telephone number, or mobile devices can send messages to a ColdFusion listener CFC by using the gateway instance's telephone number. Incoming messages include the sender's number, so listener CFCs can respond to messages sent by mobile devices.

About SMS application development and deployment

To develop an SMS gateway application, you use the ColdFusion SMS application development tools and process to interact with SMS messaging providers.

ColdFusion SMS application tools

ColdFusion provides the following tools for developing SMS applications:

- **SMSGateway** The class for the SMS event gateway type
- **SMS test server** A lightweight SMSC simulator
- **SMS client simulator** A graphical interface for sending and receiving SMS messages with the SMS test server You implement your SMS application by creating a ColdFusion application that uses an instance of the SMSGateway class to communicate with one or more SMSCs. You can use the SMS testing server and client simulator to test your application without requiring an outside SMS service provider.

Application development and deployment process

© 2014 Adobe Systems Incorporated. All rights reserved.

The following is a typical process for developing and deploying an SMS application:

1. Design your application.
2. Configure an SMS event gateway instance to use the ColdFusion SMS test server.
3. Write your ColdFusion CFCs, CFM pages, and any other application elements.
4. Test your application using the test server and client simulator.
5. Establish an SMPP account with a telecommunications provider.
6. Reconfigure your event gateway, or create an event gateway instance, to use your telecommunications providers SMSC. Configure the gateway using the information supplied by your provider.
7. Test your application using the telecommunications providers SMSC and target mobile devices.
8. Make the application publicly available.

About SMS providers

Before you can deploy an SMS application, establish an account with a provider that supports SMPP 3.4 over TCP/IP. Two kinds of providers exist:

- Telecommunications carriers such as nation-wide cellular phone providers
- Third-party SMPP aggregators
  The type of provider and specific provider you use depend on your needs and provider capabilities and price structures. Less expensive providers could have slower response times. Telecommunications carriers could be more expensive but might provide more throughput and faster SMPP response times.

How the SMS event gateway and provider SMSC interact

This following information provides a brief overview of the interactions between the ColdFusion SMS event gateway and the SMPP provider's SMSC. It is designed to help you to understand the basics of SMPP interactions, and defines the terms necessary to for you to understand gateway configuration and message handling. For more details, see the SMPP specification, which is available at www.smsforum.net/.
A typical interaction between an SMSC and a ColdFusion SMS event gateway instance consists of messages, or PDUs sent between the two entities, such as a mobile device and a ColdFusion event gateway instance (and therefore, and event gateway application).

Gateway binding

The event gateway must *bind* to the SMSC before they can communicate. The SMS event gateway instance initiates a binding by sending a bind_transceiver PDU to the SMSC, which includes the gateway's ID and password. If the initial bind request fails, the gateway retries the bind at the rate specified by the gateway configuration file retry-interval value until either the bind is successful or the gateway reaches the maximum number of retries, specified by the retries configuration value. If the bind operation fails, ColdFusion logs an error to the eventgateway.log file, and you restart the gateway instance in the ColdFusion Administrator to establish the connection.

> **Note**
>
> Some SMSCs can send a prohibited status in response to a bind request. If the gateway receives such a status response, it sets the retry interval to one minute and the maximum number of retries to 15. The SMS gateway detects SMPP 5.0-compliant and AT&T prohibited status responses.

When the SMSC accepts the bind request, it returns a bind_transceiver_resp PDU. The binding remains in effect until the gateway instance shuts down and sends an unbind PDU to the SMSC. Because the gateway binds as a transceiver, it can initiate messages to the SMSC, and the SMSC can send messages to it.

Incoming PDU handling

© 2014 Adobe Systems Incorporated. All rights reserved.

If the ColdFusion SMS event gateway gets an Unbind PDU from the SMSC, it sends an unbind_resp PDU to the SMSC, does a restart, and attempts to rebind to the SMSC.
When the event gateway receives an EnquireLink or any other request PDU from the SMSC, it sends a default response to the SMSC.
The gateway receives incoming messages from the SMSC in deliver _sm PDUs; it does not handle data_sm PDUs. Deliver_sm PDUs can contain user- or application-generated messages, or disposition responses for messages that the gateway has sent. The gateway extracts the short message field and source and destination addresses from the PDU, places them in a CFEvent object, and sends the object to ColdFusion event gateway services for delivery to the listener CFC. For information on how the CFML application must handle these incoming messages, see Handling incoming messages.

Outgoing message handling

The gateway supports three types of outgoing messages from ColdFusion applications. The CFML `sendGatewayMessage` function or a listener CFC method `cfreturn` tag can specify the following commands:

- **submit** Sends a submit_sm PDU with the message contents to the SMSC. This PDU sends a message to a single destination.
- **submitMulti** Sends a submit_multi PDU with the message contents to the SMSC. This PDU sends a message to multiple destinations.
- **data** Sends a data_sm PDU with the message contents to the SMSC. This command is an alternative to the submit command, and interactive applications such as those provided via a wireless application protocol (WAP) framework typically use it. The SMS gateway lets you control the contents of all of the fields of these PDUs. For more information on the individual commands, see Sending outgoing messages.
  When you send a message, if the SMSC responds with a status that indicates that the message was rejected or not sent, ColdFusion logs information about the failure in the eventgateway.log file. If the SMSC indicates that the service type is not available (SMPP v5 ESME_RSERTYPUNAVAIL status or AT&T Serviced denied status), and the gateway configuration file transient-retry value is set to yes, the gateway also tries to resend the message.

Outgoing message synchronization and notification

The gateway and SMSC communicate asynchronously: the gateway does not wait for a response from the SMSC for one message before it sends another message. However, you can configure your gateway instance so that the CFML `sendGatewayMessage` function behaves asynchronously or synchronously.

- In *asynchronous mode*, the function returns when the message is queued in ColdFusion gateway services.
- In *synchronous mode*, the function waits until the SMSC receives the message and returns a message ID, or an error occurs.
  For more information on configuring message synchronization and sending messages synchronously, see *Controlling SMS message sending and response in* Sending outgoing messages-event gateway.

© 2014 Adobe Systems Incorporated. All rights reserved.

Configuring an SMS event gateway

You provide SMS-specific configuration information to the SMS event gateway in a configuration file. You specify the configuration file location when you configure the SMS event gateway instance in the ColdFusion Administrator. ColdFusion provides a sample SMS event gateway configuration file in *cf_root*\WEB-INF\cfusion\gateway\config\sms-test.cfg on J2EE configurations, and *cf_root*\gateway\config\sms-test.cfg on server configurations. The following table describes the configuration file contents.

Note

The following configuration information describes the configuration fields, but does not include detailed explanations of SMPP-specific terminology, listings of all valid values of properties that are defined in the SMPP specification, or explanations of how to select appropriate SMPP-specific values for your application. For further information, see documentation on the SMPP 3.4 protocol at www.smsforum.net/ and other publicly available documentation. Your SMS service provider generally specifies requirements for several of these configuration values. Consult the provider documentation.

Property	Default	Description
ip-address		IP address of the SMSC, as specified by the SMPP provider. For the ColdFusion SMS test server, you normally use 127.0.0.1.
port	0	Port number to bind to on the SMSC. The ColdFusion SMS test server uses port 7901.
system-id		Name that identifies the event gateway to the SMSC, as established with the SMPP provider. To connect to the ColdFusion SMS test server, the system-id must be cf.
password		Password for authenticating the event gateway to the SMSC. To connect to the ColdFusion SMS test server, the password must be cf.
source-ton	1	Type of Number (TON) of the source address, that is, of the address that the event gateway uses for outgoing messages, as specified in the SMPP specification. Values include 0, unknown; 1, international number; 2, national number.

© 2014 Adobe Systems Incorporated. All rights reserved.

source-npi	1	Numeric Plan Indicator (NPI) of the source address as specified in the SMPP specification. Values include 0, unknown; 1, ISDN.
source-address	empty string	Address (normally, a phone number) of the event gateway. Identifies the sender of outgoing messages to the SMSC.
addr-ton	1	TON for the incoming addresses that this event gateway serves.
addr-npi	1	NPI for the incoming addresses that this event gateway serves.
address-range		The range of incoming addresses (phone numbers) that remote devices can use to send messages to the event gateway instance; often, the same as the `source-address`.
message-rate	100	Integer or decimal value that specifies the number of messages the gateway is allowed to send to your service provider per second. 0 is unlimited.
mode	synchronous	Message transmission mode: • synchronous:The gateway waits for the response from the server when sending a message. In this mode, the `SendGatewayMessage` CFML function returns the SMS `messageID` of the message, or an empty string if an error occurs. • asynchronous:The gateway does not wait for a response. In this mode, the `SendGatewayMessage` CFML function always returns an empty string.

© 2014 Adobe Systems Incorporated. All rights reserved.

network-retry	no	Gateway behavior when a network error occurs while trying to deliver a message: • yes:The gateway queues the message for delivery when the gateway is able to rebind to the SMSC. Retrying is useful if the gateway is in asynchronous mode, where the CFML `SendGatewayMessage` function does not return an error. • no:The gateway does not retry sending the message.
transient-retry	no	Gateway behavior when the SMSC returns an error that indicates a transient error, where it may be able to accept the message in the future: • yes:The gateway attempts to resend the message. Retrying is useful if the gateway is in asynchronous mode, where the CFML `SendGatewayMessage` function does not return an error. • no:The gateway does not retry sending the message.
cfc-method	onIncomingMessage	Listener CFC method for ColdFusion to invoke when the gateway gets incoming messages.
destination-ton	1	Default TON of addresses for outgoing messages.
destination-npi	1	Default NPI of addresses for outgoing messages.
service-type	empty string	Type of messaging service; can be empty or one of the following values: CMT, CPT, VMN, VMA, WAP, or USSD.
system-type	empty string	Type of system (ESME, External Short Message Entity); used when binding to the SMSC. Some SMSCs might be able to send responses that are specific to a given type of ESME. Normally set to SMPP.

© 2014 Adobe Systems Incorporated. All rights reserved.

receive-timeout	-1 (do not time out)	The time-out, in seconds, for trying to receive a message from the SMSC after it establishes a connection. To wait indefinitely until a message is received, set the `receive-timeout` to -1.
ping-interval	60	Number of seconds between EnquireLink messages that the event gateway sends to the server to verify the health of the connection.
retries	-1 (try forever)	Number of times to retry connecting to the SMSC to send a message before the gateway goes into a failed state. If the gateway is in a failed state, the `getStatus` method returns FAILED, and theColdFusion Administrator shows the gateway status as Failed. The gateway must be restarted before it can be used.
retry-interval	10	Number of seconds between connection retries.

You can also set the following values in each outgoing message: source-ton, source-npi, source-address, destination-ton, destination-npi, and service-type. The message field names differ from the configuration file property names.

© 2014 Adobe Systems Incorporated. All rights reserved.

Handling incoming messages-SMS event gateway

The SMS event gateway handles messages that are contained in deliver_sm PDUs. These PDUs request the gateway to deliver one of the following types of message:

- A user- or application-generated text message
- A message disposition response

> **Note**
>
> The SMS event gateway does not handle messages that are contained in data_sm PDUs.

The event gateway sends the object to event gateway services, which delivers it to the listener CFC. The CFEvent object that the listener CFC receives contains the following fields:

> **Note**
>
> Consider SMS messages and any other data that enters through an Event Gateway handler to be potentially hostile. For example, if SMS data is archived in a SQL database, an attacker could construct a message that modifies or deletes data, or even takes over the SQL Server. Therefore, be sure to perform Event Gateway input validation, just as you would validate web form input.

Field	Value
CfcMethod	Listener CFC method name
Data.dataCoding	Character set or the noncharacter data type of the message contents
Data.destAddress	Address to which the message was sent
Data.esmClass	Message type
Data.MESSAGE	Message contents
Data.messageLength	Length of the MESSAGE field
Data.priority	Message priority level, in the range 0-3
Data.protocol	GSM protocol; not used for other networks
Data.registeredDelivery	Requested type of delivery receipt or acknowledgment, if any
Data.sourceAddress	Address of the device that sent this message
GatewayType	Always SMS

© 2014 Adobe Systems Incorporated. All rights reserved.

OriginatorID	Address of the device that sent the message

For a detailed description of each field, see SMS Gateway incoming message CFEvent structure in the *CFML Reference*.
The CFC's listener method extracts the message from the Arguments.CFEvent.Data.MESSAGE field and acts on it as appropriate for the application. If necessary, the listener can use two fields to determine the required action:

- CFEvent.Data.esmClass indicates the type of information in the MESSAGE field.
- CFEvent.Data.registeredDelivery indicates whether the sender requested any type of delivery receipt or acknowledgment.

CFEvent.Data.esmClass field

The CFEvent.Data.esmClass field identifies whether the CFEvent.Data.Message field contains a message, or any of the following types of message disposition responses. For these responses, the CFEvent object Data.MESSAGE field contains the acknowledgment or receipt information.

- **SMSC Delivery Receipt** An indication of the message's final status, sent by the SMSC. The short message text includes the message ID of the original message, the number of messages sent and delivered (for messages sent to a distribution list), the date and time that the message was sent and delivered or otherwise disposed of, the message disposition, a network-specific error code (if available), and the first 20 bytes of the message. For details of the SMSC delivery receipt message structure, see Appendix B of the SMS 3.4 specification.
- **SME Delivery Acknowledgement** An indication from the recipient device that the user has read the short message. Supported by TDMA and CDMA wireless networks only.
- **SME Manual/User Acknowledgement** An application-generated reply message sent in response to an application request message. Supported by TDMA and CDMA wireless networks only.
- **Intermediate Delivery Notification** A provider-specific notification on the status of a message that has not yet been delivered, sent during the SMSC retry lifetime for the message. Intermediate Notification support depends on the SMSC implementation and SMSC service provider. For more information, see your provider documentation. When you send a message, you can request any combination of message disposition responses in the outgoing message's `registered_delivery` parameter. If your application requests responses, the listener CFC must be prepared to handle these messages, as appropriate.

CFEvent.Data.registeredDelivery field

The CFEvent.Data.registeredDelivery field indicates whether the message sender has requested a receipt or acknowledgment. Your application can respond to a request for an SME Delivery Acknowledgement or an SME Manual/User Acknowledgement. (Only the SMSC sends the other notification types.) For more information on these notification types, see the SMS 3.4 specification. Appendix B contains detailed information on the information that you must place in the shortMessage field of the returned acknowledgment message.

Incoming message handling example

The following example code is an SMS-only version of the echo.cfc example that is included in the ColdFusion gateway/cfc/examples directory. This example shows the minimal code required to receive and respond to an SMS message.

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cfcomponent displayname="echo" hint="Process events from the test gateway and
return echo">
<cffunction name="onIncomingMessage" output="no">
 <cfargument name="CFEvent" type="struct" required="yes">
 <!--- Get the message --->
 <cfset data=cfevent.DATA>
 <cfset message="#data.message#">
 <!--- where did it come from? --->
 <cfset orig="#CFEvent.originatorID#">
 <cfset retValue = structNew()>
 <cfset retValue.command = "submit">
 <cfset retValue.sourceAddress = arguments.CFEVENT.gatewayid>
 <cfset retValue.destAddress = arguments.CFEVENT.originatorid>
 <cfset retValue.shortMessage = "echo: " & message>
 <!--- send the return message back --->
 <cfreturn retValue>
</cffunction>
</cfcomponent>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

Sending outgoing messages-event gateway

Your ColdFusion application can send `submit`, `submitMulti`, and `data` commands to the event gateway in an outgoing message.

The submit command

To send a message to a single destination address in an SMPP SUBMIT_SM PDU, the structure used in the *Data* parameter of a `SendGatewayMessage` function or the return variable of the CFC listener method normally has the following fields:

Field	Contents
command	If present, the value must be `"submit"`. If you omit this field, the event gateway sends a submit message.
shortMessage_or_messagePayload	The Message contents. specify one of these fields, but not both. The SMPP specification imposes a maximum size of 254 bytes on the `shortMessage` field, and some carriers could limit its size further. The `message Payload` field can contain up to 64 K bytes; it must start with 0x0424, followed by 2 bytes specifying the payload length, followed by the message contents.
destAddress	The address to which to send the message (required).
sourceAddress	The address of this application. You can omit this field if it is specified in the configuration file.

You can also set optional fields in the structure, such as a field that requests a delivery receipt. For a complete list of fields, see submit command in the *CFML Reference*. For detailed descriptions of these fields, see the documentation for the SUBMIT_MULTI PDU in the SMPP3.4 specification, which you can download from the SMS Forum at www.smsforum.net/.

> **Note**
>
> To send long messages, you can separate the message into multiple chunks and use a submit command to send each chunk separately. In this case, a CFC would use multiple `SendGateway Message` functions, instead of the `cfreturn` function.

Example: Using the submit command in sendGatewayMessage function

The following example from a CFM page uses a `sendGatewyMessage` CFML function with a `submit` command to send an SMS messages that you enter in the form. This example uses the SMS gateway that is configured in the ColdFusion installation, and sends the message to the SMS client simulator.

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<h3>Sending SMS From a Web Page Example</h3>
<cfif IsDefined("form.oncethrough") is "Yes">
<cfif IsDefined("form.SMSMessage") is True AND form.SMSMessage is not "">
 <h3>Sending Text Message: </h3>
  <cfoutput>#form.SMSMessage#</cfoutput><br>
  <cfscript>
  /* Create a structure that contains the message. */
  msg = structNew();
  msg.command = "submit";
  msg.destAddress = "5551234";
  msg.shortMessage = form.SMSMessage;
  ret = sendGatewayMessage("SMS Menu App - 5551212", msg);
  </cfscript>
</cfif>
<hr noshade>
</cfif>
<!--- begin by calling the cfform tag --->
<cfform action="command.cfm" method="POST">
 SMS Text Message: <cfinput type="Text" name="SMSMessage" value="Sample text
Message" required="No" maxlength="160">
<p><input type = "submit" name = "submit" value = "Submit">
<input type = "hidden" name = "oncethrough" value = "Yes">
</cfform>
</body>
</html>
```

For a simple example of a listener CFC uses the submit command to echo incoming SMS messages to the message originator, see *Incoming message handling example* in Handling incoming messages-SMS event gateway.

The submitMulti command

To send a single text message to multiple recipients using an SMPP SUBMIT_MULTI PDU, the *Data* parameter of a `SendGatewayMessage` function or the return variable of the CFC listener method normally has the following fields:

Field	Contents
command	Must be `"submitMulti"`.
shortMessage_or_messagePayload	The message contents. Specify one of these fields, but not both. The SMPP specification imposes a maximum size of 254 bytes on the `shortMessage` field, and some carriers could limit its size further. The `messagePayload` field can contain up to 64 K bytes; it must start with 0x0424, followed by 2 bytes specifying the payload length, followed by the message contents.
destAddress	A ColdFusion array of destination addresses (required).You cannot specify individual TON and NPI values for these addresses; all must conform to a single setting.

© 2014 Adobe Systems Incorporated. All rights reserved.

sourceAddress	The address of this application; you can omit this field if it is specified in the configuration file.

You can also set optional fields in the structure, such as a field that requests delivery receipts. For a complete list of fields, see *submitMulti command* in the *CFML Reference*. For detailed descriptions of these fields, see the documentation for the SUBMIT_MULTI PDU in the SMPP 3.4 specification, which you can download from the SMS Forum at www.smsforum.net/.

Example: Using the submitMulti command in an onIncomingMessage method

The following example `onIncomingMessage` method sends a response that echoes an incoming message to the originator address, and sends a copy of the response to a second address. To test the example, run two instances of the ColdFusion SMS client application. Use the default phone number of 5551212 for the first, and set the second one to have a phone number of 555-1235. (Notice that the second phone number requires a hyphen (-).) Send a message from the first simulator, and the response appears in both windows.

```
<cffunction name="onIncomingMessage" output="no">
 <cfargument name="CFEvent" type="struct" required="yes">
 <!--- Get the message. --->
<cfset data=CFEvent.DATA>
<cfset message="#data.message#">
 <!--- Create the return structure. --->
 <cfset retValue = structNew()>
 <cfset retValue.command = "submitmulti">
 <cfset retValue.sourceAddress = arguments.CFEVENT.gatewayid>
 <cfset retValue.destAddresses=arraynew(1)>
 <!--- One destination is incoming message originator;
     get the address from CFEvent originator ID. --->
 <cfset retValue.destAddresses[1] = arguments.CFEvent.originatorid>
 <cfset retValue.destAddresses[2] = "555-1235">
 <cfset retValue.shortMessage = "echo: " & message>
<cfreturn retValue>
</cffunction>
</cffunction>
```

The data command

To send binary data to a single destination address in an SMPP DATA_SM PDU, the *Data* parameter of a `SendGatewayMessage` function or the return variable of the CFC listener method must have the following fields:

Field	Contents
command	Must be `"data"` .
messagePayload	Message data. To convert data to binary format, use the ColdFusion `toBinary` function.
destAddress	Address to which to send the message.

© 2014 Adobe Systems Incorporated. All rights reserved.

sourceAddress	Address of this application; can be omitted if specified in the configuration file.

You can also set optional fields in the structure, such as a field that requests a delivery receipt. For a complete list of fields, see data command in the *CFML Reference*. For detailed descriptions of these fields, see the documentation for the SUBMIT_MULTI PDU in the SMPP3.4 specification, which you can download from the SMS Forum at www.smsforum.net/.

Example: Using the data command

The following example `onIncomingMessage` method converts an incoming message to binary data, and sends the binary version of the message back to the originator address:

```
<cffunction name="onIncomingMessage" output="no">
 <cfargument name="CFEvent" type="struct" required="yes">
 <!--- Get the message. --->
<cfset data=CFEvent.DATA>
<cfset message="#data.message#">
 <!--- Create the return structure. --->
 <cfset retValue = structNew()>
 <cfset retValue.command = "data">
 <!--- Sending to incoming message originator; get value from CFEvent. --->
 <cfset retValue.destAddress = arguments.CFEvent.originatorid>
 <cfset retValue.messagePayload = tobinary(tobase64("echo: " & message))>
 <cfreturn retValue>
</cffunction>
```

Controlling SMS message sending and response

This documentation describes some of the more common options for sending messages, and how they affect your application. For information on other ways to configure outgoing message, see the SMPP specification.

Synchronization mode

You can specify asynchronous or synchronous message mode in the gateway configuration file.

- If you specify *asynchronous* mode, the `sendGatewayMessage` function returns an empty string when the gateway submits the message to service code for sending to the SMSC. ColdFusion logs errors that occur after this point, such as if a message sent by the gateway to the SMSC times out or if the gateway gets an error response; the application does not get notified of any errors.
- If you specify *synchronous* mode (the default), the `sendGatewayMessage` function does not return until the gateway gets a response from the SMSC or the attempt to communicate times out. If the message is sent successfully, the function returns the SMPP message ID string. If an error occurs, the function returns an error string.
  Use synchronous mode if your application must determine whether its messages reach the SMSC. Also use synchronous mode if the application requests return receipts.

> **Note**
>
> If you use synchronous mode and the SMSC returns the messgeID as a hexadecimal string, ColdFusion converts it automatically to its decimal value.

© 2014 Adobe Systems Incorporated. All rights reserved.

The following example is an expansion of *Example: Using the submit command in sendGatewayMessage function* di scussed in *The submit command*. It checks for a nonempty return value and displays the message number returned by the SMS. This example uses the SMS gateway that is configured when ColdFusion is installed. If you change the gateway specified in the `SendGatewayMessage` function, make sure that your gateway's configuration file specifies synchronous mode.

```
<h3>Sending SMS From a Web Page Example</h3>

<cfif IsDefined("form.oncethrough") is "Yes">
 <cfif IsDefined("form.SMSMessage") is True AND form.SMSMessage is not "">
   <h3>Sending a Text Message: </h3>
   <cfoutput>#form.SMSMessage#</cfoutput><br><br>

  <cfscript>
   /* Create a structure that contains the message. */
   msg = structNew();
   msg.command = "submit";
   msg.destAddress = "5551234";
   msg.shortMessage = form.SMSMessage;
   ret = sendGatewayMessage("SMS Menu App - 5551212", msg);
  </cfscript>
 </cfif>
 <cfif isDefined("ret") AND ret NEQ "">
  <h3>Text message sent</h3>
  <cfoutput>The Message Id is: #ret#</cfoutput>
  <br><br>
 </cfif>
 <hr noshade>
</cfif>

<!--- begin by calling the cfform tag --->
<cfform>
 SMS Text Message: <cfinput type="Text" name="SMSMessage"
  value="Sample text Message" required="No" maxlength="160">
 <p><input type = "submit" name = "submit" value = "Submit">
 <input type = "hidden" name = "oncethrough" value = "Yes">
</cfform>
```

Optional parameters for outgoing SMS

You can send vendor-specific optional parameters by way of ColdFusion SMS gateway.
To set the optional parameters, specify them using the `optionalparameter` attribute.
If the gateway receives optional parameters in a message, they are included in the data struct that is returned to the listener CFC method named `onIncomingMessage` under the `optionalParameters` key.
The following code describes how to add optional parameters:

```
params=StructNew();
params["parameter"]=BinaryDecode("string","binaryencoding");
params["parameter"]=CharsetDecode("string, encoding");
outgoingSMS.optionalParameters=params;
```

- `parameter`: Vendor-specific optional parameter.

© 2014 Adobe Systems Incorporated. All rights reserved.

- `BinaryDecode`: See BinaryDecode in *CFML Reference.*
- `CharsetDecode`: See CharsetDecode in *CFML Reference.*
  If there is only one optional parameter, you can instead use the following code:

```
out.optionalParameter=parameter;
out.optionalParameterValue="value";
```

Note

Ensure that the Java Short.decode(String) function can parse the key or the value is a byte.

Message disposition notification

You can request the SMSC to return a message disposition response to indicate the fate of your message. To request a delivery receipt, include a RegisteredDelivery field in the *Data* parameter of a `SendGatewayMessage` function or the return variable of the CFC listener method. This field can have the following values:

Value	Meaning
0	(Default) Do not return delivery information.
1	Return a receipt if the message is not delivered before the time-out.
2	Return a receipt if the message is delivered or fails.

Some providers also support intermediate delivery notifications. For more information, see your provider's documentation.
To use delivery notification, send your message using synchronous mode, so you get a message ID. Your incoming message routine must be able to handle the receipts (see Handling incoming messages).

Validity period

You can change the length of time that the SMSC keeps a message and tries to deliver it. (Often the default value is 72 hours.) For a message sent to an emergency worker, for example, a short validity period (such as 15 min.) can be appropriate. To change this value, include a validityPeriod field in the *Data* parameter of a `SendGatewayMessage` function or the return variable of the CFC listener method. To specify a time period, use the following pattern: *YYMMDDhhmmsst_00R. In this pattern, _t* indicates tenths of seconds, and 00R specifies that this value is a relative time period, not a date-time value. The time format 000001063000000R, for example, specifies a validity period of 0 years, 0 months, 1 day, 6 hours, 30 minutes.

© 2014 Adobe Systems Incorporated. All rights reserved.

ColdFusion SMS development tools

ColdFusion provides the following tools for developing SMS applications:

- SMS test server
- SMS client simulator

SMS test server

The ColdFusion SMS test server is a lightweight SMSC simulator that listens on TCP/IP port 7901 for SMPP connection requests from other SMS resources, such as ColdFusion SMS gateways or the SMS client simulator. The resource supplies a user name, password, and telephone number (address). The user name and password must correspond to a name and password in the simulator's configuration file (described later).
After the SMS test server establishes a connection, it listens for incoming messages and forwards them to the specified destination address, if the destination address also corresponds to an existing SMPP connection.
The SMS test server lets you develop SMS applications without having to use an external SMSC supplier such as a telecommunications provider. The server supports the ColdFusion SMS gateway `submit` and `submitMulti` comm ands. It also accepts, but does not deliver messages sent using the SMS gateway `data` command. It does not include any store and forward capabilities.
Start the SMS test server by clicking the Start SMS Test Server button on the Settings page in the Event Gateways area in the ColdFusion Administrator.

> **Note**
>
> The SMS test server does not automatically restart when you restart ColdFusion. Manually restart the server if you restart ColdFusion.

The SMS test server reads the *cf_root*\WEB-INF\cfusion\lib\sms-test-users.txt file on J2EE configurations or *cf_root*\l ib\sms-test-users.txt file on server configurations to get valid user names and passwords. ColdFusion includes a version of this file configured with several names and passwords. One valid combination is user name cf and password cf. You can edit this file to add or delete entries. The file must include a name and password entry for each user that connects to the test server. Separate user entries with blank lines, as the following example shows:

```
name=cf
password=cf

name=user1
password=user1
```

SMS client simulator

The ColdFusion SMS client simulator is a simple External Short Message Entity (ESME) that simulates a (limited-function) mobile phone. It can connect to the SMS test server and exchange messages with it.

> **Note**
>
> On UNIX and Linux systems, the client simulator requires X-Windows.

Use the following procedure to use the simulator.

Use the SMS simulator

1. Ensure that you have started the SMS test server and configured and started an SMS event gateway

© 2014 Adobe Systems Incorporated. All rights reserved.

instance in ColdFusion Administrator.

2. Run SMSClient.bat in Windows or SMSClient.sh on UNIX or Linux. These files are located in the *cf_root*\WEB-INF\cfusion\bin directory on J2EE configurations and the *cf_root*\bin directory on server configurations.If you installed a pure Java version of ColdFusion, for example, on Apple Mac OS X systems, enter the following command to start the simulator:

```
java -jar cf_root/WEB-INF/cfusion/lib/smpp.jar
```

3. A dialog box appears, requesting the server, port, user name, password, and the phone number to use for this device. The simulator sends this phone number as the source address, and accepts SMS messages sent by the SMSC server to it using this number as the destination address. To connect to the SMS test server, accept the default values and specify an arbitrary phone number; you can also specify any user name-password pair that is configured in the *cf_root*\WEB-INF\cfusion\lib\sms-test-users.cfg file or *cf_root*\lib\sms-test-users.cfg or file.
4. Click Connect.
5. The SMS device simulator client appears. In the Send SMS To field, enter a phone number in the address-range property specified in the configuration file of the SMS event gateway that you want to send messages to.
6. Type a message directly into the message field (to the left of the Send button), or use the simulator keypad to enter the message.
7. Click the Send button.
   The client simulator has a Connection menu with options to connect and disconnect from the SMSC server, and to check the connection. The connection information appears in a status line at the bottom of the client.

© 2014 Adobe Systems Incorporated. All rights reserved.

Sample SMS application

The following CFC implements a simple employee phone directory lookup application. The user sends a message containing some part of the name being looked up (a space requests all names). The `onIncomingMessage` respon se depends on the number matches.

- If no match exists, the `onIncomingMessage` function returns a message indicating that no matches exist.
- If one match exists, the function returns the name, department, and phone number.
- If up to ten matches exist, the function returns a list of the names preceded by a number that the user can enter to get the detailed information.
- If over ten matches exist, the function returns a list of only the first ten names. A more complex application could let the user get multiple lists of messages to provide access to all names.
- If the user enters a number, and previously got a multiple-match list, the application returns the information for the name that corresponds to the number.
  The following listing shows the CFC code:

```
<cfcomponent>
 <cffunction name="onIncomingMessage">
  <cfargument name="CFEvent" type="struct" required="YES">
  <!--- Remove any extra white space from the message. --->
  <cfset message =Trim(arguments.CFEvent.data.MESSAGE)>
  <!--- If the message is numeric, a previous search probably returned a
   list of names. Get the name to search for from the name list stored in
   the Session scope. --->
  <cfif isNumeric(message)>
   <cfscript>
    if (structKeyExists(session.users, val(message))) {
     message = session.users[val(message)];
    }
   </cfscript>
  </cfif>

  <!--- Search the database for the requested name. --->
  <cfquery name="employees" datasource="cfdocexamples">
   select FirstName, LastName, Department, Phone
   from Employees
   where 0 = 0
   <!--- A space indicates the user entered a first and last name. --->
   <cfif listlen(message, " ") eq 2>
    and FirstName like '#listFirst(message, " ")#%'
    and LastName like '#listlast(message, " ")#%'
   <!--- No space: the user entered a first or a last name. --->
   <cfelse>
    and (FirstName like '#listFirst(message, " ")#%'
    or LastName like '#listFirst(message, " ")#%')
   </cfif>
  </cfquery>

  <!--- Generate andreturn the message.--->
  <cfscript>
   returnVal = structNew();
   returnVal.command = "submit";
   returnVal.sourceAddress = arguments.CFEVENT.gatewayid;
   returnVal.destAddress = arguments.CFEVENT.originatorid;

   //No records were found.
```

© 2014 Adobe Systems Incorporated. All rights reserved.

```
    if (employees.recordCount eq 0) {
     returnVal.shortMessage = "No records found for '#message#'";
    }
    //One record was found.
    else if (employees.recordCount eq 1) {
    // Whitespace in the message text results in bad formatting,
    // so the source cannot be indented.
     returnVal.shortMessage = "Requested information:
#employees.firstName# #employees.lastName#
#employees.Department#
#employees.Phone#";
    }
    //Multiple possibilities were found.
    else if (employees.recordCount gt 1) {
     //If more than ten were found, return only the first ten.
     if (employees.recordCount gt 10)
     {
      returnVal.shortMessage = "First 10 of #employees.recordCount# records";
     }else{
      returnVal.shortMessage = "Records found: #employees.recordCount#";
     }
     // The session.users structure contains the found names.
     // The record key is a number that is also returned in front of the
     // name in the message.
     session.users = structNew();
     for(i=1; i lte min(10, employees.recordCount); i=i+1)
     {
      // These two lines are formatted to prevent extra white space.
      returnVal.shortMessage = returnVal.shortMessage & "
#i# - #employees.firstName[i]# #employees.lastName[i]#";
      // The following two lines must be a single line in the source
      session.users[i]="#employees.firstName[i]# #employees.lastName[i]#";
     }
    }
    return returnVal;
   </cfscript>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

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

#back to top

© 2014 Adobe Systems Incorporated. All rights reserved.

Using the FMS event gateway

The FMS event gateway provides interfaces between the Flash Media Server 2 and the Adobe ColdFusion server. As a result, ColdFusion applications and Adobe Flash clients can share data.
Before you use the gateway, become familiar with ColdFusion event gateway principles and programming techniques (see Using Event Gateways). A basic knowledge of Flash Media Server is also helpful.

© 2014 Adobe Systems Incorporated. All rights reserved.

About Flash Media Server

Flash Media Server 2 is the newest version of Flash Communication Server. Flash Media Server 2 offers traditional streaming media capabilities and a flexible development environment for creating and delivering innovative, interactive media applications. You can use Flash Media Server to create and deliver the following media experiences:

- Video on Demand
- Live web-event broadcasts
- Mp3 streaming
- Video blogging
- Video messaging
- Multimedia chat environments
  To learn more about and to download the Flash Media Server, go to the Adobe website. at www.adobe.com/go/learn_cfu_flashmediaserver_en.

© 2014 Adobe Systems Incorporated. All rights reserved.

How ColdFusion and Flash Media Server interact through the FMS gateway

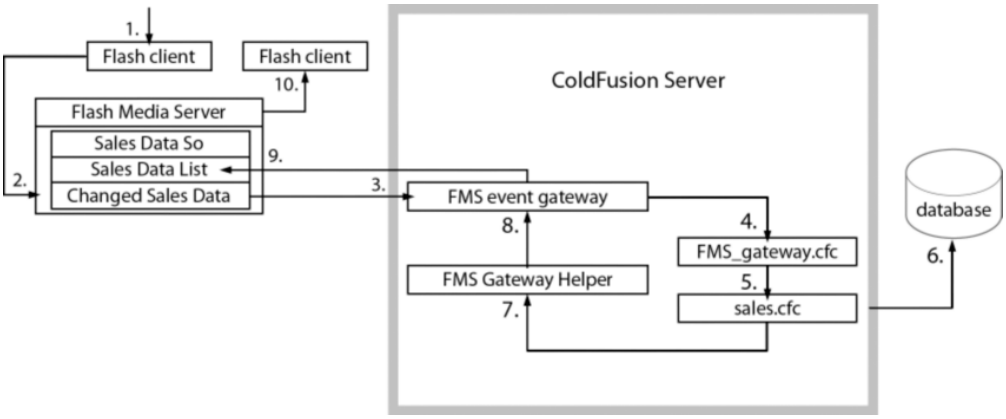
The FMS event gateway lets you modify data through the ColdFusion application or the Flash client, and reflect the change in the Flash Media Server shared object. The FMS event gateway listens to the shared object, and notifies ColdFusion when other clients modify shared objects. The FMS event gateway also lets ColdFusion modify shared objects.
ColdFusion provides the following tools for developing FMS applications:

- **FCSj.jar** The JAR file that implements the Java API to communicate with Flash Media Server.
- **FMSGateway** The class for the FMS event gateway type. You implement your FMS application by creating a ColdFusion application that uses an instance of the FMSGateway class to communicate with one or more Flash Media Server.

Modifying data in the Flash client

The FMS event gateway listens to Flash Media Server shared objects, and notifies ColdFusion when a Flash client modifies a shared object. The following steps occur when a Flash client modifies a Flash Media Server shared object:

1. A user modifies data in the Flash client.
2. Flash Media Server updates the appropriate shared object.
3. Flash Media Server notifies the FMS event gateway.
4. The FMS event gateway calls the appropriate methods in CFCs in your ColdFusion application. These CFCs perform all actions required, including notifying the FMS Gateway Helper to update the shared object.
5. The FMS Gateway Helper sends a message to the FMS event gateway to update the shared object.
6. The FMS event gateway updates the shared object.
7. Flash Media Server notifies all the Flash clients that it modified the shared object. As a result, the Flash clients reflect the change.
   The following image shows the interaction between Flash Media Server, the FMS event gateway, and the ColdFusion application:



Modifying data in a ColdFusion application

The FMS event gateway lets ColdFusion applications modify Flash Media Server shared objects. The following steps occur when data that affects a shared object is modified in a ColdFusion application:

1. The user submits a form that contains data to modify using a ColdFusion page.
2. The ColdFusion page calls the appropriate CFC, which contains a method to update the database.
3. The method in the CFC updates the database and calls a method in the FMS Gateway Helper.
4. The FMS Gateway Helper calls the FMS event gateway to update the appropriate shared object.
5. Flash Media Server updates the shared object.
6. Flash Media Server notifies the Flash client that a shared object has changed.

© 2014 Adobe Systems Incorporated. All rights reserved.

7. The Flash client makes the changes in its content as appropriate.

© 2014 Adobe Systems Incorporated. All rights reserved.

Application development and deployment process

The following is a typical process for developing and deploying an application that uses the FMS event gateway:

1. Design your application.
2. Configure an FMS event gateway instance to use the Flash Media Server.
3. Write your ColdFusion CFCs, CFM pages, and any other application elements.
4. Create or identify a Flash client that manipulates a Flash Media Server shared object to test your ColdFusion application.
5. Test your application using Flash Media Server and the Flash client.
6. Make the application publicly available.

Configuring an FMS event gateway

You provide FMS event gateway-specific configuration information to the FMS event gateway in a configuration file. You specify the configuration file location when you configure the FMS event gateway instance in the ColdFusion Administrator. The configuration file contains the URL of the Flash Media Server application and the name of the Flash Media Server shared object. The following example is a sample configuration file:

```
#
# FMS event gateway configuration
#

# This is the URL to the Flash Media Server application.
rtmpurl=rtmp://localhost/SalesDataApp

# This is the shared object you would like this gateway to connect and listen to.
sharedobject=SalesDataSO
```

FMS event gateway GatewayHelper class methods

The following table lists the FMS event gateway GatewayHelper class methods:

Method	Description
`setProperty`	Sets the property of the Flash Media Server shared object. The following parameters are valid:`name`: The string that contains the name of the shared object.`value`: The shared object.
`getProperty`	Gets the property of the Flash Media Server shared object. The following parameters are valid:`name`: The string that contains the name of the shared object.

Data translation

ColdFusion and Flash Media Server use different data types; therefore, data translation is required to pass data from one to the other. In addition to basic data types such as numeric, String, and Boolean, you can pass ColdFusion queries, structures, and arrays to Flash Media Server. You pass a ColdFusion query object to Flash Media Server as an array of java.util.HashMap. Each HashMap object in the array contains a key-value pair for column names and values for each row in the query. When you pass a ColdFusion array to Flash Media Server, the FMS event gateway converts it to a Java array of objects. When you pass a ColdFusion structure, no conversion is

© 2014 Adobe Systems Incorporated. All rights reserved.

required.
The FMS event gateway does not support passing CFCs in shared objects.

#back to top

© 2014 Adobe Systems Incorporated. All rights reserved.

Using the Data Services Messaging Event Gateway

Using the Data Services Messaging gateway type provided with Adobe ColdFusion, you can create applications that send messages to and receive messages from LiveCycle Data Services ES. You configure the Data Services Messaging gateway and write and test an application that uses the event gateway.

Before you use the Data Services Messaging gateway, become familiar with ColdFusion event gateway principles and programming techniques (see Using Event Gateways). Also be familiar with Adobe LiveCycle Data Services ES.

© 2014 Adobe Systems Incorporated. All rights reserved.

About Flex and ColdFusion

ColdFusion includes the Data Services Messaging event gateway, which uses the ColdFusion event Gateway Adapter to send messages to and receive messages from the LiveCycle Data Services ES. This means that ColdFusion applications and Flex applications can publish to and consume events from the same event queue.

> **Note**
>
> To use the Data Services Messaging event gateway to interact with a Flex application, the Flex application must be running on LiveCycle Data Services ES.

How ColdFusion and Flex interact

You can send messages from a ColdFusion application to a Flex application, through the Data Services Messaging event gateway. Conversely, you can send messages from a Flex application to a ColdFusion application.
Either the ColdFusion application or the Flex application can initiate sending a message.

1. The Flex application generates a message.
2. The Flex Message Service passes the message to the ColdFusion Event Gateway Adapter.
3. The ColdFusion Event Gateway Adapter sends the message to the Data Services Messaging event gateway, by using Java Remote Method Invocation (Java RMI).
4. The Data Services Messaging event gateway and the ActionScript translator convert ActionScript 3.0 data types to the appropriate ColdFusion values and add the message to the event gateway queue.y
5. The ColdFusion server calls the onIncomingMessage method of the Data Services Messaging event gateway listener CFC.
6. The ColdFusion application generates a message, which it sends to the ColdFusion server.
7. The ColdFusion server sends the message to the Data Services Messaging event gateway.
8. The Data Services Messaging event gateway and the ActionScript translator convert ColdFusion values to the appropriate ActionScript 3.0 data types and then the gateway sends the message to the ColdFusion Event Gateway Adapter.
9. The ColdFusion Event Gateway Adapter sends the message to the Flex Message Service.
10. The Flex Message Service passes the message to the Flex application.

> **Note**
>
> The RMI registry, which facilitates communication between the ColdFusion Event Gateway Adapter and the Data Services Messaging event gateway uses port 1099, which is the default port for Java RMI. You cannot change this port number. To ensure that the RMI registry provides registry service for both LiveCycle Data Services and ColdFusion, start LiveCycle Data Services first, and then start ColdFusion. If you stop Flex, restart LiveCycle Data Services, and then restart the gateway.

Application development and deployment process

The following is a typical process for developing and deploying a ColdFusion application that communicates with a Flex application through the Data Services Messaging event gateway:

1. Design your application.
2. Configure a Data Services Messaging event gateway instance.
3. Write your ColdFusion CFCs, CFM pages, and any other application elements.
4. Test your application using Flex Enterprise Services 2.
5. Make the application publicly available.

© 2014 Adobe Systems Incorporated. All rights reserved.

Configuring a Data Services Messaging event gateway

Although you can configure an instance of a Data Services Messaging event gateway by creating a configuration file and specifying that file as the configuration file when you create an instance of the event gateway, you can also provide the configuration information in the message sent from the Flex application. You provide configuration information to the Data Services Messaging event gateway in a configuration file to do either of the following:

- Have the Data Services Messaging event gateway send messages to Flex Enterprise Services 2 on a different computer
- Use the Data Services Messaging event gateway with a specific Flex destination, and ignore any destination specified in the message
  The Data Services Messaging event gateway configuration file is a simple Java properties file that contains the following properties:

Property	Description
`destination`	A hard-coded destination. If you specify this value, any destination information in the message is ignored.
`host`	The host name or IP address of the Flex Enterprise Services 2 server.

The following example is a configuration file:

```
#
# Flex event gateway configuration
#

# This is the destination of the messages.
destination=Gateway1

# host name or IP address of the Flex Enterprise Server
host=127.0.0.1
```

If you create a configuration file, save it in the *cf_root*/gateway/config/ directory, with the extension .cfg.

© 2014 Adobe Systems Incorporated. All rights reserved.

Sending outgoing messages to a Flex application

Your ColdFusion application sends a message to a Flex application by doing the following actions:

1. The ColdFusion application sends an outgoing message, in a `cfreturn` tag in the listener method of the listener CFC, or by calling the ColdFusion `SendGatewayMessage` function.
2. A method provided by the Data Services Messaging gateway gets called when you send an outgoing message.
   In outgoing messages sent from CFML, the following structure members are translated to the Flex message:

Name	Contents
body	Body of the message. Required.
CorrelationID	Correlation identifier of the message.
Destination	Flex destination of the message. Required if it is not specified in the configuration file.
Headers	If the message contains any headers, the CFML structure that contains the header names as keys and values.
LowercaseKeys	If the value is set to `yes`, the structure keys are converted to lowercase during creation of ActionScript types.
TimeToLive	Number of milliseconds during which this message is valid.

In addition, the Data Services Messaging event gateway automatically provides values for the following Flex message fields:

Name	Contents
MessageID	A UUID that identifies the message.
Timestamp	Time the message is sent.
ClientID	ID of the Data Services Messaging event gateway instance.

Note

A single instance of the Data Services Messaging event gateway can send messages to any destination that is registered with the ColdFusion Event Gateway Adapter. However, if the destination is configured in the Data Services Messaging gateway configuration file, the destination in the message is ignored.

Sending outgoing message example

© 2014 Adobe Systems Incorporated. All rights reserved.

The following example from a CFM page creates a structure that contains the message. The destination is the destination ID specified in the flex-services.xml file for the instance of the Data Services Messaging event gateway to send the message to. The body is the body of the message. The `sendGatewyMessage` CFML function sends the message to the instance of the gateway.

```
<cfset success = StructNew()>
<cfset success.msg = "E-mail was sent at " & Now()>
<cfset success.Destination = "gateway1">
<cfset ret = SendGatewayMessage("Flex2CF2", success)>
```

To ensure that properties maintain the correct case, define Flex-related information as follows:

```
myStruct['mySensitiveProp']['myOtherSensitiveProp']
```

The following is an example of using headers to send to a specific subtopic of the destination:

```
<cfset var msg = structnew()>
<cfset msg.destiNation = 'ColdFusionGateway'>
<cfset msg.body = 'somebody'>
<cfset msg['headers']['DSSubtopic'] = 'somesubtopic'>
<cfset sendgatewaymessage('CF2FLEX2' , msg)>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

Handling incoming messages from a Flex application

When a Flex application sends a message to a ColdFusion application, the Data Services Messaging event gateway sends a CFEvent structure to the onIncomingMessage function of the configured CFC, with the following information mapped to the data of the event:

Name	Contents
body	Body of the message.
ClientID	ID of the client that sent the message.
CorrelationID	Correlation identifier of the message.
Destination	Flex destination of the message.
Headers	If the message contains any headers, the CFML structure that contains the header names as keys and values.
Timestamp	Timestamp of the message.

The incoming message data structure also includes the values of messageID and timeToLive from the Flex message.

Incoming message handling example

The following example places data that is contained in the body of the message from the Flex application into a structure. It then uses the contents of the structure to generate an e-mail message.

```
<cfcomponent displayname="SendEmail" hint="Handles incoming message from Flex">
 <cffunction name="onIncomingMessage" returntype="any">
 <cfargument name="event" type="struct" required="true">

  <!--- Create a structure to hold the message object sent from Flex--->
  <cfset messagebody = event.data.body>

  <!--- Populate the structure. --->
  <cfset mailfrom="#messagebody.emailfrom#">
  <cfset mailto="#messagebody.emailto#">
  <cfset mailsubject="#messagebody.emailsubject#">
  <cfset mailmessage ="#messagebody.emailmessage#">

  <!--- Send e-mail with values from the structure. --->
  <cfmail from="#mailfrom#"
   to="#mailto#"
   subject="#mailsubject#">
   <cfoutput>#mailmessage#</cfoutput>
  </cfmail>
 </cffunction>
</cfcomponent>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

If the Flex application sends the message in the header instead of in the body, you create and populate the structure, as the following example shows:

```
<cfset messageheader = StructNew()>
<cfset messageheader.sendto = event.data.headers.emailto>
<cfset messageheader.sentfrom = event.data.headers.emailfrom>
<cfset messageheader.subject = event.data.headers.emailsubject>
<cfset messageheader.mailmsg = event.data.headers.emailmessage>

<cfset mailfrom="#messageheader.sentfrom#">
<cfset mailto="#messageheader.sendto#">
<cfset mailsubject="#messageheader.subject#">
<cfset mailmessage ="#messageheader.mailmsg#">
```

New methods introduced in ColdFusion 9.0.1

The following new methods have been introduced in ColdFusion Messaging Gateway CFCs:

- `allowSend`
- `allowSubscribe}}Both the methods take {{subtopic` as the parameter. These methods help you place control over subscribing and sending data to a particular subtopic.

> **Note**
>
> To call these methods on their gateway CFC, specify the gateway id under the messaging destination in messaging-config.xml (Web_INF/Flex). By default, the value is `*`.

© 2014 Adobe Systems Incorporated. All rights reserved.

Data translation

The following table lists the ColdFusion data types and the corresponding Flash or ActionScript data type:

ColdFusion data type	Flash data type
String	String
Array	[] = Array
Struct	{} = untyped Object
Query	ArrayCollection
CFC	Class = typed Object (if a matching ActionScript class exists, otherwise the CFC becomes a generic untyped Object (map) in ActionScript)
CFC Date	ActionScript Date
CFC String	ActionScript String
CFC Numeric	ActionScript Numeric
ColdFusion XML Object	ActionScript XML Object

#back to top

© 2014 Adobe Systems Incorporated. All rights reserved.

Using the Data Management Event Gateway

Using the Data Management event gateway type provided with Adobe ColdFusion, you can have ColdFusion applications notify Adobe Flex applications when data managed by a destination has changed. You configure the Data Management event gateway and write an application that uses the event gateway.

Before using the Data Management event gateway, become familiar with ColdFusion event gateway principles and programming techniques (see Using Event Gateways). Also, be familiar with LiveCycle Data Services ES.

© 2014 Adobe Systems Incorporated. All rights reserved.

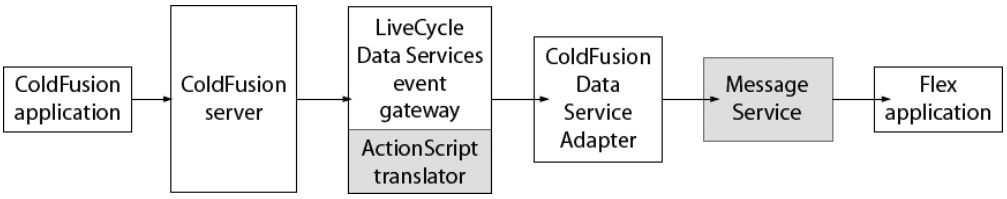
About ColdFusion and Flex

ColdFusion includes the Data Management event gateway, which uses the ColdFusion Data Service Adapter to send messages to LiveCycle Data Services ES. This means that ColdFusion applications can notify a Flex application about changes in the data that the destination manages.

> **Note**
>
> To use the Data Management event gateway to send messages to a Flex application, the Flex application must be running on LiveCycle Data Services ES.

How ColdFusion and Flex interact

You can send messages from a ColdFusion application to a Flex application through the Data Management event gateway. This gateway type only lets you send messages from a ColdFusion application to a Flex application. The following image shows the process in which a message is sent from the ColdFusion application to the Flex application:



1. The ColdFusion application generates a message, which it sends to the ColdFusion server.
2. The ColdFusion server sends the message to the Data Management event gateway.
3. The Data Management event gateway and the ActionScript translator convert ColdFusion values to the appropriate ActionScript 3.0 data types, and then the gateway sends the message to the ColdFusion Data Service Adapter.
4. The ColdFusion Data Service Adapter sends the message to the LiveCycle Data Services Message Service.
5. The Message Service passes the message to the Flex application. If you are running LiveCycle Data Services ES on the ColdFusion server, communication between LiveCycle Data Services ES and ColdFusion does not use RMI.If you are running LiveCycle Data Services ES remotely, to ensure that the RMI registry provides registry service for both LiveCycle Data Services ES and ColdFusion, start LiveCycle Data Services ES first, and then start ColdFusion. If you stop LiveCycle Data Services ES, restart LiveCycle Data Services ES, and then restart the gateway.If you are running LiveCycle Data Services ES remotely, the RMI registry, which facilitates communication between the ColdFusion Data Service Adapter and the Data Management event gateway uses port 1099. This port is the default value for Java RMI. You can change the port number by adding `-Dcoldfusion.rmiport=}}1234, replacing {{1234` with the appropriate port number, to the Java JVM arguments on both the ColdFusion server and the Flex server.

Application development and deployment process

The following is a typical process for developing and deploying a ColdFusion application that communicates with a Flex application through the Data Management event gateway:

1. Design your application.
2. Configure a Data Management event gateway instance.
3. Write your ColdFusion CFCs, CFM pages, and any other application elements.
4. Test your application using LiveCycle Data Services ES.
5. Make the application publicly available.

© 2014 Adobe Systems Incorporated. All rights reserved.

Configuring a Data Management event gateway

Although you can configure an instance of a Data Management event gateway by creating a configuration file and specifying that file as the configuration file when you create an instance of the event gateway, you can also provide the configuration information in the message. You provide configuration information to the Data Management event gateway in a configuration file to do either of the following:

- Have the Data Management event gateway send messages to LiveCycle Data Services ES on a different computer.
- Use the Data Management event gateway with a specific Flex destination, and ignore any destination specified in the message.
  The Data Management event gateway configuration file is a simple Java properties file that contains the following properties:

Property	Description
`destination`	A hard-coded destination. If you specify this value, any destination information in the message is ignored.
`host`	The host name or IP address of the LiveCycle Data Services ES server. Omit the host name if you are running LiveCycle Data Services ES as part of ColdFusion.

The following example is a configuration file:

```
#
# Data Management event gateway configuration
#
# This is the destination where messages are sent.
destination=myDestination

# Host name or IP address of the LiveCycle Data Services ES Server
host=127.0.0.1
```

If you create a configuration file, save it in the *cf_root*/gateway/config/ directory, with the extension .cfg.

Note

A single instance of the Data Management event gateway can send messages to any destination that is registered with the ColdFusion Data Service Adapter. However, if the destination is configured in the Data Management event gateway configuration file, the destination in the message is ignored.

© 2014 Adobe Systems Incorporated. All rights reserved.

Sending messages

Your ColdFusion application sends a message to a Flex application by calling the ColdFusion SendGatewayMessage function. In messages sent from CFML, the following structure members are translated to the Flex message:

Name	Contents
`destination`	Destination of the message. This entry is required if it is not specified in the configuration file.
`action`	Required. The notification action that is being performed: `create`, `delete`, `deleteID`, `refreshfill`, or `update`.
`item`	Required when `action="create"` or `action="delete".` The record that was added or deleted.
`identity`	Required when `action="deleteID"`. A structure that contains the identity properties (primary key) of the record that was deleted.
`fillparameters`	Optional. An array that contains the fills parameters that specify which fill operations to refresh.
`newversion`	Required when `action="update"`. The record that was updated.
`previousversion`	Optional. The previous record, before the update. This entry is used for conflict resolution.
`changes`	Optional when `action="update"`. A comma-delimited list or array of property names that were updated in the record. If you omit this entry, ColdFusion assumes that all properties changed. When you change a large number of records, you specifying the property names can improve performance.Required when `action="batch"`. An array of structures that contain the changes. You can batch multiple changes and send them in a single notification. The changes can be of different types, for example five updates, one delete, and two creates. Each event structure must contain an action.

Example

The following example creates a structure for each event type. It then creates a structure that contains the message. The message structure contains an array of event structures to send to Flex. The destination is the destination ID specified in the flex-services.xml file for the instance of the Data Management event gateway to send the message to. The body is the body of the message. The `sendGatewyMessage` CFML function sends the message to the instance of the gateway.

© 2014 Adobe Systems Incorporated. All rights reserved.

```
<cfscript>
// Create event
createEvent = StructNew();
createEvent.action = "create";
createEvent.item = newContact;

// Create update notification
updateEvent = StructNew();
updateEvent.action="update";
updateEvent.previousversion = oldContact;
updateEvent.newversion = updatedContact;

// Create delete notification
identity = StructNew();
identity["contactId"] = newId;
deleteEvent = StructNew();
deleteEvent.action = "deleteID";
deleteEvent.identity = identity;

// Send a batch notification
all = StructNew();
all.destination="cfcontact";
all.action="batch";
all.changes = ArrayNew(1);
all.changes[1] = createEvent;
all.changes[2] = updateEvent;
all.changes[3] = deleteEvent;
r = sendGatewayMessage("LCDS", all);
</cfscript>
```

© 2014 Adobe Systems Incorporated. All rights reserved.

Data translation - Developing guide

The following table lists the ColdFusion data types and the corresponding Adobe Flash or ActionScript data type:

ColdFusion data type	Flash data type
String	String
Array	[] = Array
Struct	{} = untyped Object
Query	ArrayCollection
CFC	Class = typed Object (if a matching ActionScript class exists, otherwise the CFC becomes a generic untyped Object (map) in ActionScript)
CFC Date	ActionScript Date
CFC String	ActionScript String
CFC Numeric	ActionScript Numeric
ColdFusion XML Object	ActionScript XML Object

#back to top

© 2014 Adobe Systems Incorporated. All rights reserved.

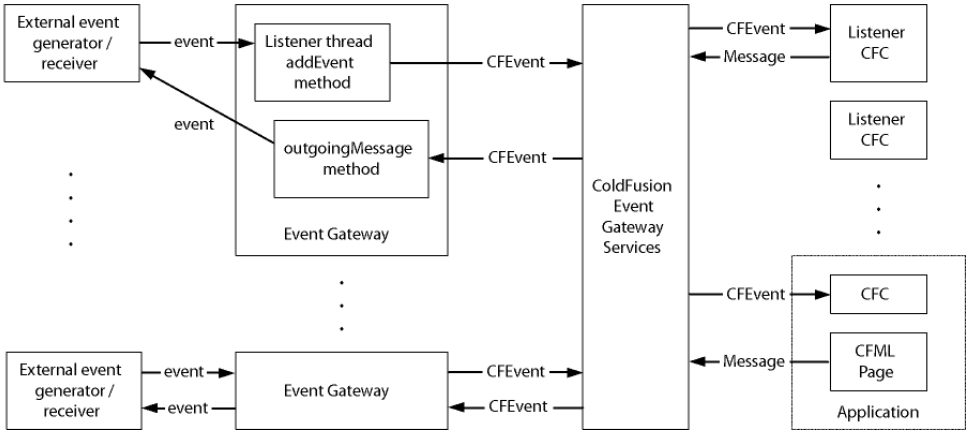
Creating Custom Event Gateways

Adobe ColdFusion lets you create event gateways. Building a gateway requires a knowledge of Java programming, including Java event-handling and thread-handling concepts, and of the technology to which you are providing the gateway, including the types of messages that you handle. This documentation also assumes that you have a thorough knowledge of ColdFusion development concepts and practices, including ColdFusion components (CFCs).

© 2014 Adobe Systems Incorporated. All rights reserved.

Event gateway architecture

A ColdFusion event gateway listens for events and passes them to ColdFusion for handling by the application listener CFC or CFCs. It must implement the coldfusion.eventgateway.Gateway interface, and use the ColdFusion GatewayServices class. The following image expands on the basic event handling architecture diagram to show how a ColdFusion event gateway works:



Receiving messages: The event gateway listener thread receives events from an external event source such as a socket or SMSC server, and calls the GatewayServices `addEvent` method to send a `CFEvent` instance to ColdFusion.

Sending messages: The ColdFusion event gateway service calls the `outgoingMessage` method of the event gateway and passes it a CFEvent instance with the destination and message information. The event gateway forwards the message as appropriate to the external receiver.

The event gateway architecture is not limited to handling messages from external sources, such as SMS devices or IM clients. It can also be used to handle events that are internal to the local system or even the ColdFusion application. Also, a gateway does not have to implement two-way communications.

The sample directory watcher gateway provided with ColdFusion is an example of an internal, one way, gateway. It has a single thread that periodically checks a local directory and sends a message to a CFC when the directory contents change. This gateway does not support outgoing messages. (The code for this gateway is in the gateway/src/examples/watcher directory.)

Another internal gateway, the asynchronous CFML gateway, is provided as part of the ColdFusion product. Unlike most gateways, it does not have a listener thread. Its `outgoingMessage` method gets messages from CFML `Send GatewayMessage` functions, and dispatches them to a CFC `onIncomingMessage` method for handling. This gateway lets ColdFusion support request-free asynchronous processing. For more information on using this gateway, see Using the CFML event gateway for asynchronous CFCs.

© 2014 Adobe Systems Incorporated. All rights reserved.

Event gateway elements

You use the following the elements to create and configure a gateway:

- Gateway interface
- GatewayServices class
- CFEvent class
- GatewayHelper class
- Gateway configuration file
- Gateway development classes

> **Note**
>
> The gateway interfaces and classes, except for the GenericGateway class, are fully documented in Gateway development interfaces and classes in the CFML Reference. All interfaces and classes in this list, including the GenericGateway class, are documented in less detail in the JavaDocs located in the ColdFusion gateways\docs directory. The JavaDocs documentation lacks examples and does not have the detailed usage information that you find in the CFML Reference.

Gateway interface

The ColdFusion event gateway must implement the coldfusion.eventservice.Gateway interface. The following table lists the interface method signatures:

> **Note**
>
> For detailed information on implementing each method, see Building an event gateway. For reference pages for these methods, see Gateway interface in the CFML Reference.

Signature	Description
`void setGatewayID(String id)`	Sets the gateway ID that uniquely identifies the gateway instance. ColdFusion calls this method before it starts the event gateway, even if the gateway class constructor also sets the ID.
`void setCFCListeners(String[]` *listeners*`)`	Identifies the CFCs that listen for incoming messages from the event gateway. The array contains one or more paths to the listener CFCs. ColdFusion calls this method before it starts the event gateway, and if the configuration for a running event gateway changes.
`GatewayHelper getHelper()`	Returns a coldfusion.eventgateway.GatewayHelper class instance, or null. The GatewayHelper class provides event gateway-specific utility methods to CFML applications. ColdFusion calls this method when a ColdFusion application calls the `GetGatewayHelper` function.
`String getGatewayID()`	Returns the gateway ID.

© 2014 Adobe Systems Incorporated. All rights reserved.

`int getStatus()`	Gets the event gateway status. The interface defines the following status constants: `STARTING`, `RUNNING`, `STOPPING`, `STOPPED`, `FAILED`.
`void start()`	Starts the event gateway. Starts at least one thread for processing incoming messages. ColdFusion calls this method when it starts an event gateway.
`void stop()`	Stops the event gateway. Stops the threads and destroys any resources. ColdFusion calls this method when it stops an event gateway.
`void restart()`	Restarts a running event gateway. ColdFusion calls this method when the ColdFusion Administrator restarts a running event gateway.
`String outgoingMessage (coldfusion.eventgateway.CFEvent` *cfmessage*`)`	Handles a message sent by ColdFusion and processes it as needed by the gateway type to send a message. ColdFusion calls this method when the listener method of a listener CFC returns a CFEvent or when a ColdFusion application calls the `SendGatewayMessage` function. The CFML `SendGatewayMessage` function gets the returned String as its return value.

GatewayServices class

The Gateway class uses the coldfusion.eventgateway.GatewayServices class to interact with the ColdFusion event gateway services. This class has the following methods:

Signature	Description
`GatewayServices` getGatewayServices	Static method that returns the GatewayServices object. Gateway code can call this method at any time, if necessary.
`boolean` addEvent*msg*`)`	Sends a `CFEvent` instance to ColdFusion for dispatching to a listener CFC. The event gateway uses this method to send all incoming messages to the application for processing. Returns False if the event is not added to the queue.
`int` getQueueSize	Returns the current size of the ColdFusion event queue. This queue handles all messages for all gateways.
`int` getMaxQueueSize	Returns the maximum size of the ColdFusion event queue, as set in the ColdFusion Administrator.

© 2014 Adobe Systems Incorporated. All rights reserved.

`Logger` getLogger`Logger` getLogger`logfile)`	Returns a ColdFusion Logger object that the event gateway can use to log information in the eventgateway.log log file (the default) or the specified log file. The logfile attribute must be a filename without a filename extension, such as mylogifile. ColdFusion places the file in the ColdFusion logs directory and appends .log to the specified filename. For information on using the logger object, see *Logging events and using log files* in Building an event gateway.

CFEvent class

The Gateway class sends and receives CFEvent instances to communicate with the ColdFusion listener CFC or application. The Gateway notifies ColdFusion of a message by sending a CFEvent instance in a `GatewayServices.addEvent` method. Similarly, the Gateway receives a CFEvent instance when ColdFusion calls the gateway `outgoingMessage` method.
The CFEvent class extends the java.util.Hashtable class and has the following methods to construct the instance and set and get its fields. (In CFML, you treat CFEvent instances as structures.)

Methods	Description
CFEvent`gatewayID)`	CFEvent constructor. The *gatewayID* parameter must be the value that is passed in the gateway constructor or set using the Gateway `setGatewayID` method.
`void` setGatewayType`type)String` getGatewayType	Identifies the type of event gateway, such as SMS. For the sake of consistency, use this name in the Type Name field when you add an event gateway type on the Gateway Types page in the ColdFusion Administrator.
`void` setData`data)Map` getData	The event data; includes the message being passed to or from ColdFusion. The content of the field depends on the event gateway type. The Map keys must be strings.Because ColdFusion is not case sensitive, it converts the Map passed in the `setData` method to a case-insensitive Map. As a result, do not create entries in the data with names that differ only in case.
`void` setOriginatorID`id)String` getOriginatorID	Identifies the originator of an incoming message or the destination of an outgoing message. The value depends on the protocol or event gateway type.
`void` setCFCPath`path)String` getCFCPath	An absolute path to the application listener CFC that processes the event. By default, ColdFusion uses the first path configured for the event gateway instance on the Event Gateways page in the ColdFusion Administrator.

© 2014 Adobe Systems Incorporated. All rights reserved.

`void` setCFCMethod`method)String` getCFCMethod	The method in the listener CFC that ColdFusion calls to process this event. By default, ColdFusion calls the `onIncomingMessage` method. For the sake of consistency, Adobe recommends that any event gateway with a single listener does not override this default. A gateway, such as the ColdFusion XMPP gateway, that uses different listener methods for different message types, uses this method to identify the destination method.
`void` setCFCTimeout`seconds)String` getCFCTimeout	The time-out, in seconds, for the listener CFC to process the event request. When ColdFusion calls the listener CFC to process the event, and the CFC does not process the event in the specified time-out period, ColdFusion terminates the request and logs an error in the application.log file. By default, ColdFusion uses the Timeout Request value set on the Server Settings page in the ColdFusion Administrator.
`String` getGatewayID	The event gateway instance that processes the event. Returns the gateway ID that was set in the CFEvent constructor.

GatewayHelper class

ColdFusion includes a coldfusion.eventgateway.GatewayHelper Java marker interface. You implement this interface to define a class that provides gateway-specific utility methods to the ColdFusion application or listener CFC. For example, an instant messaging event gateway could use a helper class to provide buddy list management methods to the application.

The Gateway class must implement a `getHelper` method that returns the helper class or null (if the gateway does not need such a class).

ColdFusion applications call the `GetGatewayHelper` CFML function, which calls gateway's the `getHelper` method to get an instance of the helper class. The application can then call helper class methods using ColdFusion object dot notation.

The following code defines the SocketHelper class, the gateway helper for the SocketGateway class. It has an empty constructor and two public methods: one returns the socket IDs; the other closes a specified socket. These classes let an application monitor and end session connections.

© 2014 Adobe Systems Incorporated. All rights reserved.

```
public class SocketHelper implements GatewayHelper {
 public SocketHelper() {
 }
 public coldfusion.runtime.Array getSocketIDs () {
  coldfusion.runtime.Array a = new coldfusion.runtime.Array();
  Enumeration e = socketRegistry.elements();
  while (e.hasMoreElements()) {
   a.add(((SocketServerThread)e.nextElement()).getName());
   }
   return a;
  }
 public boolean killSocket (String socketid) {
  try
  {
   ((SocketServerThread)socketRegistry.get(socketid)).socket.close();
   ((SocketServerThread)socketRegistry.get(socketid)).socket = null;
   socketRegistry.remove(socketid);
   return true;
  }
  catch (IOException e) {
   return false;
  }
 }
}
```

Gateway configuration file

Gateways can use a configuration file to specify information that does not change frequently. For example, the ColdFusion SMS event gateway configuration file contains values that include an IP address, port number, system ID, password, and so on.

You can specify a configuration file path for each event gateway instance in the ColdFusion Administrator. ColdFusion passes the file path in the gateway constructor when it instantiates the event gateway. The configuration file format and content handling is up to you. It is the responsibility of the gateway class to parse the file contents and use it meaningfully.

One good way to access and get configuration data is to use the java.util.Properties class. This class takes an ISO8859-1 formatted input stream with one property setting per line. Each property name must be separated from the value by an equal sign (=) or a colon (:), as the following example shows:

```
ip-address=127.0.0.1
port=4445
```

The example SocketGateway event gateway uses this technique to get an optional port number from a configuration file. For an example of reading a properties file and using its data, see the code in Class constructor.

Gateway development classes

ColdFusion provides two classes that you can use as building blocks to develop your event gateway classes. Each corresponds to a different development methodology:

- The coldfusion.eventgateway.GenericGateway class is an abstract class from which you can derive your gateway class.

© 2014 Adobe Systems Incorporated. All rights reserved.

- The EmptyGateway class in the gateway\src\examples directory is a template gateway that you can complete to create your gateway class.

The GenericGateway class

ColdFusion includes a coldfusion.eventgateway.GenericGateway abstract class that implements many of the methods of ColdFusion Gateway interface and provides some convenience methods for use in your gateway class. You can derive your gateway class from this class, which handles the basic mechanics of implementing a gateway, such as the `getGatewayID` and `SetCFCListeners` methods. Your derived class must implement at least the following methods:

- `startGateway` (not `start`)
- `stopGateway` (not `stop`)
- `outgoingMessage`
  Your derived gateway class also must implement the following:

- If you support a configuration file, a constructor that takes a configuration file, and configuration loading routines.
- If you use a gatewayHelper class, the `getHelper` method.
- If the event source status can change asynchronously from the gateway, the `getStatus` method.
  The example JMS gateway is derived from the generic gateway class. The gateway class JavaDocs in the gateway\docs directory provide documentation for this class. (The *CFML Reference* does not document this class.)

The EmptyGateway class

The gateway\src\examples\EmptyGateway.java file contains an event gateway template that you can use as a skeleton for creating your own event gateway. (The gateway directory is in the *cf_root* directory in the server configuration and the *cf_root*\WEB-INF-cfusion directory on J2EE configurations.) This file contains minimal versions of all methods in the coldfusion.eventgateway.Gateway interface. It defines a skeleton listener thread and initializes commonly used Gateway properties. The EmptyGateway source code includes comments that describe the additional information that you must provide to complete an event gateway class.

© 2014 Adobe Systems Incorporated. All rights reserved.

Building an event gateway

To build a Gateway class, you can start with the EmptyGateway.java file as a template. (In the server configuration, this file is located in the *cf_root*/gateway/src/examples/ directory; in the J2EE configuration, the file is in the *cf_root*/WEB-INF/cfusion/gateway/src/examples/ directory.) This file defines a nonfunctional event gateway, but has the basic skeleton code for all Gateway class methods.
Wherever possible, this document uses code based on the sample Socket event gateway to show how to implement event gateway features. (In the server configuration, this file is *cf_root*/gateway/src/examples/socket/SocketGateway.java; in the J2EE configuration, the file is *cf_root*/WEB-INF/cfusion/gateway/src/examples/socket/SocketGateway.java.)

Class constructor

An event gateway can implement any of the following constructors:

- `MyGateway(String gatewayID, String configurationFile)`
- `MyGateway(String gatewayID)`
- `MyGateway()`
  When ColdFusion starts, it calls the constructor for each event gateway instance that you configure in ColdFusion. (ColdFusion also calls the gateway `Start` method after the event gateway is instantiated.). ColdFusion first attempts to use the two-parameter constructor.
  Because each event gateway instance must have a unique ID, ColdFusion provides redundant support for providing the ID. If the event gateway implements only the default constructor, ColdFusion provides the ID by calling the `setGatewayID` method of the event gateway.
  If the event gateway does not implement the two-parameter constructor, it does not get configuration file information from ColdFusion.
  The constructor normally calls the static GatewayServices.getGatewayServices method to access ColdFusion event gateway services. Although you need not make this call, it is a good coding practice.
  A minimal constructor that takes only a gateway ID could look like the following:

```
public MyGateway(String gatewayID) {
 this.gatewayID = gatewayID;
 this.gatewayService = GatewayServices.getGatewayServices();
}
```

The gateway constructor must throw a coldfusion.server.ServiceRuntimeException exception if an error occurs that otherwise cannot be handled. For example, throw this exception if the event gateway requires a configuration file and cannot read the file contents.
If your gateway uses a configuration file, have the constructor load the file, even if the `Start` method also loads the file. Load the file because the constructor does not run in an independent thread, and ColdFusion can display an error in the ColdFusion Administrator of the file fails to load. If the `Start` method, which does run in a separate thread, fails to load the file, ColdFusion logs the error, but it cannot provide immediate feedback in the administrator.
The sample Socket event gateway has a single constructor that takes two parameters. It tries to load a configuration file. If you specify a configuration file in the ColdFusion Administrator, or the file path is invalid, it gets an IO exception. It then uses the default port and logs a message indicating what it did. The following example shows the Gateway constructor code and the `loadProperties` method it uses:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
public SocketGateway(String id, String configpath)
{
 gatewayID = id;
 gatewayService = GatewayServices.getGatewayServices();
 // log things to socket-gateway.log in the CF log directory
 log = gatewayService.getLogger("socket-gateway");
 propsFilePath=configpath;
 try
 {
  FileInputStream propsFile = new FileInputStream(propsFilePath);
  properties.load(propsFile);
  propsFile.close();
  this.loadProperties();
  }
  catch (IOException e)
 {
  // Use default value for port and log the status.
  log.warn("SocketGateway(" + gatewayID + ") Unable to read configuration
   file '" + propsFilePath + "': " + e.toString() + ".Using default port
   " + port + ".", e);
 }
}

private void loadProperties() {
 String tmp = properties.getProperty("port");
 port = Integer.parseInt(tmp);
}
```

Providing Gateway class service and information routines

Several gateway methods perform event gateway configuration services and provide event gateway information. The ColdFusion event gateway services call many of these methods to configure the event gateway by using information stored by the ColdFusion Administrator, and to get access to resources and information that the event gateway services and applications require. Some of these methods can also be useful in event gateway code. The following methods provide these services and information:

- `setCFCListeners`
- `setGatewayID`
- `getHelper`
- `getGatewayID`
- `getStatus`

  ColdFusion calls the `setCFCListeners` method with the CFC or CFCs that are specified in the ColdFusion Administrator when it starts a gateway. ColdFusion also calls the method in a running event gateway when the configuration information changes, so the method must be written to handle such changes. The `setCFCL isteners` method must save the listener information so that the gateway code that dispatches incoming messages to gateway services can use the listener CFCs in `setCFCPath` methods.

  ColdFusion calls the `setGatewayID` method when it starts a gateway. The `getGatewayID` method must return the value set by this method.

  ColdFusion calls the `getHelper` method when an application calls the CFML `GetGatewayHelper` function.

  The following code shows how the SocketGateway class defines these methods. To create a gateway, modify the `getHelper` definition to return the correct class, or to return null if no gateway helper class exists. Most gateways can leave the other method definitions unchanged.

© 2014 Adobe Systems Incorporated. All rights reserved.

```
public void setCFCListeners(String[] listeners) {
 this.listeners = listeners;
}
public GatewayHelper getHelper() {
 // SocketHelper class implements the GatewayHelper interface.
 return new SocketHelper();
}
public void setGatewayID(String id) {
 gatewayID = id;
}
public String getGatewayID() {
 return gatewayID;
}
public int getStatus() {
 return status;
}
```

Starting, stopping, and restarting the event gateway

Because an event gateway uses at least one listener thread, it must have `start`, `stop`, and `restart` methods to control the threads. These methods must also maintain the status variable that the Gateway class `getStatus` meth od checks, and change its value among `STARTING`, `RUNNING`, `STOPPING`, `STOPPED`, and `FAILED`, as appropriate.

The start method

The `start` method initializes the event gateway. It starts one or more listener threads that monitor the event source and respond to any messages it receives from the source.
The `start` method should return within a time-out period that you can configure for each event gateway type in the ColdFusion Administrator. If it does not, the ColdFusion Administrator has a Kill on Startup Timeout option for each gateway type. If you select the option, and a time-out occurs, the ColdFusion starter thread calls an interrupt on the gateway thread to try to kill it, and then exits.

> **Note**
>
> If the `start` method is the listener (for example, in a single-threaded gateway), the method does not return until the gateway stops. Do not set the Kill on Startup Timeout option in the ColdFusion Administrator for such gateways.

If the gateway uses a configuration file, load the configuration from the file. Doing so lets users change the configuration file and restart the gateway without restarting ColdFusion. Also load the configuration file in the constructor; for more information, see Class constructor.
In the SocketGateway class, the `start` method starts an initial thread. (In a single-threaded Gateway, this thread would be the only one.) When the thread starts, it calls a `socketServer` method, which uses the Java ServerSocket class to implement a multi-threaded socket listener and message dispatcher. For more information on the listener, see *Responding to incoming messages* below.

© 2014 Adobe Systems Incorporated. All rights reserved.

```
public void start()
{
 status = STARTING;
 listening=true;
 // Start up event generator thread
 Runnable r = new Runnable()
 {
  public void run()
  {
   socketServer();
  }
 };
 Thread t = new Thread(r);
 t.start();
 status = RUNNING;
}
```

The stop method

The `stop` method performs the event gateway shutdown tasks, including shutting down the listener thread or threads and releasing any resources. The following example shows the SocketGateway `stop` method:

© 2014 Adobe Systems Incorporated. All rights reserved.

```
public void stop()
{
 // Set the status variable to indicate that the server is stopping.
 status = STOPPING;
 // The listening variable is used as a switch to stop listener activity.
 listening=false;
 // Close the listener thread sockets.
 Enumeration e = socketRegistry.elements();
 while (e.hasMoreElements()) {
  try
  {
   ((SocketServerThread)e.nextElement()).socket.close();
  }
  catch (IOException e1)
  {
   // We don't care if a close failed.
   //log.error(e1);
  }
 }
 // Close and release the serverSocket instance that gets requests from the
 // network.
 if (serverSocket != null) {
  try
  {
   serverSocket.close();
  }
  catch (IOException e1)
  {
  }
  //Release the serverSocket.
  serverSocket = null;
 }
 // Shutdown succeeded; set the status variable.
 status = STOPPED;
}
```

The restart method

In most cases, you implement the `restart` method by calling the `stop` method and the `start` method consecutively, but you could be able to optimize this process for some services. The following code shows the SocketGateway class `restart` method:

```
public void restart() {
 stop();
 start();
}
```

Responding to incoming messages

One or more listener threads respond to incoming messages (events). The threads must include code to dispatch the messages to ColdFusion event gateway services, as follows:

© 2014 Adobe Systems Incorporated. All rights reserved.

1. Create a CFEvent instance.
2. Create a Map instance that contains the message and any other event gateway-specific information, and pass it to the CFEvent `setData` method.
3. Call the CFEvent `setOriginator` method to specify the source of the message. (This call is required if the ColdFusion application sends a response.)
4. Call the CFEvent `setGateWayType` method to specify the event gateway type.
5. Set any other CFEvent fields where the default behavior is not appropriate; for example, call the `setCFCPath` method to replace the default listener CFC. (For information on default CFEvent fields, see CFEvent class.)
6. Call the `gatewayService.addEvent` method to dispatch the CFEvent instance to ColdFusion.
7. Handle cases where the event is not added to the event gateway service queue (the `addEvent` method returns False).
   If your application sends any messages to multiple listener CFCs, the gateway must create and configure a CFEvent instance and call the `gatewayService.addEvent` method to send the message to each separate listener CFC. The `setCFCListeners` method of the gateway must make the CFC paths available to the gateway for configuring the CFEvent instances.
   If your ColdFusion server carries a heavy event gateway message load, the ColdFusion event gateway services event queue could reach the maximum value set in the ColdFusion Administrator. When the queue reaches the maximum, the `gatewayService.addEvent` method returns False and fails. Your code can do any of the following:

- Return a message to the sender to indicate that their message was not received.
- Wait until the queue is available by periodically comparing the values returned by the GatewayService `getQueueSize` and `getMaxQueueSize` methods, and retry the `addEvent` method when the queue size is less than the maximum.
- Log the occurrence using the logger returned by the GatewayService `getLogger` method. (For more information, see *Logging events and using log files* below.)
  The SocketGateway class implements the listener using a java.net.ServerSocket class object and SocketServerThread listener threads. (See the SocketGateway source for the SocketServerThread code.) When the listener thread gets a message from the TCP/IP socket, it calls the following `processInput` method to dispatch the message to ColdFusion. This method explicitly sets all required and optional CFEvent fields and sends the event to ColdFusion. If the `addEvent` call fails, it logs the error.

Note

Much of the `processInput` method code supports multiple listener CFCs. A gateway that uses only a single listener CFC, would require only the code in the latter part of this method.

© 2014 Adobe Systems Incorporated. All rights reserved.

```
private void processInput(String theInput, String theKey)
{
 // Convert listeners list to a local array
 // Protect ourselves if the list changes while we are running
 String[] listeners;
 int size = cfcListeners.size();
 if (size > 0)
 {
  // Capture the listeners list
  synchronized (cfcListeners)
  {
   listeners = new String[size];
   cfcListeners.toArray(listeners);
  }
 }
 else
 {
  // Create a dummy list
  listeners = new String[1];
  listeners[0] = null;
 }
 // Broadcast to all the CFC listeners
 // Send one message at a time with different CFC address on them
 for (int i = 0; i < listeners.length; i++)
 {
  String path = listeners[i];
  CFEvent event = new CFEvent(gatewayID);
  Hashtable mydata = new Hashtable();
  mydata.put("MESSAGE", theInput);
  event.setData(mydata);
  event.setGatewayType("SocketGateway");
  event.setOriginatorID(theKey);
  event.setCfcMethod(cfcEntryPoint);
  event.setCfcTimeOut(10);
  if (path != null)
   event.setCfcPath(path);
   boolean sent = gatewayService.addEvent(event);
  if (!sent)
   log.error("SocketGateway(" + gatewayID + ") Unable to place message on
    event queue. Message not sent from " + gatewayID + ", thread " + theKey
    + ".Message was " + theInput);
 }
}
```

Responding to a ColdFusion function or listener CFC

The ColdFusion event gateway services call the `outgoingMessage` method of the gateway to handle messages generated when the listener method of an event gateway application listener CFC returns a message or any CFML code calls a `SendGatewayMessage` function. This method must send the message to the appropriate external resource.

The `outgoingMessage` method parameter is a CFEvent instance, containing the information about the message to send out. The CFEvent `getData` method returns a Map object that contains event gateway-specific information about the message, including any message text. All CFEvent instances received by the outgoingMessage contain information in the Data and GatewayID fields.

© 2014 Adobe Systems Incorporated. All rights reserved.

CFEvent instances returned from listener CFC `onIncomingMessage` methods include the originator ID of the incoming message and other information. However, a gateway that could handle messages from the ColdFusion `SendGatewayMessage` function cannot rely on this information being available, so it is good practice to require that all outgoing messages include the destination ID in the data Map.
The `outgoingMessage` method returns a String value. The CFML `sendGatewayMessage` function returns this value to the ColdFusion application. Indicate the status of the message in the returned string. By convention, ColdFusion event gateway `outgoingMessage` methods return "OK" if they do not encounter errors and do not have additional information (such as a message ID) to return.
Because event messages are asynchronous, a positive return normally does not indicate that the message was successful delivered, only that the `outgoingMessage` method successfully handled the message. In some cases, however, it is possible to make the `outgoingMessage` method at least partially synchronous. The SMS gateway, for example, provides two `outgoingMessage` modes:

- **Asynchronous mode** The `outgoingMessage` method returns when the message is queued internally for delivery to the short message service center (SMSC) of the messaging provider.
- **Synchronous mode** The method does not return until the message is delivered to the SMSC, or an error occurs.This way, an SMS application can get a message ID for later use, such as to compare with a message receipt.

Example outgoingMessage method

The following outgoingMessage method is like the version in the SocketGateway class. It does the following:

1. Gets the contents of a MESSAGE field of the Data Map returned by the{{}} CFEvent class `getData` method.
2. Gets the destination from an `outDestID` field in the data Map.
3. Uses the socket server thread of the destination to write the message.

```
public String outgoingMessage(coldfusion.eventgateway.CFEvent cfmsg) {
 String retcode="ok";
 // Get the table of data returned from the event handler
 Map data = cfmsg.getData();
 String message = (String) data.get("MESSAGE");
 // Find the right socket to write to from the socketRegistry hash table
 // and call the writeoutput method of the socket thread.
 // (Get the destination ID from the data map.)
 if (data.get("outDestID") != null)
  ((SocketServerThread)socketRegistry.get(data.get("outDestID"))).
    writeOutput(message);
 else {
  System.out.println("cannot send outgoing message. OriginatorID is not
   available.");
  retcode="failed";
  }
 return retcode;
}
```

Logging events and using log files

The `GatewayServices.getLogger` method returns an instance of the coldfusion.eventgateway.Logger class that you can use to log messages to a file in the ColdFusion logs directory. (You set this directory on the ColdFusion Administrator Logging Settings page.) The method can take no parameter, or one parameter:

- The default `GatewayServices.getLogger` method uses the eventgateway.log file.
- Optionally, you can specify a log filename, without the .log extension or directory path.

© 2014 Adobe Systems Incorporated. All rights reserved.

The following example tells ColdFusion to log messages from the gateway to the mygateway.log file in the ColdFusion logs directory:

```
coldfusion.eventgateway.Logger log =getGatewayServices().getLogger("mygateway");
```

The Logger class has the following methods, all of which take a message string. The method you use determines severity level that is set in the log message.

- `info`
- `warn`
- `error`
- `fatal`

  You can also pass these methods an exception instance as a second parameter. When you pass an exception, ColdFusion places the exception information in the exception.log file in the ColdFusion logs directory.

  You can use these methods to log any messages that you find appropriate. If you use the default eventgateway.log file, however, remember that all ColdFusion standard gateways us it, and other gateways could use it. As a result, limit the messages that you normally log to this file to infrequent normal occurrences (such as gateway startup and shutdown) or errors for which you want to retain data.

  ColdFusion uses the following format for the message text, so have your application follow this pattern:

```
GatewayType (Gateway ID) Message
```

The SMS event gateway, for example, includes the following exception catching code, which logs a general exception messages and the exception name in the eventgateway.log file, and also (automatically) logs the exception in the exceptions.log file:

```
catch(Exception e)
{
 logger.error("SMSGateway (" + gatewayID + ") Exception while processing
 incoming event: " + e, e);
}
```

Note

When you are developing an event gateway application, you can use the ColdFusion Log viewer to inspect your log files and the standard ColdFusion log files, including the eventgateway.log file and exception.log file. You can use the viewer to filter the display, for example, by selecting different severity levels, or searching for keywords.

© 2014 Adobe Systems Incorporated. All rights reserved.

Deploying an event gateway

To deploy an event gateway, you deploy and event gateway type and configure one or more event gateway instances.

Deploy an event gateway type in ColdFusion

1. Compile your Gateway class and place it in a JAR file along with any other required classes.

> **Note**
>
> The ColdFusion_ class loader includes the gateway \lib directory on its classpath and includes any JAR files that are in that directory on the class path._

1. Place the JAR file in the *cf_root_WEB-INF\cfusion\gateway\lib directory on J2EE configurations or the _cf_root*\gateway\lib directory on server configurations. This directory is on the ColdFusion classpath.
2. Ensure that ColdFusion event gateway services are enabled on the ColdFusion Administrator Data & Services > Event Gateway > Settings page.
3. On the ColdFusion Administrator Data & Services > Event Gateways page, click the Manage Gateway Types button to display the Gateway Types page.
4. On the Add/Edit ColdFusion Event Gateway Types form, enter a type name (for example, SMS for the SMS event gateway), a description, and the full Java class name (for example, coldfusion.eventgateway.sms.SMSGateway for the SMS event gateway). If appropriate, change the Startup Timeout settings from the default values.
5. Click the Add Type button to deploy the event gateway type in ColdFusion.
   The following procedure describes how to configure an event gateway instance that uses the gateway type.

Configure an event gateway instance

1. If you have finished deploying the event gateway type, click the Manage Gateway Instances button; otherwise, select Event Gateways > Gateway Instances in the ColdFusion Administrator.
2. On the Add/Edit ColdFusion Event Gateways Instances form, do the following:
   - Enter the instance name in the Gateway ID field
   - Select the event gateway type that you added from the Gateway Type menu
   - Specify the paths to the listener CFC or CFCs that handle the messages.
   - If the event gateway requires a configuration file, enter the path to the file in Gateway Configuration File field.
   - If you do not want the gateway to start up automatically when ColdFusion starts, change the Startup Mode selection to Manual or Disabled
3. Click the Add Gateway Instance button.
4. In the list of configured instances, click the start button (green triangle) on the entry for the gateway instance to start the instance.

#back to top

© 2014 Adobe Systems Incorporated. All rights reserved.

Using the ColdFusion Extensions for Eclipse

The Adobe ColdFusion Extensions for Eclipse include wizards that help generate code for common tasks and an extension that lets you connect to remote servers from Adobe Flash Builder and Eclipse.
To use the ColdFusion Extensions for Eclipse, you should be familiar with ColdFusion components, as well as accessing and using data in ColdFusion applications. You should also be familiar with Eclipse or Adobe Flash Builder.

© 2014 Adobe Systems Incorporated. All rights reserved.

About the ColdFusion Extensions for Eclipse

To make some common coding tasks easier, the ColdFusion Extensions for Eclipse include the following:

- Eclipse RDS Support plug-in, which lets you access files and data sources on a ColdFusion server.
- ColdFusion/Flex Application wizard, which lets you create master and detail pages in an application to create, read, update, and delete records in a database.
- ColdFusion/Ajax Application wizard, which lets you create master and detail pages that use Ajax elements in an application to create, read, update, and delete records in a database.
- RDS CRUD wizard, which lets you dynamically create a ColdFusion component (CFC) based on a table that is registered in the ColdFusion Administrator on a ColdFusion server
- ActionScript to CFC wizard, which lets you create a CFC based on an ActionScript class file.
- CFC to ActionScript wizard, which lets you create an ActionScript file based on a CFC Value Object
- Services Browser, which lets you browse CFCs, manage a list of web services, and generate the CFML code to invoke a web service.
  For information about installing the ColdFusion Extensions for Eclipse, see *Installing ColdFusion* guide.

© 2014 Adobe Systems Incorporated. All rights reserved.

Eclipse RDS Support

Remote Development Services (RDS) lets you access files and data sources registered in the ColdFusion Administrator on a ColdFusion server. To use Eclipse RDS Support, you must enable RDS when you install ColdFusion. With Eclipse RDS Support, you can use Flash Builder or CFEclipse as your IDE and access ColdFusion files remotely.
Eclipse RDS Support is supported on all ColdFusion server platforms.
Before you install Eclipse RDS Support, you must have the following installed:

- Eclipse 3.1 or later, Flex Builder 2 or later, or Flash Builder
- ColdFusion MX 7.0.1 or later

Configuring RDS

Before using RDS, you must configure ColdFusion servers.

Configure any ColdFusion servers that you want to connect to using RDS

1. In Flash Builder or Eclipse, select Window > Preferences > ColdFusion > RDS Configuration.
2. To configure the default localhost server, select localhost and specify the following:
   - Description
   - Host name (127.0.0.1)
   - Port number (8500 if you are using the built-in web server)
   - Context root, if necessary (For more information about the context root, see *Installing ColdFusion* guid e.)
   - Password, which is the RDS password
3. To specify additional servers, click New, and specify the following:
   - Description, which can be any name you want
   - Host name (IP address or machine name)
   - Port number (8500 if you are using the built-in web server)
   - Context root, if necessary For more information about the context root, see *Installing ColdFusion* guide .
   - Password, which is the RDS password
4. To remove a server definition, select the server and click Remove.
5. To test a connection, select the server and click Test Connection.

> **Note**
>
> If you are using ColdFusion MX 7 or earlier, the message "The RDS server was successfully contacted, but your security credentials were invalid," appears. The message indicates that the password was not validated, even if it is correct. Click OK to close the message.

Once you have configured the RDS connection to your CF servers, you can view the files, folders and data sources on RDS servers. Each RDS server appears as a node in the RDS Fileview and Dataview, with the name you specified when you configured the RDS server.

View files and folders or data sources do the following

1. In Flash Builder, select Window > Other Views. In Eclipse, select Window > Show View > Other.
2. Select RDS.
3. To access the file system on the RDS server, select RDS Fileview.
4. To access data sources on the RDS server, select RDS Dataview.

Using the RDS Fileview

The RDS Fileview lists all the folders and files on the RDS server. You use the navigation buttons as indicated in the

© 2014 Adobe Systems Incorporated. All rights reserved.

following table:

Button	Action
	Refresh the active RDS server.
	Create a file in the currently selected folder.
	Delete the currently selected file.
	Create a folder in the currently selected folder.
	Delete the currently selected folder.

Note

RDS Eclipse Support does not support file operations such as copy and paste, drag and drop, and changing file attributes. However, delete, save, save as, and rename are supported. Also, on ColdFusion servers after ColdFusion 5, the date last modified field does not appear.

To rename a folder or file, right-click the folder or filename.

Using the RDS Dataview

The RDS Dataview lists all the data sources on the RDS server. You use the buttons as indicated in the following table:

Button	Name	Description
	Refresh	Refresh the currently selected item.
	Query Viewer	Opens the RDS Query Viewer.

You can build queries using either the RDS Query Viewer or the Visual Query Builder. The RDS Visual Query Builder is like the ColdFusion Report Builder Query Builder and the HomeSite Query Builder.

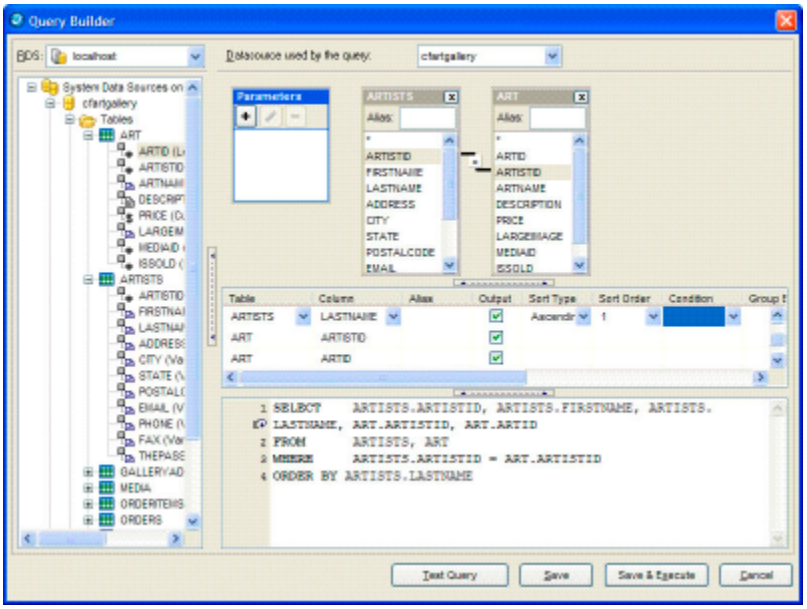
Build and execute a query using the RDS Query Viewer

1. Click the RDS Query Viewer icon on the RDS Dataview tab.The RDS Query Viewer opens in its own tab, which means that if you have other documents open, the RDS Query Viewer has focus.
2. Do one of the following:
   - Enter the SQL, and double-click the field names and table names as appropriate.
   - Click the Visual Query Builder button.
     For more information about using the Visual Query Builder, see *Using Visual Query Builder* below.
3. To try the query, click Execute query.The first 50 records of the result set appear.

© 2014 Adobe Systems Incorporated. All rights reserved.

Using Visual Query Builder

You use the Query Builder to define a SQL statement. The following image shows the Query Builder user interface:



Build a SQL statement using the Table pane and the Properties panel

1. Expand a data source.
2. Double-click the columns to be named in the SELECT statement.As you select columns, the Query Builder creates the SELECT statement in the area at the lower edge of the pane.
3. If you select columns from more than one table, you must specify the column or columns used to join them by dragging a column from one table to the related column in the second table.
4. (Optional) Specify sort order by doing the following:
   a. Locate the column in the Properties panel.
   b. Click in the Sort Type cell of the column you want to sort by.
   c. Specify Ascending or Descending.
   d. (Optional) If you specify multiple sort columns, you specify precedence using the Sort Order cell.
5. (Optional) Specify selection criteria by doing the following:
   a. Locate the column in the Properties panel.
   b. Click in the Condition cell.
   c. Select WHERE.
   Specify WHERE clause criteria in the Criteria cell.

> **Note**
>
> If you specify selection criteria, the Query Builder creates a WHERE clause. To use an INNER JOIN or other advanced selection criteria instead, you must code the SQL manually.

1. (Optional) To specify an aggregate function, GROUP BY, or an expression:
   a. Locate the column in the Properties panel.
   b. Click in the Condition cell.
   c. Select Group By or the aggregate function (such as COUNT).
2. (Optional) To specify SQL manually, type the SQL statement in the SQL pane.

© 2014 Adobe Systems Incorporated. All rights reserved.

> **Note**
>
> You code SQL manually to use an INNER JOIN instead of a WHERE clause, use an OUTER JOIN, or use a database stored procedure.

1. (Optional) To specify the data type of a query parameter:
    a. Click the + button under Parameters.
    b. Enter the name of the parameter.
    c. Select the data type.
2. Review the SELECT statement that displays in the SQL pane, and use the Table and Properties panes to make adjustments, if necessary.
3. (Optional) Click Test Query.
4. Click Save.

© 2014 Adobe Systems Incorporated. All rights reserved.

ColdFusion/Flex Application wizard

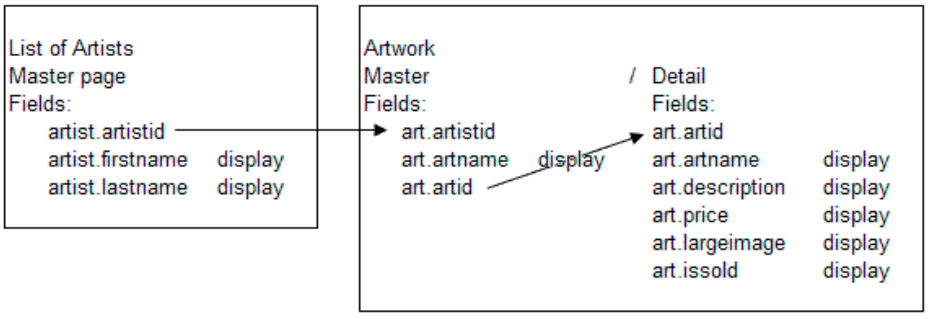
The ColdFusion/Flex Application wizard creates ColdFusion and Flex files for a create, read, update, delete (CRUD) application. You specify the master, detail, and master/detail pages to include in the application, and the relationship between the application's pages. The wizard lets you use Visual Query Builder to generate the SQL statements. For more information about using Visual Query Builder, see *Using Visual Query Builder* in Eclipse RDS Support.

Designing your application

Before starting the ColdFusion/Flex Application wizard, you should determine which pages to include in your application, including the following:

- Whether each page is a master, detail, or master/detail page
- The fields to display in each page
- The fields that connect one page to another
  In the following example, you create an application for an art gallery. The first page lists all the artists that your gallery represents. When a user selects an artist, a page that lists all the works by that artist appears. When the user then selects a work of art, a page that contains details about that piece of art appears. In this example, your application contains the following pages:

- A master page that lists the artists
- A master/detail page in which the master page lists the works of art by the artist selected on the List of Artists master page, and a detail page that contains details about the artwork selected on the Artwork master page.
  You may find it helpful to draw a diagram of the tables and fields that you want to include in your application, including which ones to display in your application, as the following image shows:



Start the ColdFusion/Flex Application wizard

1. Configure your RDS servers. For more information, see *Configuring RDS* in Eclipse RDS Support.
2. In Eclipse or Flash Builder, select File > New > Other.
3. Under ColdFusion Wizards, select ColdFusion/Flex Application wizard, and then click Next.
4. After reading the introductory text, click Next.
5. To load the settings from an application you previously created using the ColdFusion/Flex Application wizard, select the configuration file, and then click Load ColdFusion/Flex Application Wizard Settings.
6. Click Next.
7. Select the RDS server on which you want the application to reside.
8. Specify the data source to use. The data source is configured in the ColdFusion Administrator.Although you specify one default data source at this point, you can access data from other data sources in your application.
9. Click Next.

Specifying form layout

The Form Layout dialog box lets you specify the pages to use in your application. You can create master, detail, or master/detail pages. In your application, you can link master, detail, and master/detail pages as follows:

© 2014 Adobe Systems Incorporated. All rights reserved.

Page type	Can link to
master	mastermaster/detaildetailmaster and detailmaster and master/detail
master/detail	mastermaster/detail

Create a page

1. Click the plus sign (+).
2. In the Name text box, enter the name for the page.
3. Select the page type (master, detail, or master/detail).
4. Click Edit Master Page, Edit Detail page, or Edit Master Section, depending on the type of form you are creating.The Visual Query Builder starts.
5. Use Visual Query Builder to specify the data source, tables, and fields to include in the form, and then click Save to save the query. For more information about using Visual Query Builder, see *Using Visual Query Builder* in Eclipse RDS Support.
6. Repeat steps 1 through 5 for each form in your application.
7. Use the right and left arrows to specify the relationship of the forms in your application. For example, detail forms should appear indented, directly under the related master form in the Navigation Tree panel. You drag and drop items to move them in the tree structure.
8. Click Next.The Project information page appears.
9. Specify the following:
   - The context root, if applicable
   - Whether to include a login page in the application
   - The location of the services-config.xml configuration file that the project should use
   - The web root URL
   - Whether to use an existing or new Flash Builder or Eclipse project
   - The project name and the location of the project if it is new
10. Click Finish.
    The ColdFusion/Flex Application wizard creates the ColdFusion and Flex files that comprise your application. You can test the application by clicking the Run Main button in Flash Builder or Eclipse, or by browsing to the main application page, which is located at http://*<server_name>*:*<port_number>*/*<project_name>*/bin/main.html. You can also manually modify the application files as appropriate for your needs.

Tips for creating applications with the ColdFusion/Flex Application wizard

Although the ColdFusion/Flex Application wizard greatly simplifies creating CRUD applications, keep in mind the following information to ensure that you create the application that you designed.

- To adjust UI elements, open the MXML file in Flash Builder or Eclipse design mode.
- When you create a project that has the same name as a project you previously created, the wizard creates a backup folder that contains the files from the project you previously created.
- If you create a master page and a detail page for a table in which there is no primary key defined, the wizard selects the first field in the database as the key value to represent the row.
- In master pages, link a field to the Parameters box to add type validation to the query by using the `cfqueryparam` tag. Doing this is optional.
- You must select a primary key column in the master form; the wizard chooses the key by default. If you create a master page and do not link it to the id property, you cannot add it to the site tree under another master page.
- Deselect the Display column for fields that your application uses that you do not want to appear in your application.
- Specify the sort order for the field by which to sort data in the page, and specify any other conditions as

© 2014 Adobe Systems Incorporated. All rights reserved.

appropriate.

- Change the labels for fields by clicking the field name in the Label column, and then entering a new field name.
- In a detail page, create a combo box that is populated by dynamic data. To do this, change the value in the Input Control column for the field to use to populate the combo box to be ComboBox, click the Input Lookup Query (sub-select) column in that field, and then use the Visual Query Builder to specify the data to use.
- When you create a detail page, display of the primary key is disabled automatically.
- When you create a detail page, input controls are assigned by default. You can change them from the default values, which appear as follows:
    - Boolean and bit values appear as a check box.
    - Memo and CLOB values appear as a text area.
    - Everything else appears as a text input control.

© 2014 Adobe Systems Incorporated. All rights reserved.

ColdFusion/Ajax Application wizard

The ColdFusion/Ajax Application wizard creates a ColdFusion create, read, update, and delete (CRUD) application that contains Ajax elements. For information about using the wizard, see ColdFusion/Flex Application wizard.
You start the wizard just as you start the ColdFusion/Flex Application wizard, except that you select the ColdFusion/Ajax Application wizard. Unlike the ColdFusion/Flex Application wizard, the ColdFusion/Ajax Application wizard does not generate login screens.

© 2014 Adobe Systems Incorporated. All rights reserved.

ActionScript to CFC wizard

The ActionScript to CFC wizard lets you create a ColdFusion component (CFC) based on an ActionScript class file.

Use the ActionScript to CFC wizard

1. In Flash Builder or Eclipse, go to the Project Navigator.
2. Right-click an ActionScript class file.
3. Select ColdFusion Wizards > Create CFC.
4. Enter the location of the CFC file in the Save to Project Folder text box, or click Browse and select the location.
5. Enter the filename of the CFC in the CFC Filename text box.
6. To replace an existing file, select Overwrite file.
7. To create get and set methods in the CFC, in addition to property definitions, select Generate Get/Set Methods.
8. To specify the property scope, select public or private.
9. Click Finish.

© 2014 Adobe Systems Incorporated. All rights reserved.

CFC to ActionScript wizard

The CFC to ActionScript wizard lets you create an ActionScript file based on a ColdFusion component (CFC) Value Object.

Use the CFC to ActionScript wizard

1. In Flash Builder or Eclipse, go to the Project Navigator.
2. Right-click a CFC Value Object file.
3. Select ColdFusion Wizards > Create AS Class.
4. Enter the location of the ActionScript file in the Save to Project Folder text box, or click Browse and select the location.
5. Enter the class package in the AS Class Package text box.
6. Enter the filename of the ActionScript class file in the AS Class Name text box.
7. To replace an existing file, select Overwrite file.
8. Enter the path to the CFC in the Path to CFC text box.
9. To create get and set methods in the ActionScript Class file, select Generate Get/Set Methods.
10. Click Finish.

© 2014 Adobe Systems Incorporated. All rights reserved.

RDS CRUD wizard

The Remote Development Services (RDS) CRUD Wizard lets you dynamically create a ColdFusion component (CFC) based on a table that is registered in the ColdFusion Administrator on a ColdFusion server. To use the RDS CRUD wizard, you must have the Eclipse RDS Support plug-in installed. (The Eclipse RDS Support plug-in is installed when you install the ColdFusion wizards.)
The RDS CRUD Wizard lets you create the following types of CFCs:

- ActiveRecord style CRUD CFC, which includes all of the properties, get and set methods, and SQL methods in one CFC. The CFC includes the following methods:`init()` or `init(`*primary key value*`)load(`*primary key value*`)save()delete()`
- Bean/DAO style CRUD CFCs, which creates two related CFCs:
  - A Bean CFC, also called a Value Object, which contains the property definitions and get and set methods.
  - The DAO CFC, which contains the following methods:`read(`*primary key value*`)create(`*cfc instance*`)update(`*cfc instance*`)delete(`*cfc instance*`)`
- Data Service assembler CFC, which includes a Bean (also referred to as a Value Object), a DAO CFC, and an assembler CFC. The assembler CFC is required to take advantage of the Flex Data Services feature

Use the RDS CRUD wizard

1. In Flash Builder or Eclipse, go to the RDS Dataview by doing the following:
   a. Select Window > Show View > Other.
   b. Select RDS.
   c. Select RDS Dataview.
2. Right click a table name.
3. Select ColdFusion Wizards > Create CFC.
4. Enter the project folder where you want to save the CFC in the CFC Folder text box.
5. Enter the CFC package in the CFC Package Name text box.
6. (Optional) Select the Primary Key column if a primary key is not defined in the database.
7. (Optional) To specify the primary key column in addition to the other values specified in the CFC, select the Primary Key is Controlled by the User option. If the primary key is automatically generated by the database (the identity field), do not select this option.
8. To replace existing files, select the Overwrite Files If They Already Exist option.
9. Select one of the following CFC Types:
   - Active Record CFC
   - Bean CFC & DAO CFC
   - Flex Data Service Assembler CFCs
10. Enter the names of the CFCs in the appropriate text boxes.
11. To create an ActionScript Value Object:
    a. Select the Create an ActionScript Value Object in Addition to the CFCs option.
    b. Enter the location for the ActionScript Value Object in the AS Folder text box, or click Browse to browse to the location.
    c. To create get and set methods in the ActionScript Class file, select Generate Get/Set Methods.
12. Click Finish.

© 2014 Adobe Systems Incorporated. All rights reserved.

Services Browser

The ColdFusion Services Browser lets you view all of the CFCs and web services on your computer.

Use the Services Browser

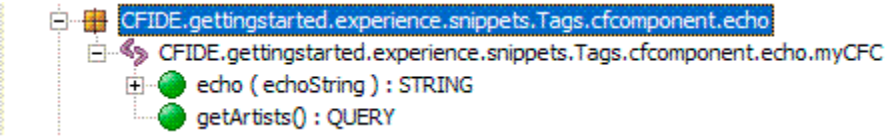
1. In Flash Builder or Eclipse, select Window > Show View > Other.
2. Select ColdFusion > Services Browser.
   The Services Browser can do the following:

- Browse components
- Manage web services

Browsing components

The Service Browser lists the following components:

- Components that the ColdFusion component browser listsThe ColdFusion component browser is located at cf_root/wwwroot/CFIDE/componentutils/componentdoc.cfm.
- Components that are located in any directories specified in the ColdFusion Administrator Mappings page
- Components that are located in any directories specified in the ColdFusion Administrator Custom Tag paths page
  You can restrict the list of CFCs according to whether the functions in a CFC are remote, public, or private. A sample element of the list appears as follows:



The first line of the listing contains the path. The second line includes the name of the CFC. The next two lines contain the names of the functions in the CFC. The function name is followed by any argument, a colon, then the type of the return value. The listing echo(echoString):STRING indicates that the `echo` function has an argument named `echoString`, and that it returns a string. The myCFC CFC appears as follows:

```
<cfcomponent>
 <cffunction name="echo" output="No" returntype="string">
  <cfargument name="echoString" required="Yes">
  <cfreturn "echo: #arguments[1]#">
 </cffunction>

 <cffunction name="getArtists" returntype="query" hint="query the database and
return the
   results.">
  <cfquery name="artists" datasource="cfcodeexplorer">
   select *
   from artists
  </cfquery>
  <cfreturn artists>
 </cffunction>
</cfcomponent>
```

Managing web services

The Services Browser lets you manage a list of web services by adding or deleting WSDL URLs from a list. In

© 2014 Adobe Systems Incorporated. All rights reserved.

addition, when you are editing a ColdFusion file, you can use the Services Browser to generate CFML code to invoke a web service or to create a web service object. Similarly, when you are editing an ActionScript file, you can use the Services Browser to generate ActionScript.
To view the list of web services, click the Show Web Services button in the top right corner of the Services Browser view.

Add a web service to the list

1. Right-click in the Services Browser view.
2. Select Add WSDL.
3. Enter a valid WSDL URL.
4. Click OK.

Delete a web service from the list

1. Right-click in the Services Browser view.
2. Select Delete WSDL.

Invoke a web service in ColdFusion

1. Place your mouse pointer where you want to insert the code.
2. View the list of web services.
3. Highlight a web service or a method in a web service and right-click.
4. Select Insert CFInvoke.
   The code that the Service Browser generates appears in the ColdFusion file. The following is an example of the code that the Service Browser generates:

```
<cfinvoke
  webservice="http://arcweb.esri.com/services/v2/MapImage.wsdl"
  method="convertMapCoordToPixelCoord"
  returnVariable="convertMapCoordToPixelCoord" >
 <cfinvokeargument name="mapCoord" type="" />
 <cfinvokeargument name="viewExtent" type="" />
 <cfinvokeargument name="mapImageSize" type="" />
</cfinvoke>
```

Create a web service object in ColdFusion

1. Place your mouse pointer where you want to insert the code.
2. View the list of web services.
3. Highlight a web service or a method in a web service and right-click.
4. Select Insert CFInvoke.
   The code that the Service Browser generates appears in the ColdFusion file. The following is an example of the code that the Service Browser generates:

```
createObject("webservice",
"http://arcweb.esri.com/services/v2/MapImage.wsdl").convertMapCoordToPixelCoord(mapC
oord, viewExtent, mapImageSize);
```

© 2014 Adobe Systems Incorporated. All rights reserved.