Developing Web Applications with ColdFusion

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

Allaire Corporation

Copyright Notice

Part number: AA-DWACF-RK

Contents

CHAPTER 1

Welcome to ColdFusion

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

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

Contents

Product Features

This release marks a significant milestone in the evolution of ColdFusion as a development system for building scalable Web applications that integrate browser, server, and database technologies.

The focus of our development work has been in four major areas: rapid development, scalable deployment, open integration, and total security. Each of these areas is highlighted below.

Rapid development

ColdFusion 4.0 will continue to enhance the speed of development and ease-of-use that have been the hallmark of the development system from its beginning. ColdFusion 4.0 will increase development productivity by integrating ColdFusion Studio more closely with ColdFusion Server, extending the visual tools, and expanding the functionality of the tag-based server scripting language, CFML.

New Feature Highlights

- **Two-way Visual Programming** — ColdFusion Studio 4.0 includes new, more powerful visual programming tools including a WYSIWYG design mode and enhanced visual database tools.
- **Dynamic State Simulation** — The IDE supports establishing state for pages so developers can preview the interactions between pages that rely on multiple variables.
- **Dynamic Page Quality Assurance** — New tools support validating links, configuring dynamic page previewing and validating CFML grammar in pages.
- **One-step Deployment** — New features extend the site- and page-management features to support flexible deployment of complex applications to multiple servers, making the process of moving from development to deployment simple and straightforward.
- **Site Visualization** — ColdFusion Studio 4.0 supports the ability visualize sites and see how pages are linked to each other across a system.
- **CFScript** — CFML has been extended to support traditional scripting syntax for complex data processing on the server using branching and looping.

Scalable deployment

ColdFusion has already reached a point where it is being used to deliver very large volume sites and applications servicing tens of thousands of users. The 4.0 release provides powerful new features that significantly enhance scalability.

New Feature Highlights

- **Load Balancing** — ColdFusion 4.0 supports native load balancing giving developers the ability to deploy large volume applications in high performance clusters that scale to meet any user demands. (Enterprise Edition only.)
- **High Availability** — ColdFusion 4.0 supports the creation of multi-server clusters with automatic fail-over if any server goes down – providing the infrastructure for deploying large volume high-availability sites. (Enterprise Edition only.)
- **Open State Repository** — State information can be moved out of the registry into a pluggable external data source so servers can be easily configured for clustering and load balancing.
- **Advanced Thread Pooling** — ColdFusion Server offers sophisticated thread pooling using i/o completion ports and tight integration with web server APIs.
- **Integration with NT Performance Monitor** — ColdFusion Server is fully integrated with the NT Performance Monitor for increased manageability and tuning.

Open integration

ColdFusion offers better integration with server systems including mail, web servers and directories than any other IRAD system. In the 4.0 release, the integration has been extended to support Extensible Markup Language (XML) and enterprise technologies.

- **Automatic XML Parsing** — ColdFusion Server supports automatic parsing of XML data into CFML variables and the translation of CFQUERY record sets into XML.
- **Native Database Drivers** — ColdFusion 4.0 supports native database connectivity for Oracle and Sybase. (Enterprise Edition only.)
- **CORBA** – ColdFusion 4.0 extends its integration with component standards by supporting Common Object Request Broker Architecture (CORBA) and possibly Enterprise JavaBeans. (Enterprise Edition only.)
- **ColdFusion Extensions (CFX)** — ColdFusion 4.0 supports the creation of more complex CFXs (formerly called "Custom Tags") making it possible to extend ColdFusion with components created with CFML, C/C++, COM, CORBA, JavaBeans, JavaScript and VBScript.

Total security

ColdFusion currently provides a secure environment for development and deployment. The security features in this release enable a much greater range of flexibility and control over security both for development and deployment.

- **Open Authentication System** — Developers can leverage a wide range of different user authentication systems in their applications from within

ColdFusion including Windows NT security, LDAP directories, and proprietary user and group databases.

- **Advanced Remote Development Security** — The Remote Development Services (RDS) used by ColdFusion Studio allows for user and group security configuration for all resources including files and databases using a configurable backend authentication system that integrates with existing user and group databases.
- **Server Sandbox Deployment** — With the server sandbox, server administrators can control what resources (files, databases and components) an application has access to when it is running on a server. This lets server administrators deploy multiple applications on the same server without creating the risk that one application will access another application's resources.

Learning About Web Development and ColdFusion

Web application development is such a new field and requires such a mix of emerging and established technologies that meeting the documentation needs of ColdFusion users is quite a challenge. The skills required to build and deploy dynamic Web content range across HTML, databases, graphic arts, networking, a slew of scripting and programming languages, and even writing!

We have tried to present information on ColdFusion development and supporting technologies so that you can pursue topics of interest to you and integrate them into your overall learning process.

While it is certainly possible for an individual to master all these skills, the team approach has quickly become the only realistic development model for delivering complex applications and we address issues such as building and maintaining Web projects and working with version source control.

We also include pointers to many resources, both print and online, that provide additional information about ColdFusion and supporting technologies.

New to Web development?

The ColdFusion Markup Language is a tag-based language that integrates with HTML to provide greatly enhanced functionality for Web sites. The skills you are building in HTML and Web site development are a solid foundation for ColdFusion development.

ColdFusion Studio is an easy-to-use HTML editor that offers many powerful features for building and maintaining Web sites. It is also the integrated development environment (IDE) for ColdFusion. That means you can use Studio to learn HTML, to develop and test Web sites, and then to develop dynamic content with CFML.

New to ColdFusion?

Getting Started with ColdFusion presents a quick tour of a ColdFusion application. This book, *Developing Web Applications with ColdFusion*, is the best place to start learning about building ColdFusion applications.

If you want access to experienced ColdFusion developers, you can participate in the Allaire Online Forums, where you can post messages and read replies on all subjects relating to ColdFusion. Check out the Forums at http://forums.allaire.com.

Experienced Web developer?

You'll probably want to get going with your project, so take a look at the chapters on setting up data sources, managing input and output, the application framework, Java forms, and programming variables. *Getting Started with ColdFusion* includes a complete application with lots of working code samples that you can drop in to quickly prototype a project. If you want to integrate COM, CORBA, custom tags, CF API tags, LDAP, CFML scripting, or XML data exchange into your applications, see *Advanced ColdFusion Development*.

Developer Resources

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

Allaire Developer Services	
Resource	**Description**
Allaire Web site www.allaire.com	General information about Allaire products and services.
Technical Support www.allaire.com/support	Allaire offers a wide range of professional support programs. This page explains all of the available options.
Professional Education www.allaire.com/education	Information about classes, on-site training, and online courses offered by Allaire.

Allaire Developer Services (Continued)	
Resource	**Description**
Developer Community www.allaire.com/developer	All of the resources you need to stay on the cutting edge of ColdFusion development, including online discussion groups, Knowledge Base, Component Exchange, Resource Library, technical papers and more.
Allaire Alliance www.allaire.com/partners	The growing network of solution providers, application developers, resellers, and hosting services creating solutions with ColdFusion.

Developing Applications in ColdFusion Studio

ColdFusion Studio is a special version of HomeSite, Allaire's award-winning HTML editor. HomeSite's strengths in Web page creation have been enhanced with powerful tools specifically designed for ColdFusion development.

All of the components of dynamic page creation and site management are accessible from Studio.

- View your data sources.
- Quickly build SQL statements to insert in CFQUERY.
- Access the complete HTML and CFML tag sets from the Tag Chooser.
- Edit code from tag-specific editors or from the Tag Inspector.
- Render pages with internal or external browsers and visually edit page elements in Design view.
- Create projects to group your application pages and support files for easy maintenance and uploading.
- Quickly make global changes to files using extended search and replace.
- Save code blocks for re-use as snippets.
- Build ColdFusion expressions from the complete set of CF functions, constants, operators, and variables available in the Expression Builder.
- Debug application code.
- View your site's structure in the Visualizer.
- Validate HTML and CFML code.
- Verify links for individual files or entire projects.
- Enable version source control of your files for team development.

About ColdFusion Documentation

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

Documentation set

The documentation set contains:

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

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

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

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

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

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

Documentation distribution

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

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

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

Reading online documentation

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

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

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

Using online help in ColdFusion Studio

Studio's innovative online documentation system provides a variety of help options:

- Help References — The complete documentation set is available in HTML
- Dialog Help — Inline help for all HTML and CFML tags in the Tag Chooser and tag editors
- Tag Insight — Opens a selectable list of attributes for the current tag

To set Tag Insight options, go to the Tag Help tab in Options > Settings (F8).

The default display of the online documentation is in the Resources pane for high resolution (1024x768 and up) monitor settings and in the internal for all others.

Full-text search is available for the entire Help References set.

To use full-text search for Help References:

1. Click the Help Search button on the Help toolbar to open the search dialog.
2. Type in the search text and click a search criterion button. Search text is saved in the drop-down list for future use.
3. Set the scope of the search by selecting All References or use CTRL + mouse click to choose individual references. You can search down to the page level.
4. Click Search. The results display in the Resources area. Double-click on a result reference to open the document.

5. Click the Help References button to return to document tree.

6. Click the Results button to return to the last search results.

Tip You can extend the online documentation in Studio by adding your own HTML files. Just copy a folder to the Help directory under the ColdFusion Studio directory. Press F5 to refresh the Help reference list. You can now browse and search these files in the Help References.

Documentation conventions

When reading, please be aware of these formatting cues:

- Code samples, filenames, and URLs are set in a distinct font.
- Notes and tips are identified by bold type in the margin.
- Bulleted lists present options and features.
- Numbered steps indicate procedures.
- Toolbutton icons are generally shown with procedure steps.
- Menu levels are separated by the greater than (>) sign.
- Text for you to type in is set in *italics*.

Contacting Allaire

Corporate headquarters

Allaire Corporation
One Alewife Center
Cambridge, MA 02140

Tel: 617.761.2000 voice
Fax: 617.761.2001 fax

http://www.allaire.com

Technical support

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

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

Postings to the ColdFusion Support Forum (http://forums.allaire.com) can be made at any time.

Sales

Toll Free: 888.939.2545
Tel: 617.761.2100
Fax: 617.761.2101

Email: sales@allaire.com
Web: http://www.allaire.com/store

CHAPTER 2

ColdFusion Fundamentals

This section describes the basic components in a ColdFusion application and outlines the steps in planning and designing your ColdFusion applications.

If you're new to ColdFusion, explore the sample applications described in *Getting Started with ColdFusion* and read the following sections that describe how ColdFusion works.

Contents

ColdFusion Components

ColdFusion applications rely on several core components:

- ColdFusion Server
- ColdFusion Administrator
- ColdFusion Studio
- ColdFusion application pages
- ODBC data sources and other data sources
- ColdFusion Extensions

The ColdFusion Server

The ColdFusion Server listens for requests from the Web server to process ColdFusion application pages. It runs as a service under Windows NT.

The ColdFusion Administrator

You use the Administrator to configure various ColdFusion Server options, including:

- ColdFusion data sources
- Debugging output
- Server settings
- Application security
- Server clustering
- Scheduling page execution
- Directory mapping

See *Administering ColdFusion Server* for details on using the Administrator.

ColdFusion Studio

ColdFusion Studio is the development environment for ColdFusion Server. It offers visual development tools, including dynamic page previews using your Web browser, an interactive debugger, a query builder, an expression builder, project management and source control tools, and many other productivity enhancements.

ColdFusion application pages

Application pages are the functional parts of a ColdFusion application, including the user interface pages and forms that handle data input and format data output. They can contain ColdFusion tags, HTML tags, CFScript, JavaScript, and anything else you

can normally embed in an ordinary HTML page. The default file extension used for ColdFusion application pages is .CFM.

ODBC data sources

ColdFusion applications may interact with any database that supports the ODBC standard.

Other data sources

ColdFusion is not limited to ODBC data sources. You can also retrieve data from OLEDB, native database drivers, directory servers that support the Lightweight Directory Access Protocol (LDAP), mail servers that support the Post Office Protocol (POP), data indexed in Verity collections, and so on.

ColdFusion Extensions

ColdFusion offers an open XML-based framework for extending ColdFusion with new server components and connectivity to enterprise systems using COM, CORBA, C/C++, VBScript, JavaScript, ActiveX, or CFML.

See the Building Cold Fusion Extensions chapter in *Advanced ColdFusion Development* for more information.

Creating a ColdFusion Application

Creating a ColdFusion application involves the following considerations:

- Planning your ColdFusion application
- Building application pages
- Providing data sources
- Implementing security
- Testing and debugging your application

The following sections suggest how you might approach each phase.

Planning your ColdFusion Application

Real-world applications require planning and preparation. Keep in mind that although you can plan your application to the last detail, you should anticipate changes as the development process moves forward.

Application design relies on a number of strategies:

- Researching the problem and strategizing a solution.

- Stating the goals of your application.
- Determining what data is necessary for your application and what output is required to satisfy your goals.
- Deciding whether your application requires user or resource security.
- Planning the overall structure of your application.

Plan the flow of data in your application

Many application designers find it essential to draw a flowchart to help them visualize how their application operates. Building such a chart can help you test ideas you have about how data will be handled by your application, before you start writing pages.

Use an iterative process

If you are just starting out as a ColdFusion developer or don't have a lot of experience building applications, you might want to flesh out your plans as best you can and then start building application pages. Your process will evolve as your application components evolve.

As you test your application pages, your insight into how to proceed will evolve as well. You may find yourself retracing your steps to rebuild parts of your application. A rapid iterative cycle means you build components of your application, test them, revise them, and then move on to the next stage. This approach can help you get off the ground quickly without worrying that every detail of your application has been planned.

Developing ColdFusion Application Pages

When you build a ColdFusion application, you create application pages to capture data and provide output. Application pages can contain ColdFusion Markup Language (CFML) tags, HTML tags, Custom CFML tags, CFScript, JavaScript code, and anything else that an HTML page can contain.

When browser clients such as Netscape Communicator, or Microsoft Internet Explorer, request an ordinary HTML page, the browser interprets the HTML page and renders the output in the browser window. When a browser requests a ColdFusion application page, ColdFusion first processes it and then outputs HTML to the web browser. No client software, components, or plug-ins are needed to open ColdFusion application pages. All the processing occurs on the server.

To retrieve data in a ColdFusion application, you can build ColdFusion pages to capture the data a user enters in a form. Most ColdFusion applications interact with an ODBC data source, but other data sources are also available. You can retrieve data from a variety of sources, such as directory servers (LDAP), mail servers (POP), data indexed in Verity collections, or FTP servers.

ColdFusion application design

A ColdFusion application can consist of dozens of application pages, depending on its complexity. Pages typically perform specific functions, such as providing a user-entry interface, presenting output, or retrieving data.

Design considerations

For Web applications, efficiency is crucial. You'll want to find the best way to balance the need for server-intensive operations like large database queries against the need to display information for your users as quickly as possible. If you are designing an application to run in a corporate intranet, you may not have the bandwidth constraints imposed by designing an application to run over the public Web.

Additional design considerations include planning for code reuse (which includes commenting your code and using clear, descriptive names for variables and data sources) and following other programming and database design conventions.

Security

ColdFusion developers can use the authentication and authorization features to control access to applications based on runtime user security. After setting up the security framework in the ColdFusion Administrator, you can authenticate users against NT domains or LDAP directories.

Scheduling

ColdFusion's ability to cache pages as well as database queries can dramatically affect the performance of your Web applications. In addition, with ColdFusion's page scheduling feature, you can schedule large database operations for off-hours to produce static HTML pages of data for your application.

User entry pages

Application pages can consist of CFML or HTML or both, according to the role they play in your application. You may choose to build forms in straight HTML for users to enter data, or use one of the many Java applet-based ColdFusion form controls (such as a tree control, data grid, or slider) without having to know anything about Java.

It's easy to use forms to collect data from users. And it's just as easy to pass form data to a ColdFusion application page for processing. When a form is submitted, form variables are passed to the application page specified in the form ACTION attribute. Form variables can be referenced to display data, perform queries, or serve in a variety of other operations.

The application page can use the form data to execute some process, whether it's running a query against an ODBC data source or retrieving mail. In a shopping cart application, for example, you could give users a simple way to add items to a virtual shopping basket using a form.

Multipurpose application pages

In addition to pages that perform a data collecting role, you'll probably design pages that simply process data passed to them from a query, form input, or by some other means.

Often, ColdFusion application pages perform some degree of processing in addition to whatever information they might display to the user. For example, you might have a page that displays a table of data retrieved from a database. The ColdFusion code that performs the retrieval can be written in the page header, so that when the page is requested by a browser, ColdFusion runs a query against the specified data source and produces a page dynamically based on user selections or other variables you can create and use.

Providing Data Sources

Any ColdFusion application can make use of any available supported data source. Once a data source has been made available to ColdFusion using the ColdFusion Administrator, interacting with that data source is a simple matter of naming the data source and telling ColdFusion what you want to do. Often, this interaction employs the CFQUERY tag, as the following example shows:

```
<CFQUERY NAME="EmployeeName"
    DATASOURCE="BigCompany">
    SELECT FirstName + " " + LastName
    FROM Employees AS FullName
</CFQUERY>
```

This example is saying: "Give me the list of names from the BigCompany database that are in the FirstName and LastName columns of the Employees table. And while you're at it, combine the FirstName and LastName columns into a new column called FullName." Here, the script passed by CFQUERY is an example of a simple SQL (Structured Query Language) statement. For a quick introduction to SQL, see Chapter 11, "Inserting, Updating, and Deleting Data," on page 141.

Other types of data sources

In addition to traditional ODBC data sources (which can include spreadsheet and text files) your ColdFusion application pages can also retrieve data from a variety of Web data sources:

- Directory servers that support LDAP, such as Netscape Directory Server or Microsoft Active Directory.
- Any standard mail server that supports POP.
- File Transfer Protocol (FTP) servers.

Implementing Security

As you plan and build a ColdFusion application, you'll need to consider security in several ways — securing both your development environment and your application's resources.

ColdFusion Server now supports several levels of Advanced Security:

- **Remote Development Services Security (RDS)** — Developers accessing server resources through ColdFusion Studio can be authenticated before receiving access to protected resources.
- **User security** — Implemented in ColdFusion application pages by the ColdFusion developer, User Security offers runtime user authentication and authorization.
- **Server sandbox security** — Controlled by the ColdFusion administrator of a hosted site, offers runtime security based on directory access at hosted sites (ColdFusion Enterprise only).
- **Administrator security** — Individual administrative operations can be secured against unauthorized access.

See *Administering ColdFusion Server* for information on using the Administrator to set up security rules and policies, as well as RDS security for developers working in ColdFusion Studio.

In the User Security chapter of the *Advanced ColdFusion Development* book, you'll find information and examples on how developers can implement user runtime security using CFML tags and functions.

Note Advanced security is not currently supported in ColdFusion Server for Solaris.

Load Balancing

Cold Fusion offers powerful new features that significantly enhance scalability, including load balancing and support for multi-server clusters with automatic fail-over for large volume applications, and advanced thread pooling. Increased performance monitoring features enable you to manage and tune your servers more easily.

See the Clustering and Load Balancing chapter of *Administering ColdFusion Server*for information on load balancing features.

Testing and Debugging your Application

There are several ways to debug ColdFusion applications. ColdFusion Studio offers interactive debugging tools. You can also retrieve debugging output when the ColdFusion Server processes your application pages.

Using the interactive debugger

Developers working in ColdFusion Studio can use the interactive debugger to set breakpoints and watches, evaluate expressions and variables, step through lines of code, and investigate the stack.

See Chapter 7, "Debugging and Troubleshooting," on page 87 for instructions on setting up and running the interactive debugger in ColdFusion Studio.

Outputting the server's debug information

As you build your ColdFusion application pages, you can test pages by simply opening them in a browser. There is no need to compile or link your pages. You can make a tiny change and see the results of your change immediately by simply opening the page in your browser. Most ColdFusion developers run ColdFusion and a Web server locally, on their own computers, and test applications by editing and viewing or running pages side-by-side. Once your application is ready, you can very easily deploy your pages to a remote server.

ColdFusion provides several debugging options to help you troubleshoot your application. For every ColdFusion transaction – that is, every time a browser requests a ColdFusion page – debugging data can be viewed that provides information about the operation to help you track down problems and coding errors. With debugging activated, this information is displayed in your Web browser at the bottom of every application page.

See *Administering ColdFusion Server* for details on setting the debug output options in the Administrator.

CHAPTER 3

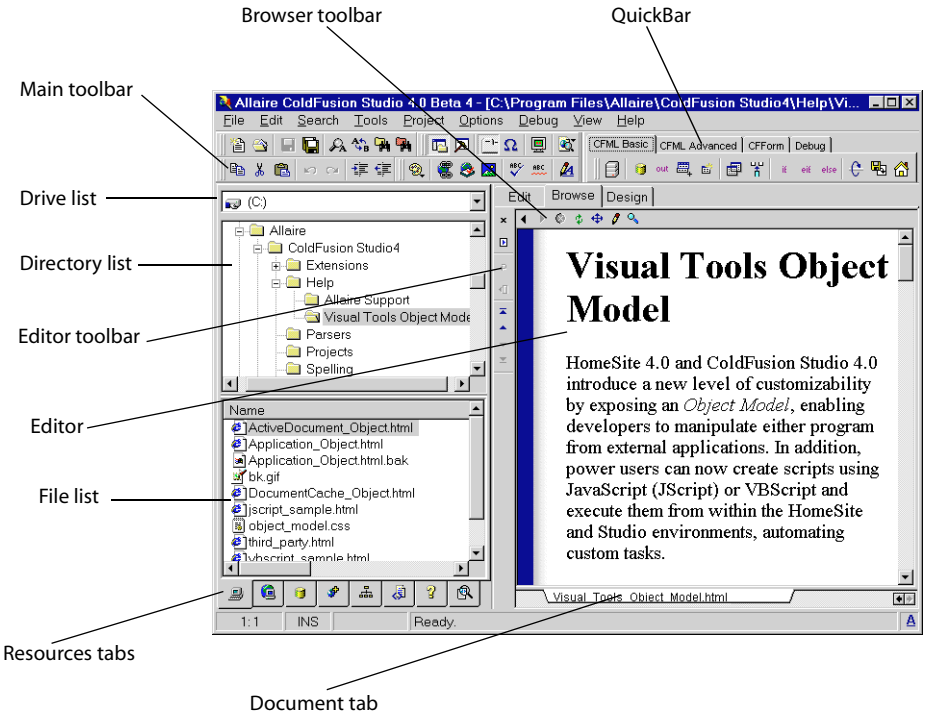
ColdFusion Studio Quick Start

This chapter introduces ColdFusion Studio, the integrated development environment (IDE) for the ColdFusion Development System. Studio combines powerful visual programming and database tools for both individual and team development.

Contents

Exploring the Studio Workspace

Studio offers a highly customizable interface; you can set it up to suit your work style. Nearly all of the Workspace elements are movable, so you can float panes and toolbars and dock them in new locations. The main workspace areas are shown below.



Using context menus

You can right-click (or left-click, for you southpaws) in any of the areas shown above to open a context-sensitive command menu.

The Resources Area

All the resources needed for development work in Studio are accessible from this area. You can click on the top border and drag the pane to float it or dock it in another part of the workspace. To resize the Resources area, you can drag the side border and the divider between the top and bottom panes. Select Options > Resource Tab Caption to change tab display. Click F10 to toggle the display.

The tabs at the bottom of this pane control display for:

- **Files** — The top pane displays the local and network directory structure, the bottom pane lists the files in the selected folder. Double-click a file to open it.
- **Remote** — Add FTP and RDS servers for remote development work.
- **DB** — Configure ColdFusion servers, view data sources, and open the Query Builder.
- **Projects** — Create projects to store and manage application pages and supporting files. You can add projects to your version source control system.
- **Site View** — Displays a graphic representation of links in the current document.
- **Snippets** — Store code blocks for re-use. Snippets can be shared with other users.
- **Help** — View ColdFusion documentation and other online resources. You can edit existing help files and add new HTML files.
- **Tag Inspector** — The top pane displays the Tag Tree, a customizable view of document hierarchies. The bottom pane displays the Tag Inspector, which gives you an interactive "property sheets" mode for writing and editing tags.

You can toggle the Resources pane display (F10) and adjust the size of the panes by dragging their borders. Right-click on a Resources tab to select display options.

The Editor pane

Click on a tab at the top of the pane to move among these modes:

- **Edit** — Create new documents, write, edit, and test code.
- **Browse** — View the current page in the internal browser.
- **Design** — Dynamically work with page elements, such as text blocks, tables, forms, and images. Changes are reflected in the HTML code.

Creating HTML Pages with Templates and Wizards

Select File > New to open an extensive selection of templates and wizards to help you get started building a site or composing page elements such as styles, frames, and tables.

Tip Use the File > Save As Template command to add the current document to the File > New > Custom tab. You can then use it as a starting point for new documents.

HTML Wizards

Many of the wizards listed here are also available on the HTML toolbars.

- **Quick Start** — Build a page framework by entering meta information.

- **Tables** — Define a table structure and properties and enter attributes for individual cells.
- **Frames** — Design a frameset, make selections for frame layout and appearance, and specify a source URL for the frame content.
- **Open Browser window** — Generates code for the JavaScript openWin function based on the specified URL and window attributes. Create links to display text and images in a custom browser window. The window display is browser-dependent and should be tested.
- **Multimedia** — Two wizards make it easy to synchronize RealAudio files with HTML page display and to embed RealAudio controls on your pages.

Dynamic HTML Wizards

Dynamic HTML wizards generate styles and JavaScript to create an outline or slide show. Click the DHTML tab on the File > New dialog to select a wizard:

- **Outline** — Create a multi-level outline by typing in or pasting the outline text and defining properties for the display. The JavaScript code for the outline is inserted in the document head.
- **Slide Show** — Build slide presentations that run in a browser. All the features of desktop presentation products – layout and duration control, transition options, and integrated images and text – are available.

Style Editor

This mini-application provides a complete interface for designing and previewing styles. You have a number of options for working in the editor:

- Click File > New and select StyleEd from the HTML tab
- Right-click in a <STYLE> block and select Edit Style Block on the menu
- Right-click on a link to a style sheet and select Edit Linked StyleSheet
- Right click on a file in the file list with the.css extension and select Edit StyleSheet

Creating Applications with Templates and Wizards

The File > New dialog also contains a collection of helpful wizards for writing templates. To use wizards that require data sources, you must first establish a connection to a Cold Fusion server. Click the Remote Resource tab, right-click in the server pane, and add an RDS server. You can then double-click on a server name in the Server pane to connect the data source.

Click the CFML tab to select a wizard:

- **Custom Tag** — Builds a parameter-checking framework for creating custom tags in CFML.
- **Data Entry** — Creates a data entry application for a selected data source. Specify a table and the fields you want to include. The wizard generates an entry form and an application page that inserts the data when the form is submitted.
- **Data Drill-Down** — Produces an application that searches the selected data sources and returns results and detail pages.
- **Exception Handling** — Generates a TRY-CATCH framework in which you can enter code for a CFTRY tag. You can optionally enable handling of database, template, included missing files, object, security, application-defined, and unexpected internal failures.
- **Table to JS** — Generates a JavaScript object from the contents of a table in a registered ColdFusion data source.
- **LDAP** — Enter LDAP server information and directory entries to create an application to view and edit directories and to submit a directory view form.
- **Library** — Builds access to a file library for users.
- **Mailing List** — Builds an email application from the selected data source of email addresses.
- **Record Viewer** — Generates an application to manage, preview, and edit selected records in a table.
- **Tree control** — Produces a Java tree control from selected data sources.
- **Verity** — Builds an application based on selected search criteria to generate an indexed collection of the specified data sources. This wizard can be used in conjunction with the Library Wizard to index library files.

Application Development Tools in Studio

This section gives a brief description of the major development tools.

Tools

You can try these features on the Tools menu to get a sense of what Studio has to offer as an IDE:

- **Tag Chooser** — A repository of HTML, CFML, HDML, VTML, and Custom tags. The embedded help window provides syntax and usage information for the selected tag.
- **Expression Builder** — Insert CFML expression elements (functions, constants, operators, variables) directly or double-click to build complex expressions in the expressions pane.
- **SQL Builder** — View database schema and construct and test SQL operations for insertion in templates.
- **Spell Check** — Use the integrated spell checker or optionally use the Microsoft Word spell checker if it is installed.
- **Code Validator** — Check the syntax of the current tag or an entire document. Open the Options > Settings dialog (F8) and click the Validation tab to enable validation and to specify CFML and HTML versions.
- **Document Weight** — Presents a list of the current document's dependencies (links to Web resources) calculates approximate download times for common modem types.
- **Verify Links** — Tests the links in the current document and returns the Web server error message for unsuccessful links.

Debugger

The Debugger lets you map ColdFusion servers and debug CFML code using an extensive tool set on the Debug menu and toolbar. See the Keyboard Shortcuts section below for a list of Debug shortcuts.

For more information, see the Debugging and Troubleshooting chapter.

Search

The Search menu provides some powerful features for editing files:

- **Extended Find and Replace** — Works across open documents or can recursively search folders. Support for regular expressions enables sophisticated search criteria.
- **Replace Special Characters** — Change ASCII extended characters to equivalent HTML and back. To set auto-conversion of extended characters, open the Options > Settings > Files tab.
- **Replace Double-Spacing with Single-Spacing** — Quickly re-format pages.

Browsers

You can preview your code in a number of ways:

- If Internet Explorer is detected during the Studio installation, you are prompted to use it as the internal browser. Open the Options > Settings > Browse tab to add it later and to set Web server mappings for processing pages.
- The Studio installation also auto-detects a number of common browsers if they are installed on your system. Click Options > Configure Internal Browsers to review and edit the list.

Customizing the Workspace

A development tool needs to provide a productive interface for its users, and the best way to do that is to give users the greatest possible flexibility to set up the interface in a way that works for them. Here are some options:

- Resize the Resources and Results panes by dragging their borders.
- Right-click on Resources tab to change the display.
- Drag toolbars to any location or dock them vertically on the workspace borders.
- Right-click in the Resources Directories pane to access Windows Explorer commands.
- Right-click in the Resources Files pane to set file and display options.
- Right-click in the main toolbar or the QuickBar to set, add, and remove toolbars.
- Open the Options > Customize dialog (SHIFT + F8) to change toolbar layout, keyboard shortcuts, and to add shortcut keys for snippets.
- Open the Options dialog (F8) to explore a wide range of settings for tags, documents, help, tools, server mappings, and the internal browser.

Keyboard Shortcuts

Studio offers a full set of keyboard commands for file management and document editing. A section at the end of the list contains shortcuts for the Debugger. You can

change shortcut keys and add new ones by opening the Options > Customize dialog and selecting the Keyboard Shortcuts tab.

File and Document Keyboard Commands	
Command	**Key**
File > Open	Ctrl+O
File > Close	Ctrl+W
File > Close All	Shift+Ctrl+W
File > New	Ctrl+N
File > Save	Ctrl+S
File > Save As	Shift+Ctrl+S
File > Print	Ctrl+P
Edit > Cut	Ctrl+X
Edit > Copy	Ctrl+C
Edit > Paste	Ctrl+V
Edit > Undo	Ctrl+Z
Edit > Redo	Shift+Ctrl+Z
Delete Line	Ctrl+Y
Delete to end of line	Ctrl+Del
Delete previous word	Ctrl+Backspace
Edit > Select All	Ctrl+A
Edit > Goto line	Ctrl+G
Goto previous start tag	Ctrl+[
Goto next start tag	Ctrl+]
Edit current tag	Ctrl+F4
Edit > Set Bookmark	Ctrl+K
Edit > Goto Next Bookmark	Shift+Ctrl+K
Insert expanded code or open templates list	Ctrl+J
Edit > Indent	Shift+Ctrl+.

File and Document Keyboard Commands (Continued)	
Command	**Key**
Edit > Unindent	Shift+Ctrl+,
Search > Find	Ctrl+F
Search > Replace	Ctrl+R
Search > Find Next	F3
Search > Extended Find	Shift+Ctrl+F
Search > Extended Replace	Shift+Ctrl+R
Find matching tag	Ctrl+M
Tools > Open Tag Chooser	Ctrl+E
Tools > Spell Check	F7
Spell check all	Shift+F7
Tools > Mark Spelling Errors	Ctrl+F7
Tools > Validate Document	Shift+F6
Tools > Validate Current Tag	F6
View > Full Screen	F10
View > Resources Tab	F9
Goto previous document	Shift+Ctrl+Tab
Goto next document	Ctrl+Tab
Open current document in external browser	F11
Open current document in DreamWeaver	Ctrl+D
Toggle browse mode	F12
Open Tag Inspector	F4
Toggle > Special Characters	Shift+Ctrl+X
Toggle Tag Insight	Shift+F2
Toggle Tag Tips	F2
Toggle QuickBar	Ctrl+H
Toggle Results pane	Shift+Ctrl+L

File and Document Keyboard Commands (Continued)	
Command	**Key**
Options > Settings	F8
Options > Customize	Shift+F8
Open Anchor dialog	Shift+Ctrl+A
Insert Bold tag	Ctrl+B
Insert BR tag	Shift+Ctrl+B
Insert BR tag and new line	Ctrl+Enter
Insert Center tag	Ctrl+Q
Insert Comment tag	Shift+Ctrl+M
Open IMG dialog	Shift+Ctrl+I
Insert Italic tag	Ctrl+I
Insert non-breaking space	Shift+Ctrl+Space
Insert Paragraph tag	Shift+Ctrl+P
Insert Underline tag	Ctrl+U
Insert Start brackets <>	Ctrl+,
Insert End bracket </>	Ctrl+.

Debugger Keyboard Commands	
Command	**Key**
Start/Continue	Ctrl + F5
Start - No debugging	Ctrl + Alt + F5
End	Alt + F5
Restart	Ctrl + Shift + F5
Step Into	Ctrl + F8
Step Over	Ctrl + F9
Run To Cursor	Ctrl + F11

Debugger Keyboard Commands (Continued)	
Command	**Key**
Variables	Alt + Q
Watches	Alt + W
Recordsets	Alt + R
Stack	Alt + K
Output	Alt +P
Breakpoints	ALT + B
Toggle Breakpoint	Alt + X
Clear All Breakpoints	Alt + F6
Debug Settings	Alt + Y
Development Mappings	Alt + M

CHAPTER 4

Creating and Manipulating Variables

In ColdFusion, a variable is, very simply, a parameter that is assigned a value. You can use variables in ColdFusion applications in many different ways. You use variables in ColdFusion forms, for example, to hold data submitted in an application. Variables are essential in handling form inputs, for passing data from each field in a form to the application's ACTION page. These variables are also called dynamic parameters.

This section describes how to use ColdFusion variables, how to distinguish between the various types of variables, and how to create and delete ColdFusion variables.

Contents

Creating and Using Variables

There are many reasons why you create variables in a ColdFusion page. To create a variable, you name it and assign a value to it.

```
<CFSET FirstName="Jack">
```

In this simple example, you use the CFSET tag to initialize the value. The scope of this variable is *local*, or within the application page where it is created.

When creating an array to store information, for example, you also use the CFSET tag to define a variable that contains the array:

```
<CFSET myarray=ArrayNew(1)>
```

You then use CFSET to add data to array elements:

```
<CFSET myarray[1]="January">
<CFSET myarray[2]="February">
...
```

For information about creating arrays in ColdFusion, see *Advanced ColdFusion Development*

Another example is a CGI variable, which can be referenced to provide information about a particular user. You could determine a user's browser type by using the following CGI variable:

```
<CFOUTPUT>
    #CGI.HTTP_USER_AGENT#
</CFOUTPUT>
```

Using CFSET to create variables

The CFSET tag supports the creation of variables and the manipulation of variable values. You can use CFSET anywhere in an application page. The variable that is created can be used anywhere in the page after the CFSET tag.

If no variable prefix is supplied, the variables you create with CFSET are *local* variables, meaning they are available only on the page where they were created and pages included in it.

The CFSET tag can use a static value, a dynamic parameter, or an expression to create the variable.

The following shows the syntax of the CFSET tag:

```
<CFSET VariableName = Value, Parameter, or Expression>
```

Example: Static values

To create a variable called *UserName* based on a static string, use this syntax:

```
<CFSET UserName="Joe Doe">
```

Use quotation marks when the variable's value is a text string.

To create a variable *UserNumber* with a static numeric value, use this syntax:

```
<CFSET UserNumber=26>
```

Note ColdFusion variables are typeless, which means you do not have to identify the type of data they contain, such as text or numbers. For more information on typeless expressions, see the Functions and Expressions chapter in *Advanced ColdFusion Development*.

Example: Dynamic parameters

To create a variable *CurrentUser_ID* based on the column name *User_ID* in a database query (*GetUserID)*, which returns a numeric value, use this syntax:

```
<CFSET CurrentUser_ID=GetUserID.User_ID>
```

To create a variable *UserDescription* with a string that combines a dynamic parameter and text, use this syntax:

```
<CFSET UserDescription="#UserName# is a wonderful person.">
```

Example: Expressions

You can also use CFSET to create variables based on expressions. To create a variable called *TotalValue* based on a mathematical expression, use this syntax:

```
<CFSET TotalValue=2 * (4 + 5)>
```

To create a variable that combines strings and expressions, use an ampersand (&) sign:

```
<CFSET Pay="John's take home pay is" & (TotalValue - 1000)>
```

For more information, see the Functions and Expressions chapter in the *Advanced ColdFusion Development* book.

Displaying variables in a page

To display the variable that has been set for a specific user, enclose the variable to be evaluated inside CFOUTPUT tags, as in the following code:

```
<CFOUTPUT>
    Your favorite color is #Client.FavoriteColor#.
</CFOUTPUT>
```

Inside CFOUTPUT tags, always enclose variable names in pound signs (#). This signals that the variable name needs to be evaluated as a dynamic parameter. This way, ColdFusion outputs the variable's value and not the variable name itself.

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 IsDefined function. For example, the following code checks to see if a Form variable named Order_ID exists:

```
<CFIF Not IsDefined("FORM.Order_ID")>
    <CFLOCATION URL="previous_page.cfm">
</CFIF>
```

See the *CFML Language Reference*for more information on the IsDefined function.

Troubleshooting

If you attempt to evaluate a variable that has not been defined, ColdFusion will not be able to process the page. To help diagnose such problems, use the interactive debugger in ColdFusion Studio or turn Debugging on in the ColdFusion Administrator. The Administrator debugging information shows which variables are being passed to your application pages.

See the Debugging and Troubleshooting chapter for information on catching and fixing errors.

Creating Default Variables with CFPARAM

Another way to create a variable is to test for its existence and supply a default value if the variable does not already exist. The following example shows how to use the CFPARAM tag to check for the existence of an optional client variable and to set a default value if the variable does not already exist:

```
<CFPARAM NAME="Client.FavoriteColor" DEFAULT="Red">
```

Using CFPARAM

The following shows the syntax of the CFPARAM tag:

```
<CFPARAM NAME="VariableName" DEFAULT="DefaultValue">
```

There are two ways to use the CFPARAM tag, depending on how you want the validation test to proceed.

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

Example: Testing for variables

Using CFPARAM with the NAME variable is a 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 to debug.

For example, the following series of CFPARAM tags indicates 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.

Example: Setting default values

In this example, CFPARAM is used to see if optional variables exist. If it does, processing continues. If it does not exist, it is created and set to the DEFAULT value.

```
<CFPARAM NAME="Cookie.SearchString" DEFAULT="temple">

<CFPARAM NAME="Client.Color" DEFAULT="Grey">

<CFPARAM NAME="ShowExtraInfo" DEFAULT="No">
```

You can also use CFPARAM to set default values for URL and Form variables, instead of using conditional logic.

Naming and Scoping Variables

This section describes how to name variables, lists the order in which ColdFusion evaluates them, and offers guidelines for using variables in your applications.

Variable names

When naming ColdFusion variables and form fields, keep these guidelines in mind:

- Variables must begin with a letter, which can be followed by any number of letters, numbers, or the underscore character.
- Variables must be all one word.
- Do not use spaces or special characters in variable names.
  For example, UserName_1, UserName2, and User_Name are valid, but 1stUser, WhatAName!, and User-Name are not.
- For field names and variables, use descriptive names, not abbreviations. It will be much easier for others to read your code, and for you to remember how it works yourself, when you revisit it months later.
- Note that queries and variables cannot have the same name in the same ColdFusion application page.
- Although ColdFusion variables are not case-sensitive, keep capitalization consistent in order to keep your code consistent.
- In writing queries and forms, match your form field names with the corresponding database field name.

Qualifying, or scoping, variable references

ColdFusion distinguishes between identically named parameters from different sources with a specific prefix for each source or "scope." For example, to specify a variable called State that is passed in a form submission, you would use Form.State. To specify a variable named State passed in a URL, you would use URL.State.

You don't need to use the prefix unless two variables in different scopes have the same name. However, for readability and processing speed, it is a good idea to use prefixes. For example, the variable "Form.lastname" is far more self-evident than a variable called "lastname."

The following chart shows the prefixes for each variable type:

Variable Prefix	
Type	**Reference**
Queries	*QueryName.variablename*
Local	Variables.*variablename*
URL Parameters	URL.*variablename*
Form Fields	Form.*variablename*
Client	Client.*variablename*
Server	Server.*variablename*
Session	Session.*variablename*
Application	Application.*variablename*
HTTP Cookies	Cookie.*variablename*
CGI Environment	CGI.*variablename*

Tip Always use the prefix for application and session variables.

Performance and scoping

You can improve performance by always qualifying your variables with the proper scope. Adding variable scopes improves processing speed but the trade-off is that it may not be so easy to reuse the same code in other applications or pages.

In the following example, both forms of the following variable are permitted. However, the example that includes a scoping prefix will be evaluated more quickly than the unscoped example:

```
<CFOUTPUT>
        #Client.fullname#
        #fullname#
</CFOUTPUT>
```

How ColdFusion looks up variables

When scoping isn't used, that is, when you don't include a variable prefix, like "Form.myformvar," ColdFusion attempts to find variables in the following order:

1. Local variables created using CFSET and CFQUERY
2. CGI variables
3. File variables
4. URL variables
5. Form variables
6. Cookie variables
7. Client variables

Note ColdFusion does not attempt to automatically find Application and Session variables. You must use prefixes with these variables.

See Kinds of Variables for information on each type of ColdFusion variable. For information on File variables, see Chapter 13, "Managing Files on the Server," on page 197.

Using pound signs

Pound signs are required around variables in strings and when variables are used as arguments for parameters in ColdFusion tags, such as CFOUTPUT, CFMAIL, and CFQUERY, and when outputting their values.

To output the value of a variable, rather than its name, you surround the variable name with pound signs and place the name between <CFOUTPUT> and </CFOUTPUT> tags.

Here are some guidelines for using pound signs with variables and expressions:

- In CFML pound signs are used to distinguish expressions from plain text.
- In CFOUTPUT and CFQUERY tags, enclose variables and functions in pound signs:

  ```
  <CFOUTPUT>The value is #Form.MyTextField#.</CFOUTPUT>
  ```

  ```
  <CFOUTPUT>The name is #FirstName# #LastName#.</CFOUTPUT>
  ```

  ```
  <CFOUTPUT>Cos(0) is #Cos(0)#</CFOUTPUT>
  ```

  In this example, the SQL statement calls for single quotes to enclose a text string, the value represented by the form variable #FORM.LastName#.

  ```
  <CFQUERY NAME="Search" DATASOURCE="Company">
  ```

```
Select * From Employees
Where LastName='#FORM.LastName#'
</CFQUERY>
```

Note that pound signs are necessary only where you need to distinguish expressions from text, for example, when variables are embedded in text strings:

```
<CFSET A="Hello, #name#">
```

- In CFSET statements, do not overuse pound signs. For example, do not use `<CFSET x=#Cos(0)#+1>`; instead, use `<CFSET x=Cos(0)+1>`.
- Similarly, `<CFSET FullName=FirstName & " " & LastName>` is the same thing as `<CFSET FullName="#FirstName# #LastName#">`.
- Pound signs are required when variables are used inside ColdFusion tags such as CFOUTPUT, CFMAIL, and CFQUERY.

  ```
  <CFOUTPUT>Your favorite color is #Client.FavoriteColor#.
  </CFOUTPUT>
  ```

- Because pound signs serve as formatting codes in ColdFusion, you need to take special measures when including pound signs in a CFOUTPUT section. To include a pound sign that is not used as a field delimiter, use two consecutive pound signs (##).

For more information on using pound signs in ColdFusion pages, see the Functions and Expressions chapter in the *Advanced ColdFusion Development* book.

Passing Variables to Pages with URLs and Forms

In some cases, you will want to make a parameter that exists in one page available to another page. There are many ways to pass a variable between two pages: in a URL, in a form, or using browser cookies and client variables.

Passing parameters with a URL

You can pass dynamic parameters from one page to another by appending them to the URL reference to the target page. Separate the parameter from the URL filename address with a question mark (?). Parameters are passed by appending the name of the variable you are passing and its associated value.

The scope of these variables is the target page of the hyperlink that carries the URL variables.

Example

In this example, the hyperlink passes a variable named "user_id" with a value of 5 and a variable named "color" with a value set to the expression #mycolor# to the `example.cfm` page:

```
<A HREF="example.cfm?user_id=5&color=#mycolor#">
```

In the target page, `example.cfm`, you can refer to these variables as *URL.user_id* and *URL.color*.

```
<CFOUTPUT>
Your user ID is #URL.user_id# and
your favorite color is #URL.mycolor#.
</CFOUTPUT>
```

Formatting issues

When formatting URL parameter values, keep the following formatting issues in mind:

- Use a ? to separate the URL address from the query string you're passing.
- Use an ampersand & to separate variable pairs.
- Do not use spaces. Some browsers cut off the URL when they detect a space.
- Do not use special characters, such as & ?, . and #.

If you are passing values that may contain spaces or special characters, use the function URLEncodedFormat.

```
<CFSET FullName="Bob Smith">
<CFOUTPUT>
<A HREF="printname.cfm?FullName=#URLEncodedFormat(FullName)#">
Click here</A>
</CFOUTPUT>
```

See the *CFML Language Reference* for more information on the URLEncodedFormat function.

Passing parameters with a form

You can use a form to pass variables between pages by allowing a user to enter or choose a value in a form. Use the resulting Form variables on the form's action page to process values passed in the form. You can also use hidden input types to pass a variable in a form to another page.

Example: Hidden input

The form below includes a hidden input named "Customer_ID" that is passed to the `example.cfm` page:

```
<FORM ACTION="example.cfm" METHOD="Post">
        <INPUT TYPE="Hidden"
            NAME="Customer_ID" VALUE="24">
        <INPUT TYPE="Submit" VALUE="Enter">
</FORM>
```

The `example.cfm` page will be passed the parameter *Form.Customer_ID* with the value set by the previous page.

Example: Dynamic parameters

It is also possible to pass a value based on a dynamic parameter, such as the result of a query, as illustrated below:

```
<FORM ACTION="example.cfm" METHOD="Post">

<CFOUTPUT QUERY="GetCustomer">
    <INPUT TYPE="Hidden" NAME="Customer_ID"
        VALUE="#Customer_ID#">
</CFOUTPUT>

<INPUT TYPE="Submit" VALUE="Enter">
</FORM>
```

In the `example.cfm` page, you can refer to the variables created with this form as *Form.Customer_ID*.

Kinds of Variables

ColdFusion supports several types of variables. This section describes the types of variables available and how you use them.

The following table describes the types of variables you can use in a ColdFusion application page.

ColdFusion Variable Types	
Variable Type	**Description**
Queries	As soon as a query has been run, you can use its results as dynamic parameters. For example, if you create a query named LookupUser that finds the ID for a given user name, you might want to use this ID in another query or in a CFOUTPUT.
Local Variables	This is the default scope for variables created with the CFSET and CFPARAM tags. For example, <CFSET A=5> sets the variable A to 5. This variable is available only on the page where it is created and any included pages.
URL Parameters	Parameters appended to URLs after the application page name using a *variablename=value* format.
Form Fields	The most common way of passing parameters to a page is to use form fields. When a user enters data in a form field, a parameter with the name of the form field is passed to the action page.

ColdFusion Variable Types	
Variable Type	**Description**
Client	ColdFusion client variables are variables associated with a particular client. Client variables allow you to maintain state as a user moves from page to page in an application. They are stored in the system registry by default, but you can also choose to store them in a cookie or in a database. For more information, see the Using the Application Framework chapter of this book.
Server	ColdFusion server variables are associated with the current Web server and are available to all ColdFusion applications until the ColdFusion server is shut down. This server scope allows you to define variables that all your ColdFusion application pages can reference.
Session	Session variables are tied to an individual client and persist for as long as that Client ID maintains a session. Session variables, like current client variables, require a client name to work and are only available to that Client ID. Unlike client variables, session variables are stored in the server's memory and can be set to time-out after a precise period of inactivity.
Application	Application variables are tied to an individual application as defined in the CFAPPLICATION NAME attribute, typically used in the `Application.cfm` file. Application variables only work if an application name has been defined. For more information about the `Application.cfm` file, see the Using the Application Framework chapter of this book.
HTTP Cookies	HTTP Cookie variables are stored in the browser. They are available every time the browser makes a page request. You can create cookie variables with the CFCOOKIE tag.
CGI Environment	Every page request has several environment variables sent to it that relate to the context in which it was requested. The variables available will depend on the browser and server software in use for a given request. **Note**: CGI environment variables are created even when you are using one of the Web servers that supports a server API.

The section How ColdFusion looks up variables describes the order in which ColdFusion finds variables in application pages.

Using variables across several application pages

Within ColdFusion, most variables apply only to a single template. However, four types of variables can be used across multiple application pages:

- **Client variables** store identifying information about an individual client.

  Client variables are designed to hold long-term information particular to an individual client. You can store client variables in the system registry, a database, or in a cookie named for the application.

- **Server variables** are valid for all ColdFusion applications on a particular server.

  Server variables are used to store information (typically read-only) that does not change often and can be shared across many users and ColdFusion applications.

- **Session variables** are valid for as long as an individual Client ID maintains a session.

  Session variables are useful for storing short-term information needed for a single site visit or set of requests. They are specific to individual users.

- **Application variables** are valid for a specified application.

  Application variables are available to individual, specified applications and to all users of that application who access your ColdFusion server. They are stored in the server's memory and can be set to time out at specified intervals. Application variables are general and available to all users.

Note In cases where variables are available to several application pages, make sure to keep your variable names straight. See Variable names in this chapter for more information.

Client Variables

In ColdFusion, client variables help you to keep track of users as they move through the pages within an application. They offer a convenient way to store user preferences.

ColdFusion achieves client state management by creating a client record for each browser that requests a page in an application in which client state management is enabled. The client record is identified by a unique token (a combination of CFID and CFTOKEN), which is stored in an HTTP cookie in the user's browser.

The application can then define variables within the client record. These client variables are accessible as parameters in every application page that the client requests within the scope of an application.

See the Client Variables and Client State Management section in the Using the Application Framework chapter for information on setting up client state management using the CLIENTMANAGEMENT attribute in the CFAPPLICATION tag.

Creating a client variable

When client state management is enabled for an application, you can use the system to keep track of any number of variables associated with a particular client.

You create client variables by defining a variable with the **Client** scope using either the CFSET or CFPARAM tag.

To create a client variable:

1. Turn on client state management by setting `CLIENTMANGEMENT="Yes"` in the CFAPPLICATION tag in your Application.cfm file.

   ```
   <CFAPPLICATION NAME="myapplication"
       CLIENTMANAGEMENT="Yes">
   ```

2. Also in the CFAPPLICATION tag, choose a storage location for your client variables by adding the CLIENTSTORAGE attribute.

   The default location is set in the Variables page of the ColdFusion Administrator. It can be the system registry, an existing datasource, or cookies.

   See Client variable storage options for more information.

   `CLIENTSTORAGE="mydatasource"`

3. In ColdFusion Studio, click the CFSET tool. Enter the variable name, preceded by the Client scope, and assign a value to it, for example:

   ```
   <CFSET Client.FavoriteColor="Cornflower Blue">
   ```

See Variable names for guidelines on creating variable names.

Once a client variable has been set in this manner, it is available for use within any application page in your application that is accessed by the client for whom the variable is set.

Standard client variables

In addition to storing custom client variables, the Client object has several standard variables. These variables can be useful in providing customized behavior depending 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>
```

The standard Client object attributes are read-only (they can be accessed but not set by your application) and include

- CFID
- CFToken

- URLToken
- HitCount
- TimeCreated
- LastVisit

Note In the Variables page of the ColdFusion Administrator, you disable automatic updating of global variables. This keeps ColdFusion from updating every page when client variables are enabled. See *Administering ColdFusion Server* for more information.

Using client state management without cookies

You can use ColdFusion's client state management without cookies. To use this functionality, you must pass the client ID (CFID) and the client security token (CFTOKEN) between pages, either in hidden form fields or appended to URLs.

Note In ColdFusion, client state management is explicitly designed to work with cookies, the standard tool for identifying clients. Using client state management without cookies requires careful programming to ensure that the URLToken is always passed between application pages.

Client variable storage options

The system-wide default for storing client variables is in the system registry. In the ColdFusion Administrator, you can change this setting and choose instead to store client variables in a SQL database or in cookies.

You do this in two steps: by configuring the client variable storage option in the ColdFusion Administrator, and then noting the client variable storage location in the CFAPPLICATION tag. See *Administering ColdFusion Server* for information on using the Administrator to set up a client variable storage location.

You use the CLIENTSTORAGE attribute in the CFAPPLICATION tag to specify where you want to store client variables, providing one of three values:

- `Registry`
- The name of a configured client store
- `Cookie`

The following example shows how you enable client state management using a sample database called *mydatasource*.

```
<CFAPPLICATION NAME="myapplication"
    CLIENTMANAGEMENT="Yes"
    CLIENTSTORAGE="mydatasource">
```

If no CLIENTSTORAGE setting is specified, the default location, as noted in the ColdFusion Administrator Variables page, is used.

Note Client storage mechanisms are exclusive; when one storage type is in use, the values stored in other client stores are unavailable.

Storing client variables in cookies

When you choose `CLIENTSTORAGE="Cookie"`, ColdFusion creates a cookie named for the application name. Storing client data in a cookie is scalable to large numbers of clients, but this storage mechanism has some limitations. Chief among them is that if the client turns off cookies in the browser, client variables won't work.

Consider these additional limitations before implementing cookie storage for client variables:

- Netscape Navigator allows only 20 cookies from a particular host to be set. ColdFusion uses two of these cookies for CFID and CFTOKEN, and also creates a cookie named CFGLOBALS to hold global data about the client, such as HitCount, TimeCreated, and LastVisit.
- Netscape Navigator sets a size limit of 4K bytes per cookie. ColdFusion encodes non-alphanumeric 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 for each client. ColdFusion will throw an error if you try to store more than 4000 encoded bytes of data for a client.

Getting a list of client variables

To obtain a list of the custom client parameters associated with a particular client, use the GetClientVariablesList function.

```
<CFOUTPUT>#GetClientVariablesList()#</CFOUTPUT>
```

The GetClientVariablesList function returns a comma-separated list of variable names defined for the application context declared by CFAPPLICATION, if any. The standard system-provided client variables (CFID, CFToken, URLToken, HitCount, TimeCreated, and LastVisit) are not returned in the list.

Deleting client variables

Unlike normal variables, client variables and their values persist over time. (In this fashion they are akin to cookies.) To delete a client variable, use the DeleteClientVariable function. For example:

```
<CFSET IsDeleteSuccessful=DeleteClientVariable("MyClientVariable")>
```

The DeleteClientVariable function operates only on variables within the scope declared by CFAPPLICATION, if any.

Also, through the Variables page of the ColdFusion Administrator, you can edit the client variable storage to remove client variables after a set number of days. (The default value is 90 days if client variables are stored in the registry, ten days if stored in a datasource.)

Note You cannot delete the system-provided client variables (CFID, CFToken, URLToken, HitCount, TimeCreated, and LastVisit).

For more information

For more information about using client variables in ColdFusion applications, see Chapter 6, "Using the Application Framework," on page 67.

Server Variables

Server variables are available to all users and all applications accessing the current web server, until the ColdFusion server is shut down.

Server variables are useful for storing information that does not change often and can be shared across many users and ColdFusion applications. You might use server variables to store information needed by all applications running at your site — for example, information about a datasource. Use server variables when it's more efficient to store general information in memory than to run a query with each request.

In addition, the following built-in server variables are available in ColdFusion and provide feedback on your ColdFusion server:

- Server.ColdFusion.ProductName — Returns the ColdFusion product name.
- Server.ColdFusion.ProductVersion — Returns the ColdFusion product release information.
- Server.ColdFusion.ProductLevel — Returns the ColdFusion product level information.
- Server.ColdFusion.SerialNumber — Returns the ColdFusion serial number.
- Server.OS.Name — Returns the server operating system name.
- Server.OS.AdditionalInformation — Returns additional information about the server operating system.
- Server.OS.Version — Returns the operating system version of the server.
- Server.OS.BuildNumber — Returns the build number of the server operating system.

Note Server variables are read-write; be careful not to overwrite these built-in server variables.

Sample server variable output

The following table shows sample server variable output for a Windows NT installation:

Server Variable Sample Output	
Variable	**Sample Output**
Server.ColdFusion.ProductName	`ColdFusion Engine`
Server.ColdFusion.ProductVersion	`4, 0, 0, 0`
Server.ColdFusion.ProductLevel	`Professional`
Server.ColdFusion.SerialNumber	`varies`
Server.OS.Name	`NT/Solaris`
Server.OS.AdditionalInformation	`Service Pack 3`
Server.OS.Version	`4.0/5.0.3`
Server.OS.BuildNumber	`1381`

For information about the CGI variables that are created by Web servers and browsers, see "Using CGI Environment Variables" on page 51 in this chapter.

Using Application and Session Variables

Session and application variables are persistent variable "scopes." You access these variables by prefacing the variable name with the scope name, for example: "Session.MyVariable" or "Application.MyVariable." And because they are persistent, you can pass values between pages with a minimum of effort.

Enabling application and session variables

Session and application variables are similar in operation to client variables. Like client variables, they are enabled with the CFAPPLICATION tag. See the *CFML Language Reference* for more information on the CFAPPLICATION tag.

Unlike client variables, however, which are stored in either the system registry, a datasource, or a cookie, application and session variables are always stored in the ColdFusion server's memory. This method offers obvious performance advantages. In addition, you can set time-out values for these variables either with CFAPPLICATION, or by specifying time-outs in the ColdFusion Administrator. You can also simply disable application and session variables entirely.

For information on setting time-outs for variables, see the *Administering ColdFusion Server* book.

Session variables

Use session variables when the variables are needed for a single site visit or set of requests. For example, you might use session variables to store a user's selections in a shopping cart application. (You would use client variables if the variable is needed for future visits.)

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 as long as a single user is running that application. Using session variables, an application could initialize itself with user-specific data the first time a user hit a page of that application. This information could then remain available while that user continues to use that application. For example, information about a specific user's preferences could be retrieved from a database once, the first time the user hits any page of an application. This information would remain available throughout that user's session, thereby avoiding the overhead of retrieving the preferences again and again.

Session variables work exactly as client variables do, in that they require a client name (client ID) and are always scoped within that client ID. Session variables also work within the scope of an application name if one is supplied, in which case their scope will be the combination of the client ID and the application name.

To enable session variables, set `SESSIONMANAGEMENT="Yes"` in the CFAPPLICATION tag in your `Application.cfm` file.

You create session variables using the CFSET tag. See "Using CFSET to create variables" on page 32 in this chapter.

Session variable time-outs

Session variables have a specific lifetime, and it is this lifetime that defines a "session." For example, when you access a session variable within its specified lifetime, the variable returns a value because your request occurs during a single, time-limited session. On the other hand, if you do not access a session variable within its specified lifetime, the variable will time-out and will no longer be available to you.

The default time-out for session variables is set to 20 minutes. In the Variables page of the ColdFusion Administrator, you can change this time-out value. See the *Administering ColdFusion Server* book for more information.

You can also set the time-out period for session variables inside a specific application (thereby overruling the Administrator default setting) by using the SESSIONTIMEOUT attribute of the CFAPPLICATION tag.

Application variables

An application refers to all the related web pages that make up a particular piece of functionality. In ColdFusion, you define an application by giving it a name using the

CFAPPLICATION tag. By using the same application name in a CFAPPLICATION tag, a set of templates can define themselves as being part of the same logical application.

Note The value you set for the NAME attribute in CFAPPLICATION is limited to 64 characters.

You create application variables using the CFSET tag. See "Using CFSET to create variables" on page 32 in this chapter.

Using application variables

Application variables are designed to store application-level data. They are a convenient place to store information that all pages of your application might need no matter who (what client) is running that application. Using application variables, an application could initialize itself, say, when the first user hits any page of that application. This information could then remain available indefinitely to all subsequent hits of any pages of that application, thereby avoiding the overhead of repeated initialization.

Because the data stored in application variables are available to all pages of an application and remains available indefinitely, or until the ColdFusion server shuts down, application variables are very convenient. However, because all clients running an application see the same set of application variables, they are not useful for client-specific information. If you wish to do the same sorts of things, but on a client-specific basis, session variables are for you.

Application variables work very much as client variables do, except that they require an application name be associated with them. (Thus, they are always scoped within that application name.)

Unlike client variables, however, they do not require that a client name (client ID) be associated with them. Thus, they are available to any clients that specify the same application name.

Differentiating client, session, and application variables

Here's a table that maps out these relationships:

Client, Session, and Application Variables

Variable Type	Application Names	ClientIDs	Client Mgmt	Session Mgmt	Time-out
Client	Optional	Required	Required	n/a	Optional
Session	Optional	Required	Required	Required	Optional
Application	Required	n/a	n/a	n/a	Optional

Application variable time-outs

Application variables have a specific lifetime, and it is this lifetime that defines an "application." For example, when you access an application variable inside a specific application, the variable returns a value because your request occurs on a page declared in the CFAPPLICATION tag to be part of a single application.

The default time-out period for application variables is two days. In the Variables page of the ColdFusion Administrator, you can define time-out values for application and session variables. See the *Administering ColdFusion Server* book for more information.

You can also set the time-out period for application variables inside a specific application (thereby overruling the Administrator default setting) by using the APPLICATIONTIMEOUT attribute of the CFAPPLICATION tag.

Scoping application and session variables

ColdFusion does not attempt to automatically evaluate Application and Session variables. You must use variable prefixes with these variables, as in Session.variablename or Application.variablename.

For more information about application and session variables, see the Using the Application Framework chapter in this book.

Creating HTTP Cookie Variables

Cookies are a general mechanism used by server-side applications such as ColdFusion to store information in individual browsers. Cookies stored in a browser can then be retrieved by the server-side application. With cookies, applications can create variables specifically for an individual browser. For example, you could create a cookie for background color and then customize the background color of your site for each user.

Cookies are domain-specific, that is, they are set and retrieved for a specific server reference, such as www.allaire.com or 127.0.0.1. A specific domain can set a maximum of 20 cookies in a user's browser (ColdFusion uses two of these cookies, for CFID and CFTOKEN).

Using the Secure Sockets Layer (SSL), cookies can be sent securely. They are persistent, so they will stay stored in the browser until they expire or are deleted. Cookies are currently supported by almost all major commercial browsers.

Creating cookies with the CFCOOKIE tag

A cookie created using the CFCOOKIE tag is available to all ColdFusion pages as well as other Web applications in the domain that can access cookies. This means you can pass parameters to subsequent pages using browser cookies.

Note that cookies were not designed to store secure information such as passwords or credit card numbers.

To create a cookie:

1. In ColdFusion Studio, click the Cookie tool button on the CFML Advanced toolbar.
2. Enter the name and value of the cookie variable. This example creates a variable named Cookie.User_ID with a value of 2344, which will expire in 100 days:

```
<CFCOOKIE NAME="User_ID" VALUE="2344"
    EXPIRES="100">
```

Note If ColdFusion executes a CFLOCATION tag on the same page following the creation of cookie variables with CFCOOKIE, the cookie variables are lost.

For more information on CFCOOKIE, see the *CFML Language Reference*.

Using cookies in a page

Once you store a cookie in the client's browser, it is automatically sent to your Web server every time a page is requested by that client. The value of a cookie variable can be accessed in the same way that other types of dynamic parameters (such as URL and Form variables) are accessed. For example, use the Cookie prefix to display the User_ID cookie variable created in the previous example in a CFOUTPUT section, following this syntax:

```
<CFOUTPUT> #Cookie.User_ID# </CFOUTPUT>
```

Note It's a good idea to test whether a cookie exists before you use it in an application page.

Deleting cookies

To delete a cookie, you use the CFCOOKIE tag with the EXPIRES attribute set to "now":

```
<CFCOOKIE NAME="User_ID" VALUE="#User_ID#"
    EXPIRES="now">
```

The cookie will be deleted when the user closes the browser.

Using CGI Environment Variables

Each time a browser makes a request to a server, a set of environment variables are created, some by the Web server and some by the browser.

In ColdFusion, these variables are referred to as CGI environment variables and they use the "CGI" prefix (even if your server is using a server API instead of CGI to communicate to the ColdFusion Server).

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

Note The environment variables available to your applications depend on the browser and server software in use for a given request.

Testing for CGI variables

Because not all CGI variables are supported by every browser, ColdFusion always returns TRUE when testing for the existence of a CGI variable, even if the variable is not supported by the client browser. The way around this is to test for an empty string, instead of a boolean return, to see whether the CGI variable is available.

```
<CFIF CGI.varname IS NOT "">
    CGI variable exists
<CFELSE>
    CGI variable does not exist
</CFIF>
```

CGI server variables

The following table describes the most common CGI environment variables created on the server (not all of these will be available with every server):

CGI Server Variables	
Variable	**Description**
SERVER_SOFTWARE	The name and version of the information server software answering the request (and running the gateway). Format: name/version.
SERVER_NAME	The server's hostname, DNS alias, or IP address as it appears in self-referencing URLs.
GATEWAY_INTERFACE	The revision of the CGI specification to which this server complies. Format: CGI/revision.
SERVER_PROTOCOL	The name and revision of the information protocol this request came in with. Format: protocol/revision.
SERVER_PORT	The port number to which the request was sent.
REQUEST_METHOD	The method with which the request was made. For HTTP, this is Get, Head, Post, and so on.
PATH_INFO	The extra path information, as given by the client. Scripts can be accessed by their virtual pathname, followed by extra information at the end of this path. The extra information is sent as PATH_INFO.
PATH_TRANSLATED	The server provides a translated version of PATH_INFO, which takes the path and does any virtual-to-physical mapping to it.

CGI Server Variables	
Variable	**Description**
SCRIPT_NAME	A virtual path to the script being executed; used for self-referencing URLs.
QUERY_STRING	The query information that follows the ? in the URL that referenced this script.
REMOTE_HOST	The hostname making the request. If the server does not have this information, it sets REMOTE_ADDR and does not set REMOTE_HOST.
REMOTE_ADDR	The IP address of the remote host making the request.
AUTH_TYPE	If the server supports user authentication, and the script is protected, this is the protocol-specific authentication method used to validate the user.
REMOTE_USER AUTH_USER	If the server supports user authentication, and the script is protected, this is the username they have authenticated as. (Also available as AUTH_USER.)
REMOTE_IDENT	If the HTTP server supports RFC 931 identification, this variable is set to the remote username retrieved from the server. Use this variable for logging only.
CONTENT_TYPE	For queries that have attached information, such as HTTP POST and PUT, this is the content type of the data.
CONTENT_LENGTH	The length of the content as given by the client.

CGI client variables

The following chart describes the most common CGI environment variables created by the browser and passed in the request header:

CGI Client Variables	
Variable	**Description**
HTTP_REFERER	The referring document. This is the document that linked to or submitted form data.
HTTP_USER_AGENT	The browser the client is currently using to send the request. Format: software/version library/version.

CHAPTER 5

Controlling Page Flow

This chapter discusses the methods for controlling application page flow. Each time an application page is requested, ColdFusion dynamically constructs an HTML page based on the page's CFML and HTML content.

Application page flow control gives you a great deal of flexibility in how dynamic pages are created and how your application will function. You can control how ColdFusion processes your application pages using CFSWITCH and CFIF blocks to control page flow by testing different conditions.

This chapter also describes how to include other application pages in your pages, how to use looping tags, and how to redirect users.

Contents

Conditional Processing (CFIF and CFSWITCH)

ColdFusion offers two ways to handle conditional processing: CFSWITCH and CFIF. These tags allow you to customize the behavior of your ColdFusion applications in powerful ways.

Using CFSWITCH with CFCASE and CFDEFAULTCASE

Used with CFCASE and CFDEFAULTCASE, the CFSWITCH tag evaluates a passed expression and passes control to the CFCASE tag that matches the expression result. You can also optionally include a CFDEFAULTCASE, which receives control if there is no matching CFCASE tag value.

The CFSWITCH tag offers better performance than a series of CFIF/CFELSEIF tags, and the resulting code is easier to read. However, the VALUE attributes of CFCASE tags must be constants whose value is known at when server processes the page.

The following example shows the syntax of a simple CFSWITCH tag:

```
<CFSWITCH EXPRESSION=#Switch#>
    <CFCASE VALUE="4"> Case four </CFCASE>
    <CFCASE VALUE="1"> Case one </CFCASE>
    <CFCASE VALUE="2"> Case two </CFCASE>
    <CFCASE VALUE="3"> Case three </CFCASE>
    <CFCASE VALUE="2.5"> Case two and a half </CFCASE>
    <CFCASE VALUE="5"> Case five </CFCASE>
    <CFCASE VALUE="6"> Case six </CFCASE>

<CFDEFAULTCASE> Default case </CFDEFAULTCASE>

</CFSWITCH>
```

In each CFCASE tag, the VALUE attribute shows one or more constant values that CFSWITCH compares to the specified expression (in this example, the variable #Switch#). If a value matches the expression, CFSWITCH executes the code between the CFCASE start and end tags.

See the *CFML Language Reference* for details on the syntax of CFSWITCH.

Using CFIF with CFELSEIF and CFELSE

You can also use CFIF, CFELSE, and CFELSEIF to conditionally process a section of an application page.

The specific syntax for a simple conditional block is:

```
<CFIF value operator value>
... HTML and CFML tags
<CFELSE>
... HTML and CFML tags
</CFIF>
```

The CFELSE tag is not required.

Note that the expression inside the CFIF tag uses "operators" such as IS, IS NOT, etc. rather than equal signs. See the Functions and Expressions chapter in *Advanced ColdFusion Development* for information on conditional operators in ColdFusion.

Example 1: Conditionally returning a query result set

To test whether a query returned any records, you can check wither the query's record count is zero, using the syntax:

```
<CFIF #CustomerSearch.RecordCount# IS 0>

<!--- Inform user that we had no hits --->
<P>Sorry, no customers matching your
criteria were found.</P>

<CFELSE>
<!--- Show the list of customers retrieved --->
    <CFOUTPUT Query="Customers">
    #FirstName# #LastName# <BR>
    </CFOUTPUT>
</CFIF>
```

To display an output section only if the user explicitly requests it, as recorded in this example in a variable called *ShowCustomers*, use the syntax:

```
<CFIF #Form.ShowCustomers# IS "Yes">
    Customer List: <P>
    <CFOUTPUT Query="Customers">
    #FirstName# #LastName# <BR>
    </CFOUTPUT>
</CFIF>
```

Example 2: Conditionally returning a record section

One of the most powerful uses of conditional tags is for record-by-record formatting of query results. To accomplish this, place the conditional tags within CFOUTPUT sections.

When conditional tags are placed within CFOUTPUT sections, they are evaluated once for every row in the query result set. This allows you to customize the display of results depending upon whether or not a field is present in an individual row.

For example, in this example, not every contact in the result set has a phone number. So putting the CFIF inside CFOUTPUT ensures that the "Phone:" label is printed only where the #Phone# field has a value. The label is not printed if the #Phone# variable is empty.

```
<CFOUTPUT QUERY="Contacts">

    <HR>
    Name: #Name# <BR>
    Title: #Title# <BR>
<CFIF #Phone# IS NOT "">
    Phone: #Phone# <BR>
</CFIF>
</CFOUTPUT>
```

Compound conditional statements

A compound conditional statement combines multiple conditional statements with Boolean operators. The specific syntax for a compound conditional block is:

```
<CFIF (value operator value) Boolean Operator
    (value operator value) Boolean Operator
    (value operator value)>

... HTML and CFML tags
</CFIF>
```

You can use CFELSE and CFELSEIF with compound conditional blocks. The most common Boolean operators are:

Operator	Description
AND	Conjunction
OR	Disjunction
NOT	Logical negation

Example

The following example assumes that a query named "GetEmployee" has returned information about an employee. The code displays an HTML message if the query result for an employee file shows that the employee is both in the sales department and earns a bonus of more than $5,000 a year.

```
<CFIF (#GetEmployee.Department# IS "Sales") AND
    (#GetEmployee.Bonus# GE 5000)>

<H4>Congratulations on your sales bonus!</H4>
</CFIF>
```

Using CFELSEIF

Using CFELESIF allows you to perform sophisticated conditional processing within your dynamic pages. It offers a way to combine conditional statements without

multiple nested IF statements. The following example shows the syntax of a conditional block.

```
<CFIF condition1>
    Display this text only if condition1 is true.

<CFELSEIF condition2>
    Display this text only if condition1 is
    false and condition2 is true.

<CFELSEIF condition3>
    Display this text only if condition1
    and condition2 are false and condition3 is true.

<CFELSE>
    Display this if condition1, condition2,
    and condition3 are false.
</CFIF>
```

Note CFELSEIF can only be used within a CFIF. The CFELSE is optional.

Example

Imagine an application in which you enter your age and then see a message based on that age. The following conditional block would display the proper message depending on the age you entered.

```
<CFIF #Form.Age# LESS THAN 12>
You're only a child (0 - 11 years old).

<CFELSEIF #Form.Age# LESS THAN 20>
You're a teenager (12 - 19 years old).

<CFELSEIF #Form.Age# LESS THAN 60>
You're an adult (20 - 59 years old).

<CFELSE>
You're a senior citizen (older than 59).
</CFIF>
```

Redirecting Application Page Requests (CFLOCATION)

You can redirect a page request to another page or to another URL using the CFLOCATION tag. This is useful if you want to define an application page that performs one or more CFQUERYs and then moves on to another page, or if you want the URL to which the user is directed to depend on a dynamic parameter.

Example

For example, you can use CFIF to test if a user is logged in (has the user's password been confirmed?) and if not, relocate the user to another URL using CFLOCATION:

```
<CFIF #NewPassword# IS NOT '#PasswordConfirmation#'>
    <CFLOCATION URL="invalidpassword.cfm">
</CFIF>
```

This example manages validation of a new user's password by evaluating whether the new password was confirmed properly. If not, the application page routes the user to a different page that notifies the user that the password was not confirmed properly (for example, "invalidpassword.htm").

It is also possible to use dynamic parameters within the URL attribute of the CFLOCATION tag. For example, to dynamically determine the page to send the user to based on a dynamic parameter named "Page," use the syntax:

```
<CFLOCATION URL="#Page#">
```

Stopping Application Page Processing (CFABORT)

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

As the following example illustrates, the CFABORT tag has no attributes.

```
<P>This HTML content is returned
    to in the final page.</P>
    <CFABORT>
<P>This HTML content is NOT returned
    in the final page.</P>
```

Including Application Page Files (CFINCLUDE)

As the applications you develop with ColdFusion become more complex, you will require ways to simplify the presentation of your CFML application page files and to reuse code. The CFINCLUDE tag helps to fulfill these requirements.

Essentially, CFINCLUDE inserts another application file into the current page. It's a handy way to reuse common code.

Note You cannot use CFINCLUDE to cause a CFIF, CFLOOP, or CFOUTPUT tag block to be split across different application pages.

The CFINCLUDE tag has a TEMPLATE attribute that specifies an existing application page file to process and return to the client. This application page file is processed as if it is a part of the file into which it is included. For example, if you create a variable in an

application page and use a CFINCLUDE, the included application page can reference the variable created in the main application page.

Syntax

The syntax of the CFINCLUDE tag is:

```
<CFINCLUDE TEMPLATE="FileName">
```

Here, *FileName* represents the relative path of the application page file to be included.

Examples

For example, one of the simplest applications of CFINCLUDE is for adding common header and footer code to your pages, following this model:

```
<CFINCLUDE TEMPLATE="header.cfm">
... Page contents
<CFINCLUDE TEMPLATE="footer.cfm">
```

To process the file `index.cfm`, which is up one directory level from the current application page:

```
<CFINCLUDE TEMPLATE="../index.cfm">
```

Potential uses of the CFINCLUDE tag

- Your application might display the same CFTABLE repeatedly on different pages. Rather than copy and paste the code from page to page, you could create an application page for the table and include it wherever it is needed.
- You could create application page files called "header.cfm" and "footer.cfm," which contain a standard set of HTML text and tags to include at the beginning and end of the pages in your application. You could then modify the look and feel of your entire site by changing just two application pages.
- You might have a complicated set of nested CFIF/CFELSE statements that could be clarified if the code to be executed was hidden away in application pages rather than displayed inline.

Do not split code blocks across included pages. Be sure to keep beginning and ending tags together.

Creating Loops (CFLOOP)

Looping is a very powerful programming technique that lets you repeatedly display a set of instructions or some output depending on a particular set of conditions. CFLOOP allows for four different types of loops:

- Index loops (also called FOR loops)
- Conditional loops (also called WHILE loops)

- Looping over a query
- Looping over a list
- Looping over a COM collection

You use the attributes of the CFLOOP tag to choose which type of loop to use.

Index loops

An index loop repeats based on a range of numeric values. Use this type of loop when you know the number of times the loop should run.

Example of an index loop

The INDEX variable will be incremented on each iteration of the loop. The following example loops five times, displaying the index value of the loop each time:

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

The results would look like this in a browser:

The loop index is 1.

The loop index is 2.

The loop index is 3.

The loop index is 4.

The loop index is 5.

Example 2 of a stepped index loop

The STEP value has a default value of 1, but you can set the step value to change the way the INDEX value is incremented. The following example counts backwards from 5:

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

The results would look like this in a browser:

The loop index is 5.

The loop index is 4.

The loop index is 3.

The loop index is 2.

The loop index is 1.

Index loops are commonly known as a FOR loop, as in 'loop FOR this range of values.'

Conditional loops

A conditional loop iterates over a set of instructions while a given condition is true. To use this type of loop correctly, the instructions must change the condition every time the loop iterates until the condition evaluates as FALSE.

Example

The following example increments the parameter "CountVar" from 1 to 5.

```
<!--- Set the variable CountVar to 1  --->
<CFSET #CountVar#= 0>

<!--- Loop until CountVar is 5 --->
<CFLOOP CONDITION="CountVar LT 5">

<CFSET #CountVar#=#CountVar# + 1>
The loop index is <CFOUTPUT>#CountVar#</CFOUTPUT>.<BR>
</CFLOOP>
```

The results would look like this in a browser:

The loop index is 1.

The loop index is 2.

The loop index is 3.

The loop index is 4.

The loop index is 5.

Conditional loops are commonly known as WHILE loops, as in 'loop WHILE this condition is true.'

Query Loops

A loop over a query repeats for every record in the query result set. The CFLOOP results are just like a CFOUTPUT. During each iteration of the loop, the columns of the current row will be available for output.

On the other hand, using a CFLOOP query is much slower than using CFOUTPUT with a query.

Example: Using CFLOOP to display a record set

The following example shows a CFLOOP tag that works just like a CFOUTPUT:

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

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

Example: Next *n* record sets in a query

CFLOOP also provides iteration over a record set with dynamic starting and stopping points. Thus, you can begin at the tenth row in a query and end at the twentieth.

Using this mechanism provides a simple means to get the next *n* sets of records from a query. The following example loops from the tenth through the twentieth record returned by "MyQuery:"

```
<CFSET Start=10>
<CFSET End=20>

<CFLOOP QUERY="MyQuery" STARTROW="#Start#" ENDROW="#End#">
    <CFOUTPUT>#MyQuery.MyColName#</CFOUTPUT><BR>
</CFLOOP>
```

The loop is done when there are no more records or when the current record is greater than the value of the ENDROW attribute.

Example: Looping over a query

The following example uses the CFINCLUDE tag to combine into a single document all the application pages returned in querying a list of application page names.

```
<CFQUERY NAME="GetFile" DATASOURCE="Library" MAXROWS=5>
    SELECT FileName FROM Templates
</CFQUERY>

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

If you just need to loop through a query "Record Count" number of times, you can use CFOUTPUT, as in this example:

```
<CFOUTPUT QUERY="MyQuery">
    Text and #variablename#
</CFOUTPUT>
```

List Loops

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

This loop displays the names of each of the Beatles:

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

Although CFLOOP expects elements in the list to be separated by commas by default, you are free to specify your own element boundaries in the DELIMITER attribute.

Here's the same loop as before, only this time CFLOOP will treat commas, colons, or slashes as list element delimiters:

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

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

Looping over a COM collection

The CFLOOP COLLECTION attribute allows you to loop over a COM/DCOM collection object. A COM/DCOM collection object is a set of similar items referenced as a group rather than individually. For example, the group of open documents in an application is a type of collection. Each collection item is referenced in the CFLOOP by the variable name you supply in the ITEM attribute. This type of iteration is generally used to access every object within a COM/DCOM collection. The loop is executed until all objects have been accessed.

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

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

```
<CFOBJECT CLASS="FileFunctions.files"
    NAME="FFunc"
    ACTION="Create">

<CFSET FFunc.Path="c:\">
<CFSET FFunc.Mask="*.*">
<CFSET FFunc.attributes=16>
<CFSET x=FFunc.GetFileList()>

<CFLOOP COLLECTION=#FFUNC# ITEM="file2">
    <CFOUTPUT> #file2.name# <BR>
    </CFOUTPUT>
</CFLOOP>
```

Nesting loops

A CFLOOP block can also contain other CFLOOPs. In this case, the inner loop will execute from start to finish as many times as the outer loop executes. Loops can be nested as deeply as you like.

Here the inner loop will execute on each iteration of the outer loop for a grand total of 15 times:

```
<CFLOOP INDEX="OuterLoopCount" FROM="1" TO="3">
    <CFOUTPUT>Outer loop #OuterLoopCount#</CFOUTPUT><BR>

    <CFLOOP INDEX="InnerLoopCount" FROM="1" TO="5">
    <CFOUTPUT>Inner loop #InnerLoopCount#</CFOUTPUT><BR>
    </CFLOOP>
</CFLOOP>
```

Breaking out of a loop

When ColdFusion encounters the CFBREAK tag in a loop, it terminates the execution of the loop and proceeds to process the application page starting with the line immediately following the end of the loop. This is generally less elegant than constructing the loop with an appropriate conditional, but there are times when breaking out of a loop can simplify a tag structure that's complicated or deeply nested.

The following example isn't a terribly good use of CFBREAK, but it shows the tag in action:

```
<CFLOOP INDEX="LoopCount" FROM=1 TO=100>
The value is <CFOUTPUT>#LoopCount#</CFOUTPUT>.<BR>
    <CFIF LoopCount IS 7>
    <CFBREAK>
    </CFIF>
</CFLOOP>
```

The results would look like this in a browser:

The value is 1.

The value is 2.

The value is 3.

The value is 4.

The value is 5.

The value is 6.

The value is 7.

For more information

For more information on the CFML tags used for controlling page flow, see the *CFML Language Reference*.

CHAPTER 6

Using the Application Framework

The ColdFusion Web Application Framework is a powerful tool you can use to help structure your ColdFusion applications. This section describes how to create and use the `Application.cfm` file, the application page that controls the application framework.

Contents

Understanding the Web Application Framework

A ColdFusion application is a collection of application pages that work together. Applications can be as simple as a guest book or as sophisticated as a full Internet commerce system with catalog pages, shopping carts, and reporting. You can combine individual applications to create advanced Web systems.

The ColdFusion Web Application Framework is based on four basic components:

- Application-level settings and functions
- Client state management
- Custom error handling
- Web server security integration

With these components, you can easily combine your ColdFusion application pages into sophisticated Web applications.

Application-level settings and functions

ColdFusion offers application-level features that help you control settings, variables, and features available across the entire application. Once you have defined an application, you can use the application-level features in addition to all of the other features in ColdFusion.

The application framework relies on a special application-wide template called `Application.cfm`, which defines application-level settings and functions such as:

- The application name
- Client state management options
- Application and session variables
- Default variables
- Custom error pages
- Data sources
- Default style settings
- Exclusive locks
- Other application-level constants

Note Because UNIX is case sensitive, the application framework file must be spelled with an initial capital, `Application.cfm`, for applications that run on UNIX platforms.

See Establishing Application-Level Settings in this chapter.

Client state management

Because the Web is a stateless system, each connection a browser makes to a Web server is unique in the eyes of the Web server. However, within an application it is important to be able to keep track of users as they move through the pages within the application. This is the definition of client state management.

You can maintain client state by seamlessly tracking variables for a browser as the user moves from page to page in an application. This can be used in place of other methods for tracking client state such as using URL parameters, hidden form fields, and HTTP cookies.

See Enabling Client State Management for more information.

Custom error handling

Using the CFERROR tag, you can display customized HTML pages when errors occur. This allows you to maintain a consistent look and feel within your application even when errors occur. It also allows you to optionally suppress the display of error information.

CFERROR is often used with CFTRY, CFCATCH, and CFTHROW to customize error pages in ColdFusion applications.

See Generating Custom Error Messages (CFERROR) for more information.

Web server security integration

You can integrate your applications with the user authentication and security provided by your Web server. In addition, the ColdFusion Server offers a security framework that controls access to applications, pages, data sources, and users. You set the bounds of a security domain using the CFAUTHENTICATE tag.

See Integrating with web server security for more information.

Establishing Application-Level Settings

When any ColdFusion application page is requested, ColdFusion searches up the page's directory tree for an `Application.cfm` file. When it is found, the `Application.cfm` code is logically included at the beginning of that page.

If it is not found, ColdFusion searches up the directory tree until it finds an `Application.cfm` file. If more than one `Application.cfm` file lives in the current directory tree, ColdFusion uses the first one it finds.

Note ColdFusion continues searching for the `Application.cfm` file up to the root directory of the hard drive where the web root resides.

Advantages of using the Application Framework

Because the `Application.cfm` page is processed before any other application page, it gives you the ability to set application-level parameters, and to perform queries and other functions.

Since `Application.cfm` files are standard ColdFusion application pages, you may use CFML code within them to dynamically determine the values of application-level settings based on queries, client-state information, and so on.

By defining a scope for one or more applications, `Application.cfm` files can be implemented for individual or multiple applications. A single `Application.cfm` file, installed in your ColdFusion application root directory, can globally control settings in multiple applications nested below the root.

Note ColdFusion Server scope variables are available to all application pages. This is a potential source of conflict with variables set in `Application.cfm` pages. See How ColdFusion looks up variables in Chapter 4, "Creating and Manipulating Variables," on page 31 for information on the order in which ColdFusion finds variables.

The flexibility of ColdFusion allows for a number of different ways to configure ColdFusion applications. The following sections describe a basic approach that will work for most cases.

Defining an application

An important step in designing a ColdFusion application is mapping out its directory structure.

Before you start building the application, establish a root directory for the application. Application pages may also be stored in sub-directories of the root directory.

Note In UNIX, the references to `Application.cfm` are case-sensitive. The filename must be spelled with an initial capital.

To set up an application framework:

1. Create an `Application.cfm` file in the root of the application directory for application-level settings.
2. Set application-level settings and error handling.
3. Enable client state management with the CFAPPLICATION tag.
4. Enable any optional features you want, such as security (using CFAUTHENTICATE), locking (using CFLOCK), or error handling (using CFERROR).

For details on setting up user security in the `Application.cfm` file, see the user security information in *Advanced ColdFusion Development*.

Establishing an application root directory

All of the page files in an application do not need to be in the same root directory. However, defining a root directory for an application has a number of advantages:

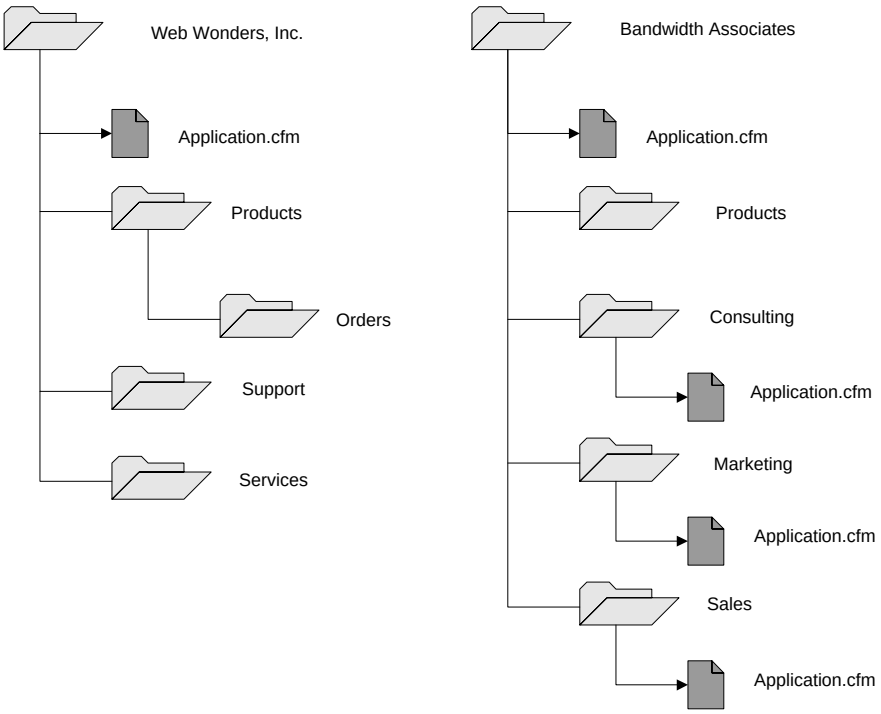
- Development: The application is easier to develop and maintain because the application page files are well organized.
- Portability: The application can be more easily moved to another server or another part of a server without having to change any code in the application page files.
- Application-level Settings: Application pages that fall under the same root directory can share application-level settings and functions.
- Security: Application pages that fall under the same directory can share Web server security settings.

You can use a single `Application.cfm` file for your application, or use many different `Application.cfm` files that govern individual sections of the application.

Application scope example

The directory trees below illustrate two approaches to implementing the Application Framework.

- In the first example, a company named Web Wonders, Inc. uses a single `Application.cfm` file installed in their application root directory to process all application page requests.
- The illustration on the right shows how Bandwidth Associates uses the settings in individual `Application.cfm` files to specify processing for ColdFusion applications at the departmental level. Only the Products application pages are processed using the settings in the root `Application.cfm` file. The Consulting, Marketing, and Sales directories each has its own `Application.cfm` file.



Behavior with CFINCLUDE

Only one `Application.cfm` file is ever processed for each ColdFusion application page. The presence of an `Application.cfm` file is an implicit CFINCLUDE. If it is present in the directory tree, there is no way not to include it. (For this reason, it is the ideal location to set application-level variables.)

When the requested application page has a CFINCLUDE tag pointing to an additional application page, ColdFusion does not initiate another search up the directory tree based on the included application page.

This is an important behavior to understand. Upon opening a requested application page, ColdFusion searches for the `Application.cfm` file only once.

Using Application and Session Variables

In ColdFusion, you use variables to work around the Web's inherent statelessness. Session and application variables are persistent variable "scopes." You access these variables by prefacing the variable name with the scope name, for example: *Session.MyVariable* or *Application.MyVariable*. And because they are persistent, you can pass values between pages with a minimum of effort.

Enabling application and session variables

Session and application variables are similar in operation to client variables. Like client variables, they are enabled with the CFAPPLICATION tag. See the *CFML Language Reference* for more information on the CFAPPLICATION tag.

Unlike client variables, however, which are stored in either the system registry, a data source, or a cookie, application and session variables are always stored in the ColdFusion server's memory. This method offers obvious performance advantages. In addition, you can set time-out values for these variables either with CFAPPLICATION, or by specifying time-outs in the ColdFusion Administrator. You can also simply disable application and session variables entirely.

For information on setting time-outs for variables, see *Administering ColdFusion Server*.

Session variables

Use session variables when the variables are needed for a single site visit or set of requests. For example, you might use session variables to store a user's selections in a shopping car application. (Use client variables when the variable is needed for future visits.)

How long do sessions last?

A session refers to all the connections that a *single client* might make to a server in the course of viewing any pages associated with a given application. This logical view of a session begins with the first connection by a client and ends (after a specified time-out period) after that client's last connection.

It's important to understand that sessions are specific to individual users. As a result, every user has a separate session and has access to a separate set of session variables.

It's also important to understand that a session is a logical construct. Because of the stateless nature of the Web, it's not always possible to define a precise point at which a session ends. In the real world, a session ends when the user finishes using an application and goes off to do something else. In most cases, however, a web application has no way of knowing if a user is finished or if he's just lingering over a page.

To impose some structure where there is none, session variables have a programmer-specified time-out period associated with them. 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.

Session variable time-outs

Session variables have a specific lifetime, and it is this lifetime that defines a "session." For example, when you access a session variable within its specified lifetime, the variable returns a value because your request occurs during a single, time-limited

session. On the other hand, if you do not access a session variable within its specified lifetime, the variable will time-out and will no longer be available to you.

The default time-out for session variables is set to 20 minutes. In the Variables page of the ColdFusion Administrator, you can change this time-out value. See the *Administering ColdFusion Server* book for more information.

You can also set the time-out period for session variables inside a specific application (thereby overruling the Administrator default setting) by using the SESSIONTIMEOUT attribute of the CFAPPLICATION tag.

Using 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. Using session variables, an application could initialize itself with user-specific data the first time a user hit a page of that application. This information could then remain available while that user continues to use that application. For example, information about a specific user's preferences could be retrieved from a database once, the first time a user hits any page of an application. This information would remain available throughout that user's session, thereby avoiding the overhead of retrieving the preferences again and again.

Session variables work exactly as client variables do, in that they require a client name (client ID) and are always scoped within that client ID. Session variables also work within the scope of an application name if one is supplied, in which case their scope will be the combination of the client ID and the application name.

Because session variables no longer require client variables to be turned on, the following read-only variables have been added to the session variable scope: CFID, CFTOKEN, and URLTOKEN.

To enable session variables, set `SESSIONMANAGEMENT="Yes"` in the CFAPPLICATION tag in your `Application.cfm` file.

Example

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

```
    MAILTO='webmaster@mysite.com'>
<CFERROR TYPE='VALIDATION' TEMPLATE='val_err.cfm'
    MAILTO='webmaster@mysite.com'>

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

Application variables

An application refers to all the related web pages that make up a particular piece of functionality. In ColdFusion, you define an application by giving it a name using the CFAPPLICATION tag. By using the same application name in a CFAPPLICATION tag, you definte a set of pages as being part of the same logical application.

Note The value you set for the NAME attribute in CFAPPLICATION is limited to 64 characters.

Using application variables

Application variables are designed to store application-level data. They are a convenient place to store information that all pages of your application might need no matter who (what client) is running that application. Using application variables, an application could initialize itself, say, when the first user hit any page of that application. This information could then remain available indefinitely to all subsequent hits of any pages of that application, by all users, 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 ColdFusion Server is shut down, application variables are very convenient. However, because all clients running an application see the same set of application variables, they are not useful for client-specific information. To target variables for specific clients, use session variables.

Application variables require an application name be associated with them and are always scoped within that application name.

Unlike client and session variables, however, they do not require that a client name (client ID) be associated with them. Thus, they are available to any clients that specify the same application name.

Application variable time-outs

Application variables have a specific lifetime, and this lifetime defines an "application." For example, when you access an application variable inside a specific

application, the variable returns a value because your request occurs on a page declared in the CFAPPLICATION tag to be part of a single application.

The default time-out period for application variables is two days. In the Variables page of the ColdFusion Administrator, you can define time-out values for application and session variables. See *Administering ColdFusion Server* for more information.

You can set the time-out period for application variables within a specific application (thereby overriding the default setting in the ColdFusion Administrator) by using the APPLICATIONTIMEOUT attribute of the CFAPPLICATION tag.

If no clients access the application within the specified time-out period, ColdFusion Server destroys its application variables.

Differentiating client, session, and application variables

This table shows the relationships among client, session, and application variables:

Kinds of Variables					
Variable Type	**Application Names**	**Client IDs**	**Client Mgmt**	**Session Mgmt**	**Time-out**
Client	Optional	Required	Required	n/a	Optional
Session	Optional	Required	Required	Required	Optional
Application	Required	n/a	n/a	n/a	Optional

Variable scopes are required

ColdFusion does not attempt to automatically evaluate Application and Session variables. You must use variable prefixes with these variables, as in Session.variablename or Application.variablename.

Tips for using session and application variables

In general, session and application variables are designed to hold information that you seldom write but are read often. In most cases, the values of these variables are set once, most often when an application is first started (Application variables) or the first time a user begins using an application (Session variables). Then the values of these variables will be referenced many times throughout the life of the application or the course of a session.

When using application variables, keep in mind that these variables are shared by all instances of an application that might be running on a server. Because of this sharing, applications cannot assume that values saved in these variables will not be overwritten by other instances of the same application that might be simultaneously running on the server. Of course, this is not a problem if these variables are treated as “write-once, read-many,” but can be a problem if they are written to indiscriminately.

Example

Here is an example of the use of several application variables. Note that they are used within the `Application.cfm` file since this is a natural place to perform an operation that needs to be done every time any of the application's pages are run. In this example, an application variable, *Application.Initialized*, is used as a flag to indicate whether or not the application variables have been initialized.

```
<!--- Declare a name for this application. This
automatically turns on application scope --->

<CFAPPLICATION NAME="AccountCheck">

<!--- Test to see if application variables
have already been defined --->

<CFIF NOT IsDefined("application.initialized")>
    <CFSET application.query1=   ??? >
    <CFSET application.query2=   ??? >
    <CFSET application.initialized=1>
</CFIF>
```

Managing session and application variables

When you install ColdFusion, the default time-out for session variables is set to 20 minutes and for application variables, two days. There are two ways to manage application and session variable expiration. You can override these values by specifying different time-out values in the CFAPPLICATION tag. Or you can redefine the default time-out values in the ColdFusion Administrator.

To enable session and application scopes, and to define time-out options, open the Variables page of the ColdFusion Administrator. You can specify individual time-out settings for each variable type. Although these variables occupy very little of ColdFusion's available memory, you have the option of disabling their use altogether, or limiting their use with time-outs. See *Administering ColdFusion Server* for more information about using the Variables settings.

Getting a list of application and session variables

The variable scope names “application” and “session” are registered as ColdFusion structures. This enables you to use the ColdFusion Structure functions to get a list of application and session variables. For example, you can use CFLOOP with the StructFind function to output a list of application and session variables defined for a specific application.

To find a list of client variables, you use the GetClientList function.

See the *CFML Language Reference* for more information on these functions. See the chapter Working with Structures in the *Advanced ColdFusion Development* book.

Client Variables and Client State Management

The Web is a stateless system. Each connection that a browser makes to a Web server is unique in the eyes of the Web server. However, within an application it is important to be able to keep track of users as they move through the pages within the application. This is the definition of client state management.

ColdFusion achieves client state management by creating a client record for each browser that requests an application page in an application in which client state management is enabled. The client record is identified by a unique token that is stored in an HTTP cookie in the user's browser.

The application can then define variables within the client record. These client variables are accessible as parameters in every application page that the client requests within the scope of the application.

Enabling Client State Management

To set up client state management, each page in the application must contain a CFAPPLICATION tag that sets the name of the application and enables the CLIENTMANAGEMENT attribute. The best place to put this tag is at the beginning of the `Application.cfm` file, which is included in all of the application's pages. This way client state management is enabled for every page in the application.

For example, to enable client state management for the sample Products application, add this CFAPPLICATION code to the beginning of the `Application.cfm` file:

```
<CFAPPLICATION NAME="Products" CLIENTMANAGEMENT="Yes">
```

All application pages at and beneath this `Application.cfm` file in the directory structure are now able to use client state management.

Note Client state management can work without cookies as well. See Using client state management without cookies in this chapter.

Choosing a client variable storage method

The system-wide default is for client variables to be stored in the system registry. But your site administrator can choose to store them instead in a SQL database or in cookies.

There are two steps to changing client variable storage: by setting a client variable storage option in the Variables page of the ColdFusion Administrator, and then by noting the client variable storage location in the CFAPPLICATION tag. See *Administering ColdFusion Server* for information on using the ColdFusion Administrator.

You use the CLIENTSTORAGE attribute in the CFAPPLICATION tag to specify where you want to store client variables, providing one of three values:

- `Registry`
- The name of a configured client store

- `Cookie`

The following example shows how you enable client state management using a sample database called *mydatasource*.

```
<CFAPPLICATION NAME="myapplication"
    CLIENTMANAGEMENT="Yes"
    CLIENTSTORAGE="mydatasource">
```

If no ClientStorage setting is specified, the default location, as noted in the ColdFusion Administrator Variables page, is used.

Note Client storage mechanisms are exclusive; when one storage type is in use, the values set in other storage options are unavailable.

Cookie storage

When you set `CLIENTSTORAGE="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. Chief among them is that if the client turns off cookies in the browser, client variables won't work.

Consider these additional limitations before implementing cookie storage for client variables:

- Netscape Navigator allows only 20 cookies from a particular host to be set. ColdFusion uses two of these cookies for CFID and CFTOKEN, 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 host.
- Netscape Navigator sets a size limit of 4K bytes per cookie. ColdFusion encodes non-alpha-numeric 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 will throw an error if you try to store more than 4000 encoded bytes of data for a client.

Using client state management

When client state management is enabled for an application, you can use the system to keep track of any number of variables associated with a particular client.

Creating a client variable

To create a client variable and set the value of the parameter, use the CFSET or CFPARAM tag.

Example

The following example sets the value of a client variable named *FavoriteColor* to "Red":

```
<CFSET Client.FavoriteColor="Red">
```

Once a client variable has been set in this manner, it is available for use within any application page in your application that is accessed by the client for whom the variable is set.

Example

The following example shows how to use the CFPARAM tag to check for the existence of a client parameter and to set a default value if the parameter does not already exist:

```
<CFPARAM NAME="Client.FavoriteColor" DEFAULT="Red">
```

Using client variables

A client variable is accessed using the same syntax as other types of variables, and can be used anywhere other ColdFusion variables are used.

Example

To display the favorite color that has been set for a specific user, use the following code:

```
<CFOUTPUT>
    Your favorite color is #Client.FavoriteColor#.
</CFOUTPUT>
```

Standard client variables

In addition to storing custom client variables, the Client object has several standard parameters. These parameters can be useful in providing customized behavior depending 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>
```

The standard Client object attributes are read-only (they can be accessed but not set by your application) and include CFID, CFToken, URLToken, HitCount, TimeCreated, and LastVisit.

Using client state management without cookies

You can use ColdFusion's client state management without cookies. However, we do not recommend this course. Developers who choose to maintain client state without cookies must ensure that every request carries CFID and CFTOKEN.

To maintain client state without cookies, set the SETCLIENTCOOKIES attribute of the CFAPPLICATION tag to No. Then, the developer must maintain client state in URLs. by passing the client ID (CFID)and the client security token (CFTOKEN) between pages, either in hidden form fields or appended to URLs. You accomplish this using the variable Client.URLTOKEN or Session.URLTOKEN.

Note In ColdFusion, client state management is explicitly designed to work with cookies, the standard tool for identifying clients. Using client state management without cookies requires careful programming to ensure that the URLToken is always passed between application pages.

Getting a list of client variables

To obtain a list of the custom client parameters associated with a particular client, use the GetClientVariablesList function.

```
<CFOUTPUT>#GetClientVariablesList()#</CFOUTPUT>
```

The GetClientVariablesList function returns a comma-separated list of variable names defined for the application context declared by CFAPPLICATION, if any. The standard system-provided client variables (CFID, CFToken, URLToken, HitCount, TimeCreated, and LastVisit) are not returned in the list.

Deleting client variables

Unlike normal variables, client variables and their values persist over time. (In this fashion they are akin to cookies.) To delete a client variable, use the DeleteClientVariable function. For example:

```
<CFSET IsDeleteSuccessful=DeleteClientVariable("MyClientVariable")>
```

The DeleteClientVariable function operates only on variables within the scope declared by CFAPPLICATION, if any. See the *CFML Language Reference* for more information on this function.

Also, through the Variables page of the ColdFusion Administrator, you can edit the client variable storage to remove client variables after a set number of days. (The default value is 90 days when client variables are stored in the registry, ten days when stored in a data source.)

See *Administering ColdFusion Server* for more information about setting time-out values.

Note The system-provided client variables (CFID, CFToken, URLToken, HitCount, TimeCreated, and LastVisit) cannot be deleted.

Client variables with CFLOCATION behavior

When using CFLOCATION to redirect to a path that contains .dbm or .cfm, the Client.URLToken is automatically appended to the URL. This behavior can be suppressed by adding the attribute ADDTOKEN="No" to the CFLOCATION tag.

Variable caching

All client variable reads and writes are cached to help decrease the overhead of client state management operations. See the *Adminstering ColdFusion Server* book for information on variables and server clustering.

Exporting the client variable database

If your client variable database is stored in the 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 allows you to export and import registry entries.

To export your client variable database from the registry:

1. Open the registry editor. In UNIX, use the program, `/<install_dir>/coldfusion/bin/regedit`.
2. Find and select the following key:
   `HKEY_LOCAL_MACHINE\SOFTWARE\Allaire\ColdFusion\CurrentVersion\Clients`
3. On the Registry menu, click Export Registry File.
4. Enter a name for the registry file.

Once you've created a registry file, you can take it to a new machine and import it by selecting Import Registry File on the Registry Editor Registry menu.

For more information on using variables in ColdFusion applications, see Chapter 4, "Creating and Manipulating Variables," on page 31 in this book.

Default Variables and Constants

It is often useful to set default variables and application-level constants in the `Application.cfm` file. For example you may want to designate:

- A data source you're using
- A domain name
- Style settings such as fonts or colors
- Other important application-level variables

Example: Application.cfm

The following example shows a complete `Application.cfm` file for the sample Products application:

```
<!--- Set application name and enable client
variables option, with client variables stored in
a data source called mycompany --->

<CFAPPLICATION NAME="Products"
    CLIENTMANAGEMENT="Yes"
    CLIENTSTORAGE="mycompany">

<!--- Install custom error pages --->
```

```
<CFERROR TYPE="REQUEST"
    TEMPLATE="RequestErr.cfm"
    MAILTO="admin@company.com">

<CFERROR TYPE="VALIDATION"
    TEMPLATE="ValidationErr.cfm">

<!--- Set application constants --->

<CFSET HomePage="http://www.mycompany.com">
<CFSET PrimaryDataSource="CompanyDB">
```

Using CFLOCK for Exclusive Locking

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

Using CFLOCK around CFML constructs that modify shared data ensures that the modifications occur one after the other and not all at the same time. As a general rule, you should use the CFLOCK tag to perform updates to variables in the Application, Server, and Session scopes in the `Application.cfm` file.

Example

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

<HTML>
<HEAD>CFLOCK Example</HEAD>

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

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

</BODY>
</HTML>
```

See the *CFML Language Reference*for more information on using CFLOCK.

Generating Custom Error Messages (CFERROR)

ColdFusion displays error pages that can help you to debug your application. There are two types of errors in ColdFusion:

1. REQUEST — Request errors occur when a application page is requested and there is an error in the page's code.
2. VALIDATION — Validation errors occur when a user violates the form field validation rules during a form submittal.

By default, ColdFusion returns a standard page for these errors. But you may want to customize the error pages that are returned, to make them consistent with the look and feel of your application. Custom error pages also allow you to control the error information that users see, as well as offering work-arounds or ways for users to report the errors.

You set the custom error application pages with the CFERROR tag. You can set the custom error application pages page-by-page, but because custom error pages generally apply to an entire application, it is more efficient to include the CFERROR tag in the `Application.cfm` file. After you create a custom error page, you must include the CFERROR tag in your application's `Application.cfm` page.

For information on the syntax of the CFERROR tag, see the *CFML Language Reference*.

Creating an error application page

The error application page is a file that includes HTML and the parameters associated with the error. The error application page cannot use any CFML tags.

The parameters associated with an error depend on the type of error. All the error parameters use the Error prefix (for example, `Error.Diagnostics`).

See the *CFML Language Reference* for more information on the error variables and on using the CFERROR tag.

The following examples show the two types of custom error pages.

Example of a request error

The following example shows a custom error page for a request error:

```
<HTML>
<HEAD>
    <TITLE>Products - Error</TITLE>
</HEAD>
<BODY>

<CFOUTPUT>
<H2>Sorry</H2>
```

```
<P>An error occurred when you requested this page.
Please email the Webmaster to report this error. We
will work to correct the problem and apologize
for the inconvenience.</P>

<TABLE BORDER=1>
<TR><TD><B>Error Information</B> <BR>
    #Error.DateTime# <BR>
    #Error.Template# <BR>
    #Error.RemoteAddress# <BR>
    #Error.HTTPRefer#
</TD></TR></TABLE>

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

Example of a validation error

The following example shows a custom error page for a validation error.

```
<HTML>
<HEAD>
    <TITLE>Products - Error</TITLE>
</HEAD>
<BODY>

<H2>Oops</H2>

<P>You failed to complete all the fields
in the form. The following problems occurred:</P>

#Error.InvalidFields#

</BODY>
</HTML>
```

Application Security

ColdFusion also offers a security framework that lets you apply security to pages, applications, and users. First you use the Advanced Security page of the ColdFusion Administrator to set up a security context that governs access to resources for authenticated users. Then you use the CFAUTHENTICATE tag in your `Application.cfm` file to authenticate users.

For details and examples of securing your application pages, see the Application Security chapter in *Advanced ColdFusion Development*.

Integrating with web server security

If the application pages in your application are located in the Web server document directory, you can use your Web server's native authentication and encryption services to secure your ColdFusion application. Each Web server has a different way of configuring security settings, creating users, groups, and establishing privileges. Consult your Web server documentation for instructions on configuring your server's settings.

Authentication

In ColdFusion, you can use the CFAUTHENTICATE tag in the `Application.cfm` file to establish a security domain for your application. See the Application Security chapter in the *Advanced ColdFusion Development* book for information on using CFAUTHENTICATE to establish a user's authentication state.

You can also use your Web server's authentication system. When the Web server authenticates a user, it returns a unique variable that is available within your application page as the CGI Environment parameter "Auth_User." You can use this parameter to access additional information about a user out of a database. In general, it is more straightforward to simply organize your security on an application page level.

Encryption

ColdFusion offers two functions to encrypt strings in application pages: Encrypt() and Decrypt(). See the *CFML Language Reference* for information on using these functions.

Because ColdFusion returns Web pages to the Web server, you can also use your Web server's encryption technology to encrypt the pages in your applications. This is especially useful for commerce applications that require a higher level of security.

For more information on securing your application pages, see the Application Security chapter in the *Advanced ColdFusion Development* manual. Also, for information on using the security features in the ColdFusion Administrator, see the security chapters in *Administering ColdFusion Server*.

CHAPTER 7

Debugging and Troubleshooting

There are several ways to debug your ColdFusion pages and track errors. This chapter describes how to use the Interactive Debugger in ColdFusion Studio to set breakpoints and debug dynamic pages against the ColdFusion Server.

It also describes the debugging output that is written as a footer to the pages generated by ColdFusion.

Contents

Using the Interactive Debugger in ColdFusion Studio

Developers can use the interactive debugger in ColdFusion Studio to debug dynamic pages against the ColdFusion application server.

You can use the debugging tools to:

- Set breakpoints and watches
- Evaluate variables and expressions
- Step through lines of code
- Investigate the code stack
- Monitor recordsets
- Observe variables in all scopes

To run the interactive debugger, you use the Debug menu or the commands on the Debug toolbar in ColdFusion Studio. But first, you need to set up the debugger with the correct paths to both your application pages and your ColdFusion server.

The following sections describe these procedures in detail.

Getting Started

First, you need to take the following steps to set up the debugging session:

- Select the ColdFusion server against which you'd like to run the debugging session.
- Provide mappings to specify the ColdFusion Studio path, the ColdFusion Server path, and the browser's path to the files you'd like to debug.
- Specify the first page in the application to be debugged.

Configuring a Remote Development Server

To access a remote server in ColdFusion Studio, you must first configure the server and provide security login information if the server is protected through ColdFusion security.

To configure a remote development server:

1. In ColdFusion Studio, open the first application page you would like to debug and choose Debug > Debug Settings.
2. In the Debug Start pane of the Remote Development Settings dialog box, choose Add RDS Server from the drop-down list.
3. In the Configure RDS Server dialog box, enter the description, host name, user name, password, and port of the Remote Server. You can also choose whether or not to use Secure Sockets Layer to access the server.

4. If your remote server is protected by ColdFusion RDS Security, enter the user name and password you use to access this protected resource.
5. Click OK.

Creating RDS Mappings

Because the interactive debugger runs in ColdFusion Studio, against the ColdFusion Server and displaying to a Web browser, you must specify file mappings for the directory where the application lives.

Before you start debugging an application, you need to know the following mappings:

- **Studio Path** — How does ColdFusion Studio see the directory?
- **Server Path** — How does the Web server/ColdFusion server see the directory?
- **Browser Path** — How does the browser see the directory?

An important part of setting up the debugger is creating mappings that inform the debugger how the local paths you use in ColdFusion Studio translate into server paths and URLs.

File mapping examples

The following scenarios show how file mappings work when you have local or remote files matched with either local or remote servers:

- ColdFusion Studio and Server on the same machine
- ColdFusion Studio debugging on remote ColdFusion Server using drive mappings
- ColdFusion Studio debugging on remote ColdFusion Server using Network Neighborhood
- ColdFusion Studio debugging on remote ColdFusion Server using RDS file access

ColdFusion Server and Studio on the same machine

Debugging against a local ColdFusion Server is the most common scenario. In most cases, this arrangement allows both the Server and Studio to see the directories in the same way.

For example, the local path `c:\inetpub\wwwroot` translates to an identical server path `C:\inetpub\wwwroot`, and a URL path of `http://215.180.21.1/`. The use of mappings in such a scenario is mainly for URL resolutions. The URL part of the mapping instructs ColdFusion Studio how a physical file can be viewed in a browser.

In this example, you would create a mapping as follows:

ColdFusion Server and Studio on same machine	
Studio Path	`C:\inetpub\wwwroot\`
Server Path	`C:\inetpub\wwwroot\`
Browser/URL Path	`http://215.180.21.1/`

Studio accesses files on a remote ColdFusion server using drive mappings

Developers often debug against a remote ColdFusion server across an internal network. In many cases, they use a network drive mapping.

For example, a user may have a remote drive `X:\` mapped to a network shared directory `\\MYSERVER\WEBPROJECTS\` where `WEBPROJECTS` is the name of the shared directory in the network server `MYSERVER`.

In such a scenario, a file that appears to Studio as

`X:\App1\Index.cfm`

May be viewed by the CF server as

`C:\webprojects\App1\index.cfm`

The browser may view it using the URL path

`http://215.180.21.1/App1/index.cfm`

In order to resolve the communication between ColdFusion Studio and the Server, you need to create a mapping for the App1 directory as follows:

Studio accesses files on remote ColdFusion server using drive mappings	
Studio Path	`X:\App1\`
Server Path	`C:\webprojects\App1\`
Browser/URL Path	`http://215.180.21.1/App1/`

Studio accesses files on a remote ColdFusion server using UNC paths/Network Neighborhood

Developers can debug code against remote ColdFusion servers across an internal network. They often use the Network Neighborhood to access a file on a remote Cold Fusion server. For example, a developer may be accessing a file on `\\myserver\webprojects\` where `webprojects` is the name of the shared directory in the network server `myserver`.

In such a scenario, a file that appears to Studio as

```
\\myserver\webprojects\App1\Index.cfm
```

May be viewed by the ColdFusion server as

```
c:\webprojects\App1\index.cfm
```

The browser may view it using the URL path

```
http://215.180.21.1/App1/index.cfm
```

ColdFusion Studio and the Server need to understand how a file location appears to the parties involved. You therefore need to create a mapping for the App1 directory as follows:

Studio accesses files on remote CF server using UNC paths/Network Neighborhood	
Studio Path	`\\MYSERVER\WEBPROJECTS\App1\`
Server Path	`C:\webprojects\App1\`
Browser/URL Path	`http://215.180.21.1/App1/`

Studio accesses files on a remote ColdFusion server using RDS-based remote file access

When developing outside local area networks, many developers access files on a ColdFusion Server across the Internet using the RDS-based remote file access available from the Remote tab in ColdFusion Studio.

In such a scenario, a file that appears to Studio as

```
RDS://MY_RDS_SERVER/C:/webprojects/App1/index.cfm
```

May be viewed by the ColdFusion server as

```
C:\webprojects\App1\index.cfm
```

The browser may view it using the URL path

```
http://215.180.21.1/App1/index.cfm
```

Although the server path can be inferred from the local RDS path, you still need to create a mapping. In special scenarios, ColdFusion Server to Studio path resolution could become ambiguous. You therefore need to create a mapping for the App1 directory as follows:

Studio accesses files on remote CF server using RDS remote file access	
Studio Path	`RDS://MY_RDS_SERVER/C:/WEBPROJECTS/App1/`

Studio accesses files on remote CF server using RDS remote file access	
Server Path	`C:\webprojects\App1\`
Browser/URL Path	`http://215.180.21.1/App1/`

Specifying server mappings

The debugger in ColdFusion Studio relies on these mappings to communicate to the ColdFusion server the correct file paths of all files with breakpoints. You create these mappings in the Mappings pane of the Remote Development Settings dialog box.

To specify server and file mappings for debugging:

1. Choose Debug > Development Mappings to open the Mappings pane of the Remote Development Settings dialog box.
2. In the ColdFusion Server list box at the top of the dialog box, choose the ColdFusion server against which you'll run the debugging session.
3. In the Studio Path box, enter the file path ColdFusion Studio uses for the page you're debugging. Click Add.

   If you're debugging against a local server, the Studio and ColdFusion Server paths are the same. So if you chose *localhost*, for example, in the ColdFusion Server list box above, the CF Server Path is the same as the Studio path you just entered.
4. If you're debugging against a remote server, enter the CF Server Path.

   The CF Server path needs to be the same as the alias or virtual mapping your web server uses to access the file.
5. Specify the Browser Path, or URL, of the application and click Add.
6. Click OK.

Running the Interactive Debugger

Running the debugger helps you find problems in code by tracing the way ColdFusion evaluates the page, step by step. You do this by placing breakpoints in the page to stop execution of the page, and then step through subsequent lines of code, checking the values of variables and expressions to make sure the code is behaving as you'd expect.

After you've set up remote development settings and file mappings, you can use the commands on the Debug menu and the Debug toolbar to run the debugger.

See Chapter 3, "ColdFusion Studio Quick Start," on page 19 for a list of keyboard shortcuts for the debugger.

Note You cannot debug encrypted templates.

To run the interactive debugger:

1. In ColdFusion Studio, open the first page in the application you want to debug.

   The Debug toolbar appears at the bottom of the application window. You can undock the toolbar by double-clicking on the undock bars on the left-hand side of the toolbar, drop it into the Quick Bar, or keep it docked.

2. In your ColdFusion application page, set breakpoints in your code by clicking in the gutter on the left side of the editor window.

   The breakpoint lines are colored red. Choose Debug > Toggle Breakpoints to turn breakpoints on and off. Use the Debug > Clear Breakpoints to delete all breakpoints in the current document.

3. In the Debug toolbar, hit the Start/Continue tool.

   The Remote Development Settings dialog box appears. It shows the default local server.

4. Specify the Start URL, which is the URL you use to view the page, and click OK. The list box shows the pages currently open in ColdFusion Studio.

   Enter a fully qualified file path relative to your local ColdFusion server -- for example, http://127.0.0.1/SomeServerPath/index.cfm.

   The Debug Start dialog box displays every time you press the Start button in the Debugger. Use the check box on the Debug Start pane to disable the display of this box the next time you hit the RUN button.

   Whenever possible, ColdFusion Studio tries to calculate the URLs of open file using the RDS development mappings, which you can set in the Mappings tab. These mappings are also required to develop on remote servers.

   Based on the RDS Security settings at your site, you'll see a login prompt to access protected resources.

5. Enter a user name and password if necessary to access remote servers.

   The debugger starts, and it forwards your URL to the Browse view. When the ColdFusion server encounters the breakpoint, a blue bar appears at the breakpoint in the Edit view. At this point, the ColdFusion server has an open session, and it waits for your reply from ColdFusion Studio.

6. To continue, hit the Start/Continue button again, and the server moves to the next breakpoint.

   The final time you hit the Start/Continue button, ColdFusion executes the page and outputs it to the browser.

7. To stop the debugging session, hit the End button.

Note The debugger is active until you hit the End button on the Debug toolbar or choose Debug > End. This allows you to debug across multiple pages.

Debug windows

Choose View > Debug Window to open the Debug window. There are several panes to this window:

- **Variables** — Displays all scopes of local variables.
- **Watches** — Use this window to set watches and evaluate expressions and variables.
- **Recordsets** — Displays the list of recordsets initialized in the current application page. This pane tracks both CFQUERY-based database recordsets and dynamic recordsets generated programmatically.
- **Output** — Shows the output of the page as it is being generated.
- **Breakpoints** — Shows all the breakpoints that you have set in all files. You can view, disable, and remove breakpoints from this window, and edit their conditions.
- **Tag Stack** — Shows a hierarchy of tag and page attributes, and values.

Each of these windows has an associated command button on the Debug toolbar.

You can undock these windows individually, so you can see breakpoints while you're looking at the Watches, for example.

Stepping through code

You can use the Step Into, Step Over, Step Out, and Run to Cursor tools on the Debug toolbar to step through your code.

- Step Into proceeds to the next unit of execution in your code. Use this command to walk through code line-by-line.

  If the next step is inside an included file or CFX, the debugger steps into the next file.
- Step Over operates in the same way as Step Into, except that it does not step into included files (CFINCLUDE, CFMODULE, or CFLOCATION) or custom tags. It executes the included code but does not trace through the included code step-by-step.
- Step Out is used when you are inside a block of included code and want to step back to the point in your original page where you entered the included code.
- Use Run to Cursor to execute up to the cursor position, without having to set a breakpoint. Note that cursor location must be beyond you. If there are breakpoints between your location and the cursor, Run to Cursor will stop at these intervening breakpoints.

Evaluating expressions and setting watches

You can use the evaluator box at the top of the Watches pane of the Debug window to evaluate arbitrary expressions when the debugger is suspended at a breakpoint. Use the evaluator when you want to know how an expression evaluates as you step through code, line by line.

Watches allow you to evaluate the same expression or variable every time you stop execution. When you set a watch, the debugger evaluates the watched expression. A hand pops up when the expression's value changes from one line to the next as you step through code.

To set watches:

1. Choose Debug > Watches or click the Watches button on the Debug toolbar. The Watches pane appears.
2. Cut and paste the expression or variable you want to watch into the list box at the top of the pane.
3. Choose Evaluate to find the value of the expression at the next breakpoint or line where the Debugger stops.

   The Evaluator window shows the results of the evaluation at the current point in processing the page.
4. Choose Watch to add the expression in the evaluator list box to the list of watched expressions.

   The Watch area shows the values of watched expressions and any error messages in resolving these parameters.
5. Press the Start/Continue button to continue debugging.
6. When you are finished debugging, press End.

Note You can use the evaluator to change values of variables, create new variables, or practice using ColdFusion functions in your expressions.

Debugging across multiple pages

The debugger is active even after a page is loaded, and until you press the End button on the Debug toolbar or choose Debug > End.

This enables you to debug applications across multiple HTML and CFML pages. For example, you can test the submission of an HTML form and its subsequent processing by a ColdFusion application page.

Debug Settings in ColdFusion Administrator

ColdFusion can provide important debugging information for every application page requested by a browser. When enabled, debug output is shown in a block following normal page output. The debug output can help you track down programming problems.

You can select from the following debug output options:

- Show variables
- Show processing time
- Show SQL and data source name
- Show query information

Note By default, when you enable any of these options, debug output becomes visible to all users. You can, however, restrict debug output to a selected IP address.

For information on the debugging and logging settings in the ColdFusion Administrator, see the *Administering ColdFusion Server* book.

In the Debugging page of the ColdFusion Administrator, click the boxes to enable these output options:

Debug Settings in ColdFusion Administrator	
Setting	**Description**
Enable performance monitoring	Allows the standard NT Performance Monitor application to display information about the current ColdFusion Server. On platforms that do not support the NT Performance Monitor, a command line utility, CFSTAT, is provided to display the same information. (You must restart the ColdFusion Server for changes in this setting to take effect.)
Show variables	Displays the CGI environment variables, Form fields, URL parameters, and cookies passed with the client request.
Show processing time	Displays the time, in milliseconds, it took the ColdFusion Server to process the page request.
Show SQL and data source name	Allows you to view the SQL statement and data source name in use when an error occurs.
Show query information	Displays the number of records, processing time, and SQL statement for each query executed.

You can limit display of debugging output to selected users by creating a list of IP addresses. Type the addresses in the text box and use the Add and Remove buttons to manage the list.

If no entries are made, the information is displayed to all users. Click Apply to save the settings.

For more information on using the Administrator, see *Administering ColdFusion Server*.

Generating debug information without setting options

You can view the parameters and CGI environment variables for an individual application page without changing the global settings. Simply append the parameter "mode=debug" to the end of the URL.

```
www.myserver.com/cfdocs/test.cfm?mode=debug
```

Generating debug information for an individual query

You can view debug information for an individual query by putting the DEBUG attribute into the opening CFQUERY tag:

```
<CFQUERY NAME="TestQuery" DATASOURCE="MyDB" DEBUG>
    SELECT * FROM TestTable
</CFQUERY>
```

When this query runs, it places the debug information into the output page where the query is placed.

Error messages

If ColdFusion is unable to fulfill a request because of an error, it returns a diagnostic message to the user. The message includes a link that allows the user to e-mail a report of the error to the site administrator. You enable this feature in the Mail Logging page of the ColdFusion Administrator. Errors are written to a log file for later review.

Database errors

If a database error occurs, the following information is returned (if enabled in the Administrator Debugging page):

- The ODBC error code and textual explanation of the error code.
- The extended error message returned from the ODBC driver.
- The name of the ODBC data source being used.
- The SQL statement submitted to the database. Note that for Insert and Update actions, this statement includes parameter markers represented by a "?" in place of the data values sent to the database.

Syntax errors

If a syntax error occurs while processing a application page, the following information is returned:

- The line of the application page file on which the syntax error occurred.
- A printout of that line and the lines above and below it.
- An indication of which tokens the CGI script was looking for when it encountered the syntax error.

Other errors

Other errors that may occur are system related. Examples of these errors include:

- Out of memory
- File or disk access errors (disk full or damaged, sharing violation, etc.)

These errors are also reported to the client and written to the log file, but it is often more difficult to provide detailed diagnostics for them. If you get a message that does not explicitly identify the cause of the error, check on key system parameters like available memory and disk space.

For information on using the Logging settings and Mail Logging settings of the ColdFusion Administrator, see the chapter *Configuring the ColdFusion Server* in the *Administering ColdFusion Server* book.

Troubleshooting

The following section describes a few common problems that you may encounter and ways to resolve them.

ODBC data source configuration

Problem: ODBC driver manager cannot make a connection to the database.

The basic requirement for getting ColdFusion to work with an ODBC data source is that the ODBC driver manager establishes a connection to the database. Many ODBC problems result from the inability to connect to the database. Connection errors are difficult to diagnose. They include problems with the location of files, network connections, and database client library configuration.

Before you take any other diagnostic steps or seek technical support, verify that you can connect to the database. You can do this by clicking the Verify button on the ODBC Data Sources page of the ColdFusion Administrator. If you are unable to make a simple connection from that page, you need to work with your database and/or driver vendor to solve the problem.

Problem: Data source does not exist or name is incorrectly specified.

Create data sources before you refer to them in your application source files. Also, check the spelling of the data source name.

HTTP/URL

Problem: The METHOD in forms sent to the ColdFusion server must be Post.

When you invoke a ColdFusion application page from within an HTML form, you must use METHOD="Post" rather than METHOD="Get", which is the default. The METHOD attribute is specified as part of the FORM tag. For example:

```
<FORM ACTION="test.cfm" METHOD="Post">
```

If you do not use METHOD="Post", ColdFusion cannot correctly decode the contents of your form submission.

Problem: URLs cannot have embedded spaces.

Many browsers complain when you include spaces in URLs. The correct way to do this is to use a plus sign (+) wherever you want to include a space. ColdFusion correctly translates the + sign into a space.

A common scenario in which this error occurs is when you dynamically generate your URL from database text fields that may 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 + signs. See the *CFML Language Reference* for information on using this function.

CFML syntax errors

Problem: An end tag is omitted.

It is a common error to omit the end tag for the CFQUERY, CFOUTPUT, CFTABLE, or CFIF tag. If you get an error message you don't understand, the first thing you should do is make sure all your CFML tags have matching end tags where appropriate.

When developing pages in ColdFusion Studio, use the Tag Completion feature, which adds an editing tag each time you create an opening tag.

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 ColdFusion syntax validation tools in ColdFusion Studio.

Problem: Mismatched quotes and escape characters.

Check strings in attributes and expressions for proper placement of single and double quotes.

CFML Syntax Checker

Version 4.0 of the Cold Fusion Application Server features stricter enforcement of CFML syntax rules. Strict checking can uncover hidden bugs and other types of undesirable behaviors in your ColdFusion application pages. Allaire recommends that you always use the strictest possible level of CFML validation.

In rare cases, the more relaxed validation mechanisms used by previous versions of Cold Fusion may have allowed you to use syntactically incorrect CFML constructs. The CFML Syntax Checker is a simple application that can aid you in the process of discovering which of your CFML templates may not conform to the rules CFML 4.0.

The CFML Syntax Checker is available in your ColdFusion installation directory at `install_dir/cfdocs/cfmlsyntaxcheck.cfm`.

For more information, see the online documentation in the CFML Syntax Checker application.

CHAPTER 8

Understanding Data Sources

This chapter provides an overview of database structures and procedures for accessing data sources for ColdFusion applications. It introduces two standards for working with databases — Open Database Connectivity (ODBC) and the Structured Query Language (SQL).

We'll also take a look at a powerful productivity tool called Query Builder. It is part of ColdFusion Studio and is designed for viewing data sources and building SQL statements.

Contents

Data Source Basics

For ColdFusion developers, the term "data source" can refer to a number of different types of structured content accessible locally or across a network. You can query Web sites, LDAP servers, POP mail servers, and documents in a variety of formats.

Most commonly though, a database will drive your applications and in this section a data source is defined as the entry point for database operations.

Open Database Connectivity (ODBC)

ODBC is the standard interface for connecting to a data source from an application. The application must have an ODBC driver installed and configured for each data source. You can check your system's installed drivers by opening the ODBC Data Source Manager in the Windows Control Panel.

You can learn more about ODBC and download the ODBC 3.0 Programmer's Reference at http://www.microsoft.com/data/odbc/kill/download.htm.

Installing ODBC drivers

You can install the Microsoft Data Access Components (MDAC 2.0) from the ColdFusion menu on the Windows Start menu. The installed set of ColdFusion ODBC drivers includes:

- Microsoft SQL Server
- Microsoft Access and FoxPro databases
- Borland dBase-compliant databases
- Microsoft Excel worksheet data ranges
- Delimited text files

Databases

A thorough knowledge of databases is not necessary for developing a ColdFusion application that interacts with a database, but you will need to learn some basic concepts and techniques.

A database is a structure for storing units of information. Database management systems (DBMS) standardize the collection and maintenance of data and provide an interface for retrieving data.

The data in a database is organized in tables, which are collections of information about particular items. Those items can be individuals, products, or any entities that can be uniquely identified. A table consists of a grid of columns and rows. A column defines a unit of data such as a name, date, or zip code. Data is entered on each row under the appropriate column headings. A single row constitutes one data record because the data in that row applies to a unique item. Data can be organized in multiple tables to fill a wide range of user needs in an efficient manner. This type of

data structure is known as a relational database and is the type used for all but the simplest data sets. The structure of a database is called the schema.

From this basic description, a few database design rules emerge. First, each record must contain a unique identifier, known as the primary key. This could be an employee ID, a part number, or a customer number. This is typically the column used to maintain each record's unique identity among the tables in a relational database. Second, once a column has been defined to contain a specific type of information, the data must be entered in that column in a consistent way. This is accomplished by defining a data type for the column, such as allowing only numeric values to be entered in the zip code column. Third, 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.

For example, most companies maintain a database of their customers. At a minimum, this database contains columns (or fields) for information such as the customer's name, address, and phone number. Additional information on orders, payments, shipping instructions, and so on may be included as part of an order processing application. Sales information for each customer order can be stored and used to analyze sales trends and to determine sales commissions. How this data is organized is a function of the size and scope of the data itself and the complexity of the intended database operations.

The best way to familiarize yourself with the capabilities of your database product or DBMS is to review the product documentation.

Adding an ODBC Data Source

Assuming there is a data source available against which your ColdFusion application is to run, you need to identify it as a ColdFusion data source. To do this, open the Data Sources ODBC page of the ColdFusion Administrator.

To add a data source:

1. Enter a name for the new data source.
2. Select the appropriate ODBC driver from the list.
3. Click Add to open the Create ODBC Data Source page.
4. Enter information for the new data source in the appropriate fields. Click CF Settings to display optional settings.
5. Click Create. The new data source is added to the system and displays on the Data Sources page. To edit data source information, click on the data source name in the list.
6. To test the data source, open the Verify Data Source page and select it from the list.

You can now work with the data source by coding data operations in your application pages.

For more information about managing data sources, see *Administering ColdFusion Server*.

Tip ColdFusion Studio provides a set of tools to make your work with data sources more productive.

ODBC Naming Conventions

The formatting of table and field names varies considerably across data sources. As a working standard for data sources, ODBC defines a naming convention that developers can follow in database construction. Adhering to the standard makes your code more portable, giving you increased deployment flexibility and significantly decreasing the cost of scaling applications to higher-performance database systems.

The standard requires that names begin with a letter and consist only of letters, numbers, and the underscore character. ColdFusion uses the same standard for identifying query columns. Therefore, the tables and queries you use with ODBC and ColdFusion must also adhere to this standard.

Although ODBC allows spaces in table names, ColdFusion does not. You must use aliases to access tables and field names that contain spaces. In addition, some data sources require you to specify table owner and table qualifier information to access the data source.

It is possible to sidestep the ODBC/ColdFusion naming requirements by enclosing references to nonstandard names with the backquote (‘) character and by using the SQL “AS” keyword to alias nonstandard names into standard names. However, these techniques are burdensome and lead to the creation of non-portable code that is significantly less readable.

Table owners and qualifiers

The ODBC data sources for which ColdFusion bundles desktop drivers generally allow you to access their tables by simply naming them in the SQL statement. A principal exception is SQL Server, which requires extended information.

The following statement is intended to return all the records from a SQL Server table called Orders. To accomplish this, the table name must be preceded by the database name (`orderdb`) and the database owner (`dbo`).

```
SELECT *
    FROM orderdb.dbo.order
```

Other client/server databases like Oracle and Sybase have their own requirements. Check the product documentation for the correct qualifiers.

The Microsoft Excel driver requires the backquote (`) and dollar sign ($) to qualify a worksheet name in a statement. The example below returns all the rows from a worksheet called "`sheet1`" in an Excel workbook that has been added as an ODBC data source.

```
SELECT *
    FROM 'sheet1$'
```

Using Native Database Drivers

Drivers written specifically for accessing data sources directly from ColdFusion can be installed with the Enterprise version of the ColdFusion Application Server. Because they do not need to route SQL queries through the ODBC Driver Manager, they offer a performance advantage for large-scale data operations.

Open the ColdFusion Administrator Datasources Native Driver page to manage the drivers.

Bundled drivers

The current set of native drivers contains:

- Oracle 7.3 — requires Oracle 7.3.3 client
- Oracle 8.0 — requires Oracle 8.0.4.0.0
- Sybase System 11 — requires Sybase 11.1.1 client libraries. Sybase patch ebf 7729 is recommended

Attributes for enabling native drivers

You can use the attributes listed below to override settings on the Administrator Native Driver page:

- DBSERVER — Enter the Sybase or Oracle server name.
- DBNAME (Sybase only) — Enter the database name.
- BLOCKFACTOR (ODBC and Oracle only) — Set the maximum number of rows to return for each query. The default value is 1, the maximum value is 100. Certain ODBC drivers may dynamically reduce the block factor at runtime.

These attributes are supported in the CFQUERY, CFINSERT, CFUPDATE, CFGRIDUPDATE, and CFSTOREDPROC tags.

Note For Sybase SQL Server the syntax for calling a stored procedure with CFQUERY is different for ODBC and the native Sybase driver. ODBC requires brackets and the word "call". The native driver does not have this requirement.

ODBC syntax:

```
<CFQUERY DATASOURCE="" NAME="">
{call dbo.stored_procedure_name}
</CFQUERY>
```

Native driver syntax:

```
<CFQUERY DATASOURCE="" NAME="">
dbo.stored_procedure_name
</CFQUERY>
```

Using OLE DB Connectivity

The OLE DB driver is included with the Professional and Enterprise versions of ColdFusion on Win32 platforms.

OLE DB data stores

CF developers can now access a range of new data stores, including:

- MAPI-based data stores such as Microsoft Exchange and Lotus Mail
- Non-relational data stores, such as Lotus Notes
- LDAP 2.0 data
- Data from OLE applications like word processors and spreadsheets
- Mainframe data
- HTML and text files, flat-file data

Enabling this capability requires installation of OLE DB *providers* available from third-party vendors. The provider software handles data processing in response to requests from the *consumer*, in this case ColdFusion.

After installing and configuring the provider software, you can open the CF Administrator Data Sources OLE DB page and add data stores supported by that provider. As with other CF data sources, you enter a name and description and select advanced settings as needed. You will also need to enter:

Provider — the ProgID

ProviderDSN — the data source name

The data source displays on the main OLE DB page and can be used in queries.

OLE DB providers for SQL data stores

Performance gains can be made by running an OLE DB provider, instead of an ODBC driver, to process SQL. This is dependent on how the provider implement the data call. Some providers route OLE DB calls through the ODBC Driver Manager, while others go directly to the database. These latter are akin to native drivers in providing an alternative to ODBC. Providers are available for all the major relational DBMS products as well as the data stores listed above.

OLE DB is a Microsoft specification. For more information, including a list of provider vendors, go to their OLE DB site.

Structured Query Language (SQL) Overview

SQL is the standard language for performing database operations. Its syntax is relatively simple, yet it is powerful enough to handle complex data tasks. SQL commands are encapsulated in the CFQUERY tag to perform data operations in ColdFusion. The simplicity of this mechanism is one of the principal reasons why ColdFusion is so popular with developers of data-driven Web applications.

Resources

While a working knowledge of SQL is an important asset for ColdFusion development, a thorough discussion of SQL syntax and usage is beyond the scope of this user guide. A number of useful resources can be found at the Online SQL Meta Reference.

SQL syntax overview

A SQL *statement* is a command made up of *clauses* that specify the operation to perform, the data source, and any instructions needed to complete the operation. Each clause must begin with a *keyword*.

Here's a simple example of a SQL statement enclosed in a ColdFusion query:

```
<CFQUERY NAME="zip_02140"
    DATASOURCE="Customer">
    SELECT first_name, last_name, phone
        FROM Customer
        WHERE zip_code = 02140
        ORDER BY last_name
</CFQUERY>
```

The statement uses the SELECT keyword to search the database. The first line of the statement consists of a keyword followed by the *identifiers* of the data columns to be retrieved. The second line is a clause that names the data source, in this case the Customer table. The third clause uses the equal sign (=) *operator* to set a *condition* that limits the records returned to the specified zip code. The fourth clause sorts the result set by the specified column.

This query returns a sorted list of the first name, last name, and phone number of every record in the Customer table that has the value "02140" in the zip code column. The result set can then be presented in an HTML table using the CFOUTPUT tag.

SQL statements also can be used to manipulate the records in a database, create and remove data objects, and to run administrative tasks such as setting user access to data sources. The complete SQL syntax is available in ColdFusion, though most ColdFusion applications focus on queries and data maintenance operations.

Note Some DBMS vendors use non-standard SQL syntax (known as a dialect) in their products. ColdFusion does not validate the SQL in a CFQUERY, so you are free to use any syntax that is supported by your data source. Check your DBMS documentation for non-standard SQL usage.

Syntax elements

The following sections present brief descriptions of the main SQL command elements.

Statements

These keywords identify commonly-used SQL commands:

- SELECT — Retrieves the specified records.
- INSERT — Adds a new row.
- UPDATE — Changes values in the specified rows.
- DELETE — Removes the specified rows.

Statement clauses

These keywords are used to refine SQL statements:

- FROM — Names the data source 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

These specify conditions and perform numeric functions:

Operator	Description
AND	Both conditions must be met, such as Paris AND Texas
OR	At least one condition must be met, such as Smith OR Smyth
NOT	Exclude the condition following, such as Paris NOT France
=	Equal to
<>	Not equal to
<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to
+	Addition
-	Subtraction

Operator	Description
/	Division
*	Multiplication

SQL Extensions

This section describes the ODBC extensions to SQL, which enable a high degree of code portability across data sources.

Several extensions to SQL grammar are defined by ODBC. These extensions make ODBC SQL more portable across data sources by providing a uniform syntax for features that vary across database systems. The most commonly used ODBC SQL extensions include those defined for date/time specification, scalar functions, and stored procedures.

Date/Time Specifications

ODBC defines a universal format for specifying date and time values. If you need to include a literal date or time value inside a SQL statement, always use this format. The formats for date, time, and date/time values are as follows:

Value	Format	Example
Date	{ d 'yyyy-mm-dd' }	{ d '1995-06-20' }
Time	{ t 'hh:mm:ss' }	{ t '15:34:08' }
Date/Time	{ ts 'yyyy-mm-dd hh:mm:ss' }	{ ts '1995-06-20 15:34:08' }

When you use the ColdFusion Insert and Update actions, date fields are automatically validated and converted to the ODBC date format, so you do not need to format the user's input. However, when you submit SQL statements directly using a CFQUERY tag, be sure you properly format all dates and times.

One way to achieve this for dates entered by users is to apply a date validation rule to the appropriate form field. This ensures that a valid date is entered and the date is automatically converted to the ODBC format.

Scalar Functions

Many database drivers allow you to call functions within SQL statements. These functions typically perform numeric calculations, string manipulation, date/time manipulation, and the retrieval of system information. Because the names and syntax of these functions are often different across data sources, ODBC defines a set of data source-independent scalar functions.

The syntax for calling ODBC scalar functions is as follows:

```
{ fn scalar-function }
```

where *scalar-function* refers to the function name and its arguments enclosed in parentheses. The following examples illustrate how two commonly used scalar functions are used in SQL statements. The first example illustrates calling CURDATE to obtain the current date:

```
SELECT *
    FROM Orders
    WHERE ShipDate>={ fn CURDATE() }
```

The second example illustrates calling LEFT to get the left-most 5 characters of a postal code entered by a user:

```
SELECT *
    FROM Customers
    WHERE PostalCode={ fn LEFT('#Form.PostalCode#', 5) }
```

The ODBC 3.0 Programmer's Reference is a good source of information on the extensive set of available functions.

CHAPTER 9

Selecting and Presenting Data

This chapter explains how to select and output data in a dynamic Web page. It shows how to use a number of CFML tags to query a database and then present the results of that query in a Web page.

Contents

Selecting Data with the CFQUERY Tag

To select and output data from a database, you create a ColdFusion application page that includes both CFML and HTML. The application page contains sections for selecting the data and for outputting the results of the selection.

This section introduces ColdFusion data source queries using the Structured Query Language (SQL) in a CFQUERY tag.

The first step is to define a query to select data from a database. ColdFusion uses the CFQUERY tag to define queries. The full syntax for the CFQUERY tag is:

```
<CFQUERY NAME="query_name"
    DATASOURCE="ds_name"
    USERNAME="username"
    PASSWORD="password"
    MAXROWS="number"
    TIMEOUT="milliseconds"
    BLOCKFACTOR="1" (default)
    DBPOOL="database connection pool name"
    DEBUG="yes/no">
        SQL statements
</CFQUERY>
```

Note If a SQL statement becomes too long to fit on a single line, you can split it across multiple lines.

Example

Here is a simple example from a database called Company associated with an ODBC data source named CompanyDB.

To create a query called "EmployeeList" that retrieves all of the records in the Employees table, use this syntax:

```
<CFQUERY NAME="EmployeeList" DATASOURCE="CompanyDB">
    SELECT *
    FROM Employees
</CFQUERY>
```

Using Dynamic Query Parameters

You can harness the real power of the CFQUERY tag when you dynamically customize the contents of a SQL statement by embedding dynamic parameters in the application page. Dynamic parameters (also called variables) include form fields, parameters passed in the URL, and CGI environment information.

The convention for including a dynamic parameter inside a SQL statement is to enclose it in pound signs, such as #State#. When ColdFusion reads text enclosed by # signs, it searches through all Form, URL, cookies, client, and CGI variables looking for one that matches the specified name. When it finds the name, it substitutes the

appropriate value for the parameter reference. To fully identify a variable, enter the variable type with the variable name, such as #Form.State#.

Sources for dynamic parameters

The following table summarizes the primary sources from which you can draw dynamic parameters for use in your SQL queries:

Sources for Dynamic Parameters	
Source	**Description**
Form fields	The most common way of passing parameters to an application page. When a user enters data in a form field, a parameter bearing the name of the form field (`#Form.formfield#`) is passed to the application page.
URL parameters	Parameters that are embedded on the end of a URL (such as, `/input.cfm?name=adam`).
Server variables	A variable that remains available to all application pages until the ColdFusion application server terminates.
CGI environment	An environment variable interpreted by the browser. Every request sent to an application page has several environment variables that relate to the context in which it was sent. The variables available depend on the browser and server software in use for a given request, such as `CGE.UserAgent.`
Query objects	Query columns you can reference once a query has been executed. Once a query has been run, its results can be used as dynamic parameters in other queries. For example, a query that returns a column called UserID can be referenced in the following form: `queryname.UserID`
HTTP Cookies	General mechanism for storing and retrieving information about the Web client (browser).
Client variables	Used to store persistent client variables in the system registry on the Web server. These variables are specific to an individual browser accessing your ColdFusion application.
Session variables	Variables available only for an individual session. Session variables are tied to an individual client and persist for as long as that Client ID maintains a session.
Application variables	Variables available only for an individual application. Application names are defined in the CFAPPLICATION tag, which is typically used in the `Application.cfm` file.

Example: Dynamic SQL

If you created a form to allow end users to search for employees by last name, you could use the following SQL statement with dynamic parameters:

```
SELECT *
    FROM Employees
    WHERE LastName = '#Form.LastName#'
```

If the user entered "Rucker" for LastName, the SQL statement sent to the database would be:

```
SELECT *
    FROM Employees
    WHERE LastName = 'Rucker'
```

Using single quotes around literal text

You will notice that the parameter Form.LastName was surrounded by single quotation marks (‘). These are necessary for all literal strings of alphanumeric characters. You do not need to use quotation marks around numbers. You might wonder what happens when a parameter value that contains one or more single quote characters is substituted into a section of a SQL statement delimited by single quotes. Wouldn't this cause a SQL syntax error?

Not if handled properly. ODBC allows you to denote a single quote inside a quote-delimited string by using two consecutive single quote characters (''). Since you can't possibly be expected to do this yourself, ColdFusion automatically replaces the single quote (') with two single quotes ('') before including parameter values in SQL statements.

Note In special cases, you might want to suppress automatic escaping of single quotes. To do this, use the PreserveSingleQuotes function.

Caching Query Results

The ability to save query result sets in server memory is a significant performance enhancement for all ColdFusion sites, especially those supporting a high volume of repetitive data access.

To call cached query data, the new query must match exactly any of the following values that it uses: have the same data source, query name, DBTYPE, SQL statement, user name, and password as the cached data set. If you are using native drivers, the query syntax must also match the DBSERVER and DBNAME (Sybase only) values.

Two CFQUERY attributes control the cache implementation:

- CFCACHEDAFTER — Sets a date to check before running a new query
- CFCACHEDWITHIN — Sets a time range to check using the CreateTimeSpan function

The cache is self-regulating once the maximum number of stored queries is set. That is, when the limit is reached, the oldest result set is replaced by the current one.You set the appropriate number of queries to cache on the ColdFusion Administrator Server Settings page.

In setting the optimal cache value, consider such factors as available system resources and the time required to process complex queries versus storing the result sets in memory.

The decision point for enabling the cache and for choosing which attribute to set is the level of tolerance you (and your users) have for the timeliness of query results. For important record sets that are frequently updated, a short time interval is recommended for maintaining the cache. Generally, the more static the record set, the greater the time interval that can be safely set.

Executing Stored Procedures

Many database systems allow you to create stored procedures to return result sets for commonly-used data queries. Check your DBMS documentation for details of your system's implementation of this feature.

This section describes two methods of coding access to stored procedures:

- Specifying call parameters in a CFQUERY
- Using CFSTOREDPROC and its associated tags

Calling stored procedures from CFQUERY

You can call stored procedures from within CFQUERY, as in the example below. This query calls a SQL Server stored procedure that retrieves all orders due to ship on the date specified:

```
<CFQUERY NAME="GetOrdersForDate"
    DATASOURCE="Orders Database">
        { call OrderDB.dbo.sp_getorders( #OrderDate# ) }
</CFQUERY>
```

While this method is still available, we recommend using the CFSTOREDPROC tag to add greater flexibility and control to stored procedure operations.

Calling stored procedures from CFSTOREDPROC

The CFSTOREDPROC tag wraps SQL call parameters in tag attributes to simplify the process of retrieving data and to add functions not available in a CFQUERY call statement. It supports both ODBC and native data sources.

CFSTOREDPROC is used to identify the stored procedure and its data source and to set options. Two additional tags are nested within it:

- CFPROCPARAM — Set the data and procedure types, variables and values, and other options
- CFPROCRESULT — Identifies the result set for output and optionally limits the result set if multiple sets are returned by the stored procedure

Before deciding which of the two stored procedure methods to employ, answer these questions:

Do I need to specify input/output parameters for the query?

Do I need to return a result code for the query?

Do I need to select from multiple result sets in a stored procedure?

If you answer Yes to all three of these questions, you should be using CFSTOREDPROC in your applications.

Date Formatting Functions

When dates are returned by ColdFusion, they might not be in precisely the format in which you want it displayed. ColdFusion provides special formatting functions that allow you to format dates for output.

Date, time, and number formatting functions

The following functions are available for manipulating dates, times, and numbers:

Date, Time, and Number Formatting Functions	
Function	**Description**
`DateFormat(Date[,mask])`	Case-insensitive. Creates a custom-formatted U.S. English date value. Use LSDateFormat for international date formats.
`TimeFormat(Date[,mask])`	Case-sensitive. Create a custom-formatted time value.
`NumberFormat(Number[,mask])`	Create a custom-formatted number value.

Note Brackets indicate optional arguments.

If the mask has no sign specifier, positive and negative numbers will not align in columns. Thus, if you expect to have both positive and negative numbers, either use the ( ) or use the "-" sign, which will force a space in front of positive numbers and a minus in front of negative numbers.

Tip To achieve the correct column spacing on the output, use the <PRE> tag.

Special formatting functions

The functions listed in the following table provide special formatting options.

Formatting Functions	
Function	**Description**
HTMLCodeFormat	Useful for display of HTML code posted using TEXTAREA fields. Strips carriage returns and escapes all special characters (<, >, ", &).
HTMLEditFormat	Behaves identically to HTMLCodeFormat except that it does not add the <PRE> tag to the output text.
ParagraphFormat	Useful for displaying data entered into TEXTAREA fields. Converts CR/LF sequences into spaces and double CR/LF sequences into HTML paragraph markers (<P>).

PreserveSingleQuotes	Useful in SQL statements to prevent ColdFusion from automatically escaping single quotes contained in values derived from dynamic parameters. For example, to include a dynamic parameter in a SQL statement and suppress the escaping of single quotes, use the syntax:
	`SELECT * FROM Customers`
	`WHERE CustomerName`
	`IN (#PreserveSingleQuotes(CustNames)#)`

StripCR	Useful for preformatted (PRE) display of data entered into TEXTAREA fields. Strips all carriage returns from the field.
URLEncodedFormat	URL encodes the string parameter that is passed to it (replaces spaces with a "+" and non-alphanumeric characters with equivalent hexadecimal escape sequences). This function enables you to pass arbitrary strings (including those with spaces in them) within URLs. (ColdFusion automatically decodes all URL parameters that are passed to an application page.)

Formatting Functions (Continued)	
Function	**Description**
ValueList & QuotedValueList	The two output functions ValueList and QuotedValueList facilitate using the results of queries to drive subsequent queries. The ValueList function takes as its argument the name of a query column, such as Customers.CustomerID, and returns a comma-separated list of the values for each of the records in the query for that column. For example, if you run a query that returns four distinct customer records, the result of ValueList function would be like: 22,43,51,96. The QuotedValueList function would return '21,' '43,' '51,' '96' for the same data.
YesNoFormat	Displays Boolean data as Yes or No. All non-zero values display as Yes. Zero values display as No.

The next section covers the presentation of data with the CFOUTPUT tag.

Displaying the Query Result Set

Once you've created a CFQUERY in your application page file, you can then reference its results within other CFML tags. The query results can be used to dynamically create an HTML page.

As you learn to use CFML tags in application page files, keep in mind that you can also use HTML tags and text in application page files. Wherever you use standard HTML tags and text inside your application page, ColdFusion simply passes the tags and text directly back to the client browser.

The most flexible way to display data retrieved from a CFQUERY is to define a CFML output section in your application page file using the CFOUTPUT tag. Output is generated for each record in a result set. Output can be linked to a specific query or contain content from multiple queries. A CFOUTPUT tag can contain:

- Literal text
- HTML tags
- References to query columns
- References to dynamic parameters like form fields
- Functions

Basic output code has the following syntax:

```
<CFOUTPUT QUERY="queryname" MAXROWS=n >
    Literal text, HTML tags, and
    dynamic field references (e.g., #FullName#)
</CFOUTPUT>
```

Database field names in CFOUTPUT sections

Some databases, such as Microsoft Access, allow field names to contain embedded spaces, as in, "Region Name." ColdFusion does not support references to these fields in CFOUTPUT sections. Aside from maintaining compatibility with ColdFusion, avoiding the use of spaces in table names is a sound practice, since it maximizes the portability of your work across database systems. You can use an underscore as a separator, such as "Region_Name"

Field names must always begin with a letter and contain only alphanumeric characters. You can use the SQL keyword AS to alias field names. For example, the statement

```
SELECT 401K AS FK FROM Employee_Withholding
```

will be processed correctly in a CFQUERY.

Example: CFOUTPUT

Use the CFOUTPUT tag to display data from the result set of the EmployeeList query in a browser. For this example, we've chosen to display the first name, last name, and phone number of each employee. It's good programming practice to specify the query name as part of the variable name.

The following example shows the complete code for the application page:

```
<!--- Query to select customers --->
<CFQUERY NAME="EmployeeList" DATASOURCE="CompanyDB">
    SELECT *
    FROM Employees
</CFQUERY>
<HTML>
<HEAD>
    <TITLE>Employee List</TITLE>
</HEAD>

<BODY>

<H2>Employee List</H2>

<!--- Output section --->
<CFOUTPUT QUERY="EmployeeList">
    <HR>
    #EmployeeList.FirstName# #EmployeeList.LastName#
    (Phone: #EmployeeList.PhoneNumber#) <BR>
</CFOUTPUT>

</BODY>
</HTML>
```

You could call this application page using a standard URL reference:

```
http://myserver/cfdocs/employeelist.cfm
```

The application page reference can be in a hyperlink, as well.

```
<A HREF=”myserver/cfdocs/employeelist.cfm">Employee List</A>
```

The output, formatted using the HTML <HR> and
 tags, would look like this:

```
<HR>
Deborah Jones (Phone: 612-227-1019) <BR>
<HR>
John Smith (Phone: 507-452-7224) <BR>
<HR>
Frank Wilson (Phone: 612-831-9555) <BR>
```

Nested CFOUTPUT and grouping

You can nest CFOUTPUT tags to create grouped displays of output, much like the grouping features of most database report writers. Grouping is accomplished by adding the GROUP attribute to a CFOUTPUT tag and then nesting another CFOUTPUT tag within the first. The formatting instructions above and below the inner tag display the group header and footer information. The formatting instructions in the inner CFOUTPUT tag display record detail information.

Example: Grouping

The following code uses the CFOUTPUT GROUP attribute to display the query results based on the “CourseLevel” value of the query. The ORDER BY keyword of the SQL statement sorts the result set by the CourseLevel field. There is no limit to the number of CFOUTPUT statements that you can nest together. If you want to use multiple levels of grouping, you need to set multiple levels of sorting in your SQL query (e.g., “ORDER BY Region, State”).

```
<CFQUERY NAME="Courses" DATASOURCE="CourseDB">
    SELECT *
    FROM CourseList
    WHERE Department_ID = '#Form.Department#'
    ORDER BY CourseLevel
</CFQUERY>

<CFOUTPUT QUERY="Courses" GROUP="CourseLevel">
<H4>#CourseLevel#</H4>
    <UL>
    <CFOUTPUT>
    <LI> #CourseNumber# - #CourseName#
    </CFOUTPUT>
    </UL>
</CFOUTPUT>
```

The area between the outer CFOUTPUT containing the GROUP attribute and the inner CFOUTPUT containing the “CourseNumber” and “CourseName” parameters contains

the text and formatting for the header of each section. Correspondingly, the area below the inner CFOUTPUT contains the text and formatting for the footer of each section.

The output from this application page is three course level headings (Basic, Intermediate, and Advanced). Each level contains an unordered list of the courses offered at that level.

The final output in a Web browser would look like this:

```
Basic
              100 - Physiology
Intermediate
              510 - Neurobiology
              500 - Plant Biology
Advanced
              820 - Neurobiology
              800 - Microbiology
```

CFQUERY properties

Each CFQUERY you execute has three properties that you can access to provide record number information.

Record Numbers	
Record Numbers	**Description**
RecordCount	The total number of records returned by the query.
CurrentRow	The current row of the query being processed by CFOUTPUT.
ColumnList	Returns a comma-delimited list of the query columns.

Example: Query columns

To report the number of records returned from the CustomerList query and a list of the query columns use the syntax:

```
<CFOUTPUT>
    The search returned information
    on #CustomerList.RecordCount# customers.<BR>
    Columns queried were #CustomerList.ColumnList#.
</CFOUTPUT>
```

To print a row number next to each record displayed in a query, use the syntax:

```
<CFOUTPUT QUERY="CustomerList">
    #CurrentRow# - #FirstName# #LastName# <BR>
</CFOUTPUT>
```

Returning partial recordsets

For large recordsets you may want to display only a portion of the records retrieved. You can accomplish this with the STARTROW and MAXROWS attributes of the CFOUTPUT tag.

STARTROW designates the first row of the recordset that should be returned in the output. The MAXROWS determines the number of rows that will be returned in total.

Example: Partial recordset

This example shows returning records 10-20 from the recordset created by the EmployeeList query:

```
<CFOUTPUT QUERY="EmployeeList"
    STARTROW="10" MAXROWS="20">
    #FirstName# #LastName# #Phone# <BR>
</CFOUTPUT>
```

Using parameters in CFOUTPUT sections

CFOUTPUT sections are not used exclusively for outputting information returned from queries. You can also use CFOUTPUT sections to display Form, URL, Cookie, Client, Server, Session, Application, and CGI environment parameters. be sure to qualify references to the parameters with the appropriate prefix (Form, URL, or CGI) so that ColdFusion is clear that the parameters are not referring to columns in the query result set. Like column names passed from a query, parameters must be enclosed in pound signs (#). The parameter's value is printed once for every row in the result set. For example, to report the criteria a user entered into the employee search form, use the syntax:

```
<CFOUTPUT>
    <P>The search for #Form.LastName# in the
    #Form.Department# returned these results:</P>
</CFOUTPUT>
```

Using the pound sign in CFOUTPUT sections

Because the # sign serves as a special formatting code for ColdFusion, you need to take special measures when you include it in a CFOUTPUT section. To include a # sign that is not used as a field delimiter, use two consecutive # signs (##).

For example:

```
<CFOUTPUT QUERY="CustomerList">
    Phone ##: #PhoneNumber# <BR>
</CFOUTPUT>
```

When ColdFusion processes the output, the two pound signs that follow the text "Phone" print as a single pound sign.

Presenting Query Results in a Table

Presenting query result sets using simple HTML formatting is usually adequate if the number of records returned is small. However, you might need a more compact and structured display of query results. Because the CFOUTPUT tag can include HTML code, you can use standard HTML table tags to build a table dynamically.

CFML tables

CFML also includes a pair of table tags, CFTABLE and CFCOL, that work together to present query results in an useful tabular format. This format uses the HTML <PRE> tag to precisely control the width and alignment of the columns displayed. The result is a clear, concise rendering of your query results.

Example: CFTABLE

This example uses the CFTABLE and CFCOL tags to present output:

```
<CFTABLE QUERY="Messages" MAXROWS=10>
    <CFCOL HEADER="Subject" WIDTH=25
        TEXT="<I>#Subject#</I>">
    <CFCOL HEADER="User Name" WIDTH=15
        TEXT="#UserName#">
    <CFCOL HEADER="Date" WIDTH=15
        ALIGN=RIGHT
        TEXT="#DateFormat(DateSubmitted)#">
</CFTABLE>
```

This code creates a table with three columns labeled “Subject,” “User Name,” and “Date.” These labels appear only if you specify at least one of the HEADER attributes. The table draws its data from the CFQUERY named “Messages” and shows no more than 10 rows. The columns display according to the HTML tags and dynamic parameters contained in their associated TEXT attribute.

Double-quotes and pound signs in the CFCOL attribute

Since ColdFusion uses the double-quotes (") and the pound sign (#) to delimit the TEXT values of the CFCOL attribute, both require special syntax. To specify a single double-quote, use two double-quotes (""). To specify a single pound sign, use two pound signs (##).

For example, to present a list of documents and provide a hyperlink to each document, you must include a hypertext anchor tag (which uses double-quotes to delimit its HREF attribute) within the TEXT attribute. To accomplish this, use the syntax:

```
TEXT="<A HREF=""/documents/#DocFile#"">#DocName#</A>"
```

When ColdFusion processes the code, it replaces the two double-quote pairs surrounding the HREF attribute with a solitary double-quote pair.

HTML tables

Even though the preformatted tables produced by the CFTABLE tag are effective, HTML tables are far more elegant and flexible. To make this transition easier, another CFTABLE attribute called HTMLTABLE is available. This attribute renders a CFTABLE as an HTML table rather than as a preformatted table.

Using the HTMLTABLE attribute changes the interpretation and applicability of some CFTABLE and CFCOL attributes:

- The HEADERLINES and COLSPACING attributes of the CFTABLE tag are ignored.
- The WIDTH attribute of the CFCOL tag is interpreted as the percentage of available screen width, rather than as the number of characters.

Note Displaying result sets using HTML tables does not require the use of the CFTABLE tag with the HTMLTABLE attribute. You can also use the CFOUTPUT tag to construct HTML tables. For maximum control over the appearance of your tables, the CFOUTPUT tag is clearly a superior approach.

Dynamic display of record detail information

The most commonly used format for displaying the results of a search is a table such as a CFTABLE or other highly compact listing. This type of listing is excellent for the purpose of quickly browsing records but is often limited in the amount of information conveyed.

The solution to this problem is to display records in a compact form and then create a hypertext link from each record to another page that provides more detailed information. This type of master-detail approach is explained, including how to implement this functionality from within both CFOUTPUT and CFTABLE tags.

Creating links to detail records

Creating a link to detail records involves the steps below, assuming you already have an application page file that displays a summary list of records.

To create a link to detail records:

1. Select a key piece of information in the summary record to serve as the clickable link to the detail information such as the company name field in a list of firms.
2. Enclose the text you want to use as the clickable link in a hypertext link anchor (HREF) that calls an application page file that displays detail information. The URL of this anchor should pass a unique record identifier to the next application page so it knows which record to display detail information for.
3. Implement a ColdFusion page file that displays the detail information.

Example: Record detail

Suppose you want to display a list of companies and let the user click on a company's name to get more detailed information on the company. Further assume that each company is represented in the database by a unique *Company_ID* and that you have an application page file called `compinfo.cfm` that can display information on a company given a passed *Company_ID*.

In this case, your CFOUTPUT section would look similar to the example below:

```
<CFOUTPUT Query="CompanySearch">
    <A HREF="compinfo.cfm?Company_ID=#Company_ID#">
    #CompName#</A><BR>
    #ContactPerson# (#ContactPhone#) <BR>
</CFOUTPUT>
```

The CompanyName field displays as a hypertext link, which invokes the application page file `compinfo.cfm` with the *Company_ID* as a URL argument. This application page then displays the appropriate detail information for the company.

Use this technique whenever you want to implement dynamic display of detail records. The general form of the technique is as follows (note once again that the HREF should be contained on a single line in your application pages):

```
<A HREF="TemplateFile?Record_ID=#Record_ID_Data#">#RecordName#</A>
```

Here, *TemplateFile* is the name of the application page that implements the record detail view. This application page uses the *Record_ID_Data* (passed as the URL parameter *Record_ID*) to determine which record to display. The user can click the text *RecordName* in the main listing to invoke the application page.

Embedding detail links in CFTABLE columns

You can also use this technique for embedding record detail links inside the TEXT attribute of CFCOL tags. Because the TEXT attribute is delimited by quotes and the HREF attribute of the anchor is also delimited by quotes, be sure that you escape the HREF quotes properly.

The general form of the technique is modified as follows to work properly inside a TEXT attribute:

```
<A HREF=""TemplateFile?Record_ID=#Record_ID_Data#"">#RecordName#</A>
```

Note that there are now two quotes surrounding the HREF attribute of the anchor. If the earlier example were modified to display inside a TEXT attribute, it would look like this:

```
TEXT="<A HREF=""compinfo.cfm?Company_ID=#Company_ID#"">#CompName#</A>"
```

Once again, be sure you keep the entire TEXT attribute on a single line.

Tip You can create simple drill down search applications that use the techniques explained in this section with one of the ColdFusion Web Application Wizards. To run a wizard, click the Web Application Wizards icon in the ColdFusion program group.

Creating an HTML Query Form

The most common way to create dynamic parameters is with a form serving as the user interface. You can build HTML forms to include user entry text boxes, select boxes, radio buttons, and checkboxes for users to enter information. Forms are easy to create and offer a number of flexible data input options. The form variables that are automatically created when a form is submitted can be used to create queries to your data source.

In addition to HTML forms, you can also create forms using the ColdFusion CFFORM tag. See Chapter 12, "Building Dynamic Java Forms," on page 161 if you want to build custom forms that contain graphical elements like tree view controls, slider controls, and grid display controls.

Example: Query form

The following example demonstrates a simple HTML form:

```
<FORM ACTION="employeesearch.cfm" METHOD="Post">
    <PRE>
        Last Name: <INPUT TYPE="text"
        NAME="LastName">
        Department:
        <SELECT NAME="Department">
            <OPTION>Accounting
            <OPTION>Administration
            <OPTION>Engineering
            <OPTION>Sales
        </SELECT>
<INPUT TYPE="Submit" VALUE="Search">
    </PRE>
</FORM>
```

The form above has two inputs: LastName and Department. The user can fill in the text area with a last name and select a department from the select list. Each OPTION tag lists a possible selection the user can make. When the submit button is clicked, the ColdFusion page specified in the ACTION attriubte is opened, and the form variables are passed to the page as dynamic parameters.

Suppose the user enters the name "Peterson" and chooses "Sales." When she clicks the submit button, the form variables shown below will be sent to the application page `employeesearch.cfm`:

LastName=Peterson

Department=Sales

Set the form's ACTION and METHOD attributes

You must set the form's ACTION attribute. The ACTION attribute in your HTML form tells the browser which application page to call when the user clicks a submit button. You also must set the form's METHOD attribute to Post. In the following example, the

employeesearch.cfm application page is executed when the user submits the form for processing.

```
<FORM ACTION="employeesearch.cfm" METHOD="Post">
```

Implement data query fields

Creating search fields for an HTML form is very simple. You need only implement the HTML form fields for each database column you want to search. To make your application pages more legible, it is helpful to make the form field names identical to your database column names.

For example, if you have a table called Employees with three columns called FirstName, LastName, and Department, your form fields might look like this:

```
Name: <INPUT TYPE="text" NAME="FirstName">
LastName: <INPUT TYPE="text" NAME="LastName">
Department: <INPUT TYPE="text" NAME="Department">
```

In your forms, you can use the full range of HTML input widgets, including list boxes, radio buttons, check boxes, and multi-line text boxes, in your forms.

REQUESTTIMEOUT URL parameter

When passing a request for a ColdFusion page, you can use the REQUESTTIMEOUT parameter in the URL to specify the number of seconds before the data source connection times out. The REQUESTTIMEOUT parameter overrides the default timeout specified in the ColdFusion Administrator. This option can prevent a data source connection from timing out for operations that need more connection time to complete.

To use this parameter, you pass a URL for an application page with the REQUESTTIMEOUT parameter specifying the number of seconds before timing out, in the following form:

```
http://myserver.com/cfpages?REQUESTTIMEOUT=100
```

This parameter is typically used for administrative tasks such as data source updates and is not necessary for normal processing.

Example: Dynamic SQL

The example application page below, searchform.cfm, can be created as a query form.

```
<HTML>
<HEAD>
    <TITLE>Employee Search</TITLE>
</HEAD>

<BODY>

<FORM ACTION="employeesearch.cfm" METHOD="Post">
    <PRE>
    Last Name: <INPUT TYPE="text" NAME="LastName">
```

```
    Department: <SELECT NAME="Department">
        <OPTION>Accounting
        <OPTION>Administration
        <OPTION>Engineering
        <OPTION>Sales
    </SELECT>
    <INPUT TYPE="Submit" VALUE="Search">
    </PRE>
</FORM>

</BODY>
</HTML>
```

The example application page below (`EmployeeSearch.cfm`) can be created to select and display the search.

```
<!--- CFML application page to implement
    employee search --->
<!--- Query the database --->
<CFQUERY NAME="EmployeeList"
    DATASOURCE="CompanyDB">
    SELECT *
    FROM Employees
        WHERE LastName = '#LastName#'
        AND Department = '#Department#'
</CFQUERY>

<!--- Page header --->
<HTML>
<HEAD>
    <TITLE>Employee Search Results</TITLE>
</HEAD>

<BODY>
<H2>Organization Search Results</H2>

<!--- Summarize search criteria for user --->
<CFOUTPUT>
    <P>The search for #Form.LastName# in
        the #Form.Department#
        returned these results:</P>
</CFOUTPUT>

<--- Display results --->
<CFOUTPUT QUERY="EmployeeList">
<HR>
#FirstName# #LastName# (Phone: #PhoneNumber#)<BR>
</CFOUTPUT>

<!--- Page footer --->
<P>
Thank you for searching the
```

```
employee database!</P>
<HR>

</BODY>
</HTML>
```

Pattern matching searches

When you provide your users the ability to enter text values as part of a search front end, you normally do not want to search for exactly the value they have entered. Rather, you want to search for a value similar to what they entered. In this case, a pattern matching search is appropriate.

To implement a pattern matching search, use the LIKE operator in combination with one or more wildcard characters (represented by "%" in ODBC queries).

For example, to allow users to search for people by filling out a form field called "LastName," use the SQL statement:

```
SELECT *
    FROM Employees
    WHERE LastName LIKE '#LastName#%'
```

This query returns the record of every person whose last name begins with the value entered. For example, a search for "Jon" would return records for people with names beginning with "Jon," including "Jones" and "Jonson."

Returning MIME Content Types (CFCONTENT)

MIME (Multipurpose Internet Mail Extensions) was created for several purposes, one of which was to create an explicit link between data files and applications that are used to view and edit the files. MIME accomplishes this by providing a content type specification with each data file.

On the Web, the exchange of data happens using Hypertext Transport Protocol (HTTP). Both the Web server and the Web client must agree on the content type of the file, and either the client must be able to understand files of that format, or the client needs to find an application that can use the files.

By default, the content type ColdFusion uses when it generates data for the browser is text/html. This is simply an HTML file type. Despite the fact that text/html is the most frequently used content type on the Web, there might be instances when you want to use ColdFusion to dynamically generate documents of other content types.

Example: Returning VRML

To dynamically return a different VRML (Virtual Reality Modeling Language) model based on a parameter passed in the URL, use the syntax:

```
<CFCONTENT TYPE="x-world/x-vrml">
<CFQUERY NAME="GetCyberCafeRoom"
DATASOURCE="CyberCafe">
SELECT VRML_Script
    FROM CyberCafeRooms
    WHERE FloorNumber=#URL.FloorNumber#
</CFQUERY>

<CFOUTPUT QUERY="GetCyberCafeRoom">
    #VRML_Script#
</CFOUTPUT>
```

An important issue to consider when sending non-HTML content types to the client is the browser's ability to understand the data type. All major browsers come with a set of predefined content types, which are either supported by the browser itself or by an auxiliary application often referred to as a viewer. Therefore, it's important that you notify a user in advance when you intend to use a content type that may not be directly supported by their browser.

Example: Populating an Excel spreadsheet

The following example demonstrates a way to use ColdFusion to dynamically generate a spreadsheet that can load directly into Microsoft Excel. In order to do this seamlessly, you must associate the text/tabdelimited content type with your local Microsoft Excel application:

```
<CFCONTENT TYPE="text/tabdelimited">
<CFQUERY NAME="GetFinancialData"
    DATASOURCE="MutualFunds">
    SELECT *
    FROM Funds
    WHERE Fund_ID=#Fund_ID#
</CFQUERY>
<P>Month Value Percentage Gain Estimate</P>
<CFOUTPUT QUERY="GetFinancialData">
#Month# #Value# #PercentageGain# #Estimate#
</CFOUTPUT>
```

The spaces between fields and headers are actually tab characters. Tab characters are required in order for Excel to correctly parse the columns in the table.

Note ColdFusion strips all text output that occurs in an application page before the occurrence of the CFCONTENT tag.

Tip If you just need to quickly add a range of cells from a spreadsheet to a page, you can copy the range and paste it into ColdFusion Studio's Design window. The data is automatically converted to an HTML table.

Example: Returning a file

The following example illustrates how to use CFCONTENT to return a file from the web server. The advantage of this scenario is that you can control access to files. Using

CFCONTENT with the FILE attribute allows you to return files that are not stored in the root of your Web server document directory.

```
<CFCONTENT TYPE="text/tabdelimited"
    FILE="c:\land\topo.txt">
```

Note ColdFusion generates an error if it finds any output after the CFCONTENT in this type of file retrieval operation.

Using CFREPORT for Crystal Reports Output

ColdFusion provides a simple tag syntax to integrate custom report formats created in Crystal Reports 5.0 or later into your application pages. This section describes the process from the ColdFusion side; consult your Crystal Reports documentation for the details of creating and managing reports.

To use this feature, follow these steps:

- Install Crystal Reports on your Web server.
- Copy the Crystal Reports files (`*.rpt`) that you want to use with ColdFusion to your ColdFusion application page directory.
- Add the CFREPORT tag to your application pages.

These steps assume that you have established an ODBC data source for ColdFusion on the Web server. The ColdFusion service calls the Crystal Reports engine to process the CFREPORT tag from a submitted application page. The formatted data set is included in the output to the browser.

Processing report requests from users can be time-consuming. CFREPORT is most efficient when used to publish reports to a static location.

Example: CFREPORT

ColdFusion allows tremendous flexibility in the runtime customization of Crystal Reports, including the ability to dynamically set the selection formula, sort order, target data source, and any custom formula within a report. The parameters you set in the CFREPORT tag take precedence over those in the report file.

Let's start with a simple example not based on any dynamic inputs:

```
<CFREPORT REPORT="d:\reports\myreport.rpt">
</CFREPORT>
```

In most cases, reports that you run will be based on criteria specified by the user or an application. The CFREPORT tag allows the specification of this criteria in a manner similar to the way the CFQUERY tag allows the specification of dynamic SQL statements.

The next example uses a function to determine the application page path. The report displays only those records that match ZIP code and last name criteria entered into an

HTML form. The curly brackets enclosing the value names are required Crystal Reports syntax.

```
<CFREPORT REPORT="#GetDirectoryFromPath
    (Application PagePath)#myreport.rpt"
    {ZipCode}="#Form.ZipCode#" AND
    {LastName} LIKE "#Form.LastName#*"
</CFREPORT>
```

You can use the body of the CFREPORT tag to provide a custom report selection formula using Crystal Reports selection language. The following example illustrates all the tag's attributes. Note the use of the named formula "@Title" to specify the report name from form input.

```
<CFREPORT REPORT="#GetDirectoryFromPath
    (CF_application page_path)#myreport.rpt"
    DATASOURCE="CustomerReports
    USERNAME="TSEliot"
    PASSWORD="#Form.password#"
    FORMULA="#Form.ReportTitle#">
    ORDERBY="ZipCode">
        {ZipCode} = "#Form.ZipCode#" AND
        {LastName} LIKE "#Form.LastName#*" AND
        {FirstName} LIKE "#Form.FirstName#*"
</CFREPORT>
```

As with CFOUTPUT, conditional processing of data using CFIF, CFELSEIF, and CFELSE provides added flexibility to your output.

CHAPTER 10

Using Studio Database Tools

This chapter describes how to use the IDE visual database tools to significantly accelerate development of the database components of ColdFusion applications.

Contents

Introduction to Database Tools

ColdFusion Studio's visual database tools support application development using both local and remote data sources. For ease of use, all data sources are treated as remote, whether they reside on the localhost machine or another machine. Access to data sources becomes transparent when a server is configured in Studio.

Studio's database tools support:

- Remote database development
- Data source schema and data browsing
- Visual SQL query building

Registering Data Sources

A basic set of ODBC drivers and sample data is installed with all versions of ColdFusion. Native database drivers are installed with the Enterprise version and OLE-DB drivers are installed with the Professional and Enterprise versions. To work with your data sources, you must first register them with ColdFusion Server. To do this, open the ColdFusion Administrator and select the ODBC or native drivers page under the data sources heading. From there, you simply name the data source, select the appropriate driver, and click Add.

For more information on managing data sources, see *Administering ColdFusion Server*.

To test the connection, open the Verify Data Source page, select from the drop-down list and click Verify. A help message is generated if the test connection fails.

When these steps are completed, you can move to Studio to access the data source.

Connecting to Data Sources

Studio uses the Database Connection Manager in the ColdFusion Server to connect to remote data sources.

To add an RDS server for database access:

1. Click the Remote Files tab.
2. Right-click in the file list and select Add RDS Server.
3. Complete the Remote Host fields in the Configure RDS Server dialog:
    - Description — Required. The name that appears in the server list.
    - Host Name — Required. The server domain name, such as `www.allaire.com` or an IP address.
    - User Name — Required if the server is password-protected. Open the ColdFusion Administrator Server page to edit the Studio password.

- Password — Optional. If the server is password-protected you must use a password. Open the ColdFusion Administrator Server page to edit the Studio password.
- Port — Required. The port on the server used by the Web server. Use default unless specified by server administrator.
- Use Secure Sockets Layer (SSL) — Optional. Allows exchanges between Studio and the server to be encrypted via SSL.

4. Complete the ColdFusion RDS Security fields:
    - User Name — Required if the server is password-protected. Open the ColdFusion Administrator Server page to edit or disable the Studio password.
    - Password — Optional. If the server is password-protected you must use a password. Open the ColdFusion Administrator Server page to edit or disable the Studio password.
5. Click Prompt for Password to enable these entries.
6. Click OK to complete the dialog.

Opening an ODBC Data Source

You can now open your data sources from Studio.

To open a data source:

1. Click the Database tab in the Resources pane to open the Database Servers list. The available ColdFusion servers are shown in the drop-down list.
2. Select the server whose data sources you want to access and double-click on the server name to open the data sources list.
3. Double-click on a data source name to expand the list.

At this point, you can view the database schema by opening the Tables list and expanding the individual tables. The name and definition of each column is listed. For example, City (TEXT 50) is a data field called City with the data type TEXT and a maximum field length of fifty characters. Double-click on a table name to view the records in that table.

Viewing Database Schema and Data

You can view the structure of any ODBC or native data source.

To view a data source schema:

1. Click the Database tab in the Resource pane.

2. Double-click on a data source in the server pane or right-click on a server and choose Connect.
3. Click the + sign next to an entry on the list of accessible data sources.
4. Click the + sign next to a Table folder to view the database tables.
5. Click the + sign next to a table to view the columns and column data types.

Views are only available in databases that support creating views or tables stored as queries.

To view data in a table:

1. Open the data source schema in the Database Resource tab.
2. Double-click on a table or right-click on a table and choose View Data.

You can only browse data in the data browsing window. You cannot modify data or add new records.

Tip To insert column or table names into a page, just drag and drop the table or column into the editor. This is useful when you are building database reports.

Building SQL Queries

Studio provides a powerful visual query builder to build, test, and save database queries with the Structured Query Language (SQL).

Choose one of these commands to open the Query Builder:

- Right-click on a database or on a table in the Database (DB) Resource tab and choose New Query.
- Click Tools > SQL Builder and select a database from the Select Database Query dialog.
- Open the CFQUERY tag editor and click New Query.

The Query Builder is designed to let you build different kinds of queries and then insert them into an application page. The Query Builder supports developing four types of SQL queries:

- Select
- Insert
- Update
- Delete

The Query Builder has three panes:

- Table pane — Provides a view of the tables in your query and allows you to create joins between tables.

- Properties pane — Allows you to set the properties of the query such as the columns that are being selected or the columns that are being updated.
- SQL pane — Shows you the SQL that is being built by the query builder. The SQL pane does not support reverse editing, so any changes you make in this pane will not be made to the query in the Properties pane or the Table pane.

The Query Builder toolbar gives you quick access to many of its functions.

Query Builder opens a SELECT statement by default, since this is the most common type of query. If you have not chosen a table, you will be asked to select one to use for the query.

To create an insert, update, or delete query, choose the type from the Query Builder toolbar or right-click in the Table pane and choose the Query Type.

Building a SELECT Query

SQL SELECT statements let you specify the data from which to build the recordset.

To create a SELECT statement:

1. Open the Query Builder for the data source you want to query.
2. When the Query Builder opens, select the first table you want to query.
3. Right-click in the Table pane and choose Add Table or click the Add Table button in the tool bar to add additional tables to the query.
4. Drag and drop column names between tables to create joins.
5. Right-click on a join to delete the join or change it to an inner or outer join.
6. Drag the columns you want to display onto the Column section of the Properties pane.
7. Set additional query attributes in the Properties pane.
8. Save or insert the query into your page.

To use a CFML variable in a SQL WHERE clause:

1. Click in the Criteria column in the Properties pane for the column that you want use in the WHERE clause.
2. Open the select box and choose CFVARIABLE.
3. Double-click on the variable and rename it to the variable in your application.

To save a query:

1. Click the Save Query button on the Query toolbar.
2. Enter a name for the query in the dialog.
3. Click Save.

Saved queries are stored on the ColdFusion Database Server. They can be edited and used by to everyone who has access to that server.

If you edit a SQL statement in the SQL pane and then save the query, the changes are saved. However, if you modify the Table pane or change any values in the Properties pane, a new SQL statement is generated, overwriting any edits you made in the SQL pane.

Inserting Queries into a Page

You have four options for inserting a query directly into a CFML page from the Query Builder:

- If you open the Query Builder from the CFQUERY tag editor, it will prompt you to insert the query when you close the editor.
- If you open the Query Builder from a CFML page, the it will prompt you to insert the query when you close it.
- Click the Copy SQL to Clipboard button on the Query toolbar. Close the Query Builder and paste the SQL into your page.
- Click the Copy CFQUERY button on the Query toolbar. Close the Query Builder and paste the CFQUERY tag into your page.

To insert a saved query:

Method 1

1. Open the data source schema in the Database Resource tab.
2. Open the Queries folder in the data source you want to use.
3. Drag and drop the query onto the page.

Method 2

1. Click the CFQUERY button on the CFML Basic tag toolbar or select CFQUERY from the Tag Chooser (CTRL+E).
2. Enter a name for the query.
3. Click the Insert Query button.
4. Choose a server from the select box.
5. Open the Queries folder in the data source you want to use.
6. Select a query and click Insert.
7. Click Apply in the CFQUERY tag editor.

Running and Editing Queries

You can do a lot of work with SQL statements after building them.

Running Queries

You can test your SQL code in Query Builder before inserting it a page.

To run a query in the Query Builder:

1. Click the Run Query button.
2. You are prompted to enter values for any CFML variables in the query.

Warning When you click the Run Query button, the SQL statement is actually processed. If you run an INSERT, UPDATE, or DELETE statement, the changes you coded in the SQL statement are made in the data source.

Editing Queries

If you save a query, you can edit it later. Pages that contain the edited query are not automatically updated, so you will need to re-insert the query for the changes to take effect.

To edit a query:

1. Open the query folder for the data source you want to use.
2. Double-click on the query you want to edit.
3. Make changes in the Table pane and the Properties pane.
4. Save the new query.

Note The interactions among the Query Builder panes differ somewhat. If you write a query in the SQL pane and save it, the code is not reflected in the Properties or Table panes. If you take an action, such as adding a table, in the Properties or Table panes, the SQL pane is refreshed with the SQL generated from these panes.

CHAPTER 11

Inserting, Updating, and Deleting Data

This chapter describes how to insert, update, and delete data in a database with ColdFusion.

Contents

Inserting Data

Inserting data into a database is usually done with two application pages:

- An insert form
- An insert action page

You can create an insert form with CFFORM tags or with standard HTML form tags. When the form is submitted, form variables are passed to a ColdFusion page that performs an insert operation (and whatever else is called for) on the specified data source. The insert page can contain either a CFINSERT tag or CFQUERY tag with a SQL insert statement. The insert page should also contain a message for the end user.

Creating an HTML Insert Form

When creating a form in HTML, recall that you must specify the ACTION and METHOD attributes. Generally, the METHOD attribute is "POST" and the ACTION attribute specifies the name of the ColdFusion page (.CFM) file you want to execute.

Setting a form's ACTION attribute

The ACTION attribute in an HTML form tells the browser what to do when the user clicks the Submit button. In the following example, the `insdata.cfm` page is executed when the user submits the form data for processing.

```
<FORM ACTION="insdata.cfm" METHOD="Post">
```

Implementing data entry fields

Creating data entry fields for an HTML form is very simple: you need only create the HTML form fields for each database field into which you want to insert data. The names of your form fields must be identical to the names of your database fields.

For example, if you have a table called Employees with three fields called FirstName, LastName, and Phone, your form fields might look like this:

```
First Name: <INPUT TYPE="text" NAME="FirstName">
Last Name: <INPUT TYPE="text" NAME="LastName">
Phone: <INPUT TYPE="text" NAME="Phone">
```

You can use the full range of HTML input controls, including list boxes, radio buttons, checkboxes, and multi-line text boxes in your forms. When ColdFusion reads the contents of the form submittal, it uses the NAME attribute to map HTML form fields to the corresponding database fields and inserts the data entered by the user into the appropriate database fields.

Hidden form fields

Hidden fields are a special type of form input field. When you define a hidden input field, the field remains a part of the HTML form but is not displayed to the user. When a user submits the form, the VALUE of the hidden field (which is typically specified when preparing the form) is sent to ColdFusion along with user-entered fields that are not hidden.

For example, if you want a form submitted by a user to always include the site from which it was submitted, you might have a hidden input field such as:

```
<INPUT TYPE="Hidden" NAME="SiteName" VALUE="CompanyName">
```

In the above case, every time a user completes the HTML form, a variable with the name "SiteName" and the value "CompanyName" is passed as a part of the form submittal.

Example: HTML insert form

The following example demonstrates a simple HTML form:

```
<FORM ACTION="insdata.cfm" METHOD="Post">
    <!-- Data entry fields -->
<PRE>
    First Name: <INPUT TYPE="text" NAME="FirstName">
    Last Name: <INPUT TYPE="text" NAME="LastName">
    Phone: <INPUT TYPE="text" NAME="Phone">
    <INPUT TYPE="Submit" VALUE="Enter Information">
</PRE>
</FORM>
```

The form has three inputs: FirstName, LastName, and Phone. The user can enter data in these text areas and click the Submit button. When the Submit button is clicked, the form action is carried out, and all inputs (including hidden inputs) are made available to the next page.

Suppose the user enters his first name as "William," his last name as "Gibson," and his phone as "(212)323-9734." When he clicks the Submit button, the form variables shown below are sent to the page:

```
FirstName=William
LastName=Gibson
Phone=(212)323-9734
```

This page might display these variables, insert them into the database, or perhaps do both depending on your application.

Creating an Insert Page with CFINSERT

The CFINSERT tag is the easiest way to handle simple inserts from either a CFFORM or an HTML form.

In most cases, the optional attributes are not needed. The TABLEOWNER and TABLEQUALIFIER attributes are rarely necessary but are provided for compatibility with ODBC drivers that require you to specify a table owner and/or table qualifier. Neither of these fields needs to be specified for the Microsoft ODBC Desktop Drivers bundled with ColdFusion.

ODBC drivers that require table owners and/or qualifiers to be specified include all SQL Server and Oracle drivers, as well as all Intersolv Q&E drivers.

CFINSERT datasource

The ODBC data source is named "Employees DB" and the table you want to insert data into is named "Employees." Given this information, the CFINSERT tag would be included in your page as follows:

```
<CFINSERT DATASOURCE="Employee DB" TABLENAME="Employees">
```

Example: HTML form page

The following example illustrates an HTML form with the CFINSERT tag.

```
<!-- HTML form to input data -->
<HTML>
<HEAD>
    <TITLE>Input Form</TITLE>
</HEAD>
<BODY>

<FORM ACTION="EmployeeInsert.cfm"
    METHOD="Post">
<PRE>
First Name: <INPUT TYPE="text"
    NAME="FirstName">
Last Name: <INPUT TYPE="text"
    NAME="LastName">
Phone: <INPUT TYPE="text"
    NAME="Phone">
<INPUT TYPE="Submit"
    VALUE="Insert Information">
</PRE>
</FORM>

</BODY>
</HTML>
```

Example: CFINSERT action page

```
<!--- Inserts the data from the the HTML Form --->
<CFINSERT DATASOURCE="Employee DB"
    TABLENAME="Employees">

<HTML>
```

```
<HEAD>
    <TITLE>Input Form</TITLE>
</HEAD>
<BODY>

<CENTER><H2>Thank You!</H2></CENTER>
<HR>
<P>Thank you for entering your data into
our database - please visit our site often!</P>
<HR>

</BODY>
</HTML>
```

Creating an Insert Page with CFQUERY

For more complex inserts from a form submittal you can use a SQL INSERT statement in a CFQUERY tag instead of a CFINSERT tag. The SQL INSERT statement is more flexible because you can insert information selectively or use functions within the statement.

Basic SQL syntax

The syntax for a basic SQL insert statement is:

```
INSERT INTO tablename (columnnames)
VALUES (values)
```

The VALUES keyword specifies the values for the columns in the new row. You have to type the values you want to add in the same order as the columns in the *columnnames* section of the statement.

Example: CFQUERY insert

To insert the form data from the example above with a CFQUERY use this syntax:

```
<CFQUERY NAME="AddEmployee"
    DATASOURCE="Employee DB">
    INSERT INTO Employees (FirstName, LastName, Phone)
    VALUES ('#Form.FirstName#', '#Form.LastName#',
    '#Form.Phone#')
</CFQUERY>
```

Updating Data

Updating data in a database is usually done with two pages:

- An update form
- An update page

The update form is created with CFFORM tags or HTML form tags. The update form calls an update page which can contain either a CFUPDATE tag or a CFQUERY tag with a SQL UPDATE statement. The update page should also contain a message for the end user that reports on the update completion.

Creating an Update Form

An update form is similar to an insert form with two key differences. An update form contains a reference to the primary key of the record that is being updated. A primary key is a field or combination of fields in a database table that uniquely identifies each record in the table. For example, in a table of Employee names and addresses, only the Employee_ID would be unique to each record. Because the purpose of an update form is to update existing data, the contents of an update form are usually populated out of a database.

Dynamically populating an update form

To populate the fields of an update form, you must first select the record out of the database with a CFQUERY. Then put the form in a CFOUTPUT statement to reference the fields.

Designating the primary key

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.

Example: Primary key value

```
<!--- Query to select record --->
<CFQUERY NAME="EmployeeRecord"
    DATASOURCE="Employee DB">
    SELECT *
        FROM Employees
        WHERE Employee_ID = #URL.EmployeeID#
</CFQUERY>

<HTML>
<HEAD>
    <TITLE>Input Form</TITLE>
</HEAD>
<BODY>

<CFOUTPUT QUERY="EmployeeRecord">

<!--- Input form --->
<FORM ACTION="EmployeeUpdate.cfm" METHOD="Post">

<!--- Primary Key value indicating record to update --->
```

```
<INPUT TYPE="Hidden" NAME="Employee_ID"
    VALUE="#Employee_ID#">
<PRE>
    FirstName: <INPUT TYPE="Text" NAME="FirstName"
        VALUE="#FirstName#">
    LastName: <INPUT TYPE="Text" NAME="LastName"
        VALUE="#LastName#">
    Phone: <INPUT TYPE="Text" NAME="Phone"
        VALUE="#Phone#"><BR>
    <INPUT TYPE="Submit" VALUE="Update Information">
</PRE>
</FORM>
</CFOUTPUT>

</BODY>
</HTML>
```

In this example, Employee_ID is the primary key of the Employees table, so a hidden field named Employee_ID is included in the HTML form. The hidden field indicates to ColdFusion which record to update. In this case, the record ID was passed as the URL parameter EmployeeID:

`http://web_root/updateform.cfm?employeeid=2`

Creating an Update 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 CFUPDATE, you must include all of the fields that comprise the primary key in your form submittal. The CFUPDATE tag automatically detects the primary key fields in the table you are updating and seeks them out in the submitted form fields. ColdFusion uses the primary key field(s) to select the record to update. It then updates the appropriate fields in the record using the remaining form fields submitted.

Example: CFUPDATE page

The ODBC data source is named "Employee DB," and the table you want to update is named "Employees." With the form example just described as the front end, the CFUPDATE tag would be included in your page as follows:

```
<CFINSERT DATASOURCE="Employee DB"
    TABLENAME="Employees">

<!--- This is the page EmployeeUpdate.cfm --->
<CFUPDATE DATASOURCE="Employee DB"
    TABLENAME="Employees">

<HTML>
<HEAD>
    <TITLE>Reply</TITLE>
</HEAD>
```

```
<BODY>

<H2>Thank You!</H2>
<HR>
<P>Thank you for updating your data in
our database - please visit our site often!</P>
<HR>

</BODY>
</HTML>
```

Creating an Update 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.

Syntax

The syntax for a SQL update statement is:

```
UPDATE tablename
    SET columnname = value
    WHERE condition
```

After the SET clause, a table column must be named. Then, you indicate a constant or expression as the value for the column.

Example: CFQUERY update page

To update the record with the front end form from the example above using a CFQUERY use this syntax:

```
<CFQUERY NAME="UpdateEmployee"
    DATASOURCE="Employee DB">
    UPDATE Employees
        SET Firstname='#Form.Firstname#',
        LastName='#Form.LastName#',
        Phone='#Form.Phone#'
    WHERE Employee_ID=#Employee_ID#
</CFQUERY>
```

Note The WHERE statement is optional, but if you do not use it in the SQL UPDATE statement or the UPDATE command, then every row in the database will be updated.

Deleting Data

Deleting data in a database can be done with a single delete page. The delete page contains a CFQUERY tag with a SQL delete statement.

Syntax

The syntax for a SQL delete statement is:

```
DELETE FROM tablename
WHERE condition
```

The *condition* controls whether or not the delete statement deletes a single record, several records, or all records.

Example: Deleting a single record

The following example demonstrates deleting a single employee from the Employees table.

```
DELETE FROM Employees
WHERE Employee_ID = 1
```

Example: Deleting several records

The following example demonstrates deleting several records from the Employee table. The example assumes that there are several Employees in the sales department.

```
DELETE FROM Employees
WHERE Department = 'Sales'
```

Example: Deleting all records

The following example demonstrates deleting all the records from the Employees table.

```
DELETE FROM Employees
```

Note Deleting records from a database is *not* reversible. Use delete statements carefully.

Example: Complete delete page

```
<!--- Page to delete single employee record --->
<CFQUERY NAME="DeleteEmployee"
    DATASOURCE="Employee DB">
    DELETE FROM Employees
    WHERE Employee_ID = #URL.EmployeeID#
</CFQUERY>

<HTML>
<HEAD>
    <TITLE>Delete Employee Record</TITLE>
```

```
</HEAD>
<BODY>
<H3>The employee record has been deleted.</H3>

</BODY>
</HTML>
```

Data Input Validation

When you use forms to capture input data from users for a database query, you often want to validate the user's input before sending the query to the database. This is especially true when you create front ends for SQL statements that require a specific data type, for example SQL statements containing date or numeric comparisons. Validation ensures correct processing by the data source.

You can apply validation rules to any form submittal sent to the ColdFusion application server. ColdFusion offers several different types of data input validation.

Data Input Validation Types	
Validation Type	**Description**
Client-side	In a CFFORM, you can specify a JavaScript program in the ONVALIDATE attribute of tags like CFINPUT, CFGRID, CFSLIDER, CFTEXTINPUT, and CFTREE to perform input validation.
Server-side	In a CFFORM, you can enable validation in tags that support input validation (like CFINPUT and CFTEXTINPUT) using the VALIDATE attribute. You can also use hidden fields in HTML forms to require user entries and to validate several common data types.

Required form fields

One of the weaknesses of HTML forms is the inability to define input fields as required. Because this is a particularly important requirement for database applications, ColdFusion provides a server-side mechanism for requiring users to enter data in fields.

To define an input field as required, use a hidden field that has a NAME attribute composed of the field name and the suffix "_required." For example, to require that the user enter a value in the FirstName field, use the syntax:

```
<INPUT TYPE="hidden" NAME="FirstName_required">
```

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 using the VALUE attribute of the hidden field. For

example, if you want the error message to read "You must enter your first name," use the syntax:

```
<INPUT TYPE="hidden"
    NAME="FirstName_required"
    VALUE="You must enter your first name.">
```

Hidden form fields

Another weakness of HTML forms is that you cannot validate data input by users. ColdFusion enables you to do several types of data validation by adding hidden fields to forms. The hidden field suffixes you can use to do validation are as follows:

Form Field Validation Using Hidden Fields

Field Suffix	Value Attribute	Description
`_integer`	Custom error message	Verifies that the user enters a number. If the user enters a floating point value, it is rounded to an integer.
`_float`	Custom error message	Verifies that the user enters a number. Does not do any rounding of floating point values.
`_range`	MIN=MinValue MAX=MaxValue	Verifies that the numeric value entered is within the specified boundaries. You can specify one or both of the boundaries separated by a space.
`_date`	Custom error message	Verifies that a date has been entered and converts the date into the proper ODBC date format. Will accept most common date forms, for example, 9/1/98; Sept. 9, 1998).
`_time`	Custom error message	Verifies that a time has been correctly entered and converts the time to the proper ODBC time format.
`_eurodate`	Custom error message	Verifies that a date has been entered in a standard European date format and converts into the proper ODBC date format.

Examples: Hidden fields

The following examples illustrate the use of hidden fields to validate data. In this example (a hotel reservation form), the FORM being validated contains the fields "Rooms," "Guests," and "ArrivalDate." To ensure that the Rooms field contains an integer, that the Guests field is from 1 to 12, and that the ArrivalDate is a valid date, add the following hidden fields to the form:

```
<INPUT TYPE="hidden"
    NAME="Rooms_integer"
    VALUE="You must enter a number for the Rooms field.">

<INPUT TYPE="hidden"
    NAME="Guests_range"
    VALUE="MIN=1 MAX=12">
<INPUT TYPE="hidden"
    NAME="ArrivalDate_date"
    VALUE="This is not a valid arrival date.">
```

The VALUE attribute is optional. A default message displays if no value is supplied.

When the form is submitted, ColdFusion scans the form fields to find any validation rules you specified. The rules are then used to analyze the user's input. If any of the input rules are violated, ColdFusion sends an error message to the user that explains the problem. The user then must go back to the previous screen, correct the problem and resubmit the form. ColdFusion will not accept the submittal until the entire form is entered correctly.

Automatic validation of numeric and date fields

If you use CFINSERT or CFUPDATE and you specified columns in your database that are numeric, date, or time, then form fields inserting data into these fields are automatically validated. You can use the hidden field validation functions for these fields to display a custom error message.

Additional notes on validation

- Adding a validation rule to a field does not make it a required field. You need to add a separate `_required` hidden field if you want to ensure user entry.
- Because numeric values often contain commas and dollar signs, these characters are automatically stripped out of fields with `_integer`, `_float`, or `_range` rules before they are validated and saved to the database.

Dynamic HTML Forms

One of the most important tools for working with relational databases in a Web application is dynamic forms. A dynamic form is an HTML form that uses elements created with database query results and CFML. Most often these elements are either radio buttons, check boxes, select lists, or multiple select lists.

You can use dynamic forms to help ensure data integrity, to speed coding, and to create relationships between tables in your database.

Example: Query form

This example shows a select list of park names that is created with a query against the parks table.

```
<CFQUERY NAME="ParkNames"
    DATASOURCE="ParkDB">
    SELECT ParkName_ID, ParkName
        FROM Parks
</CFQUERY>

<FORM ACTION="example.cfm" METHOD="Post">
    <SELECT NAME="ParkName_ID">
    <CFOUTPUT QUERY="ParkNames">
        <OPTION VALUE="#ParkName_ID#">#ParkName#
    </CFOUTPUT>
    </SELECT>
    <INPUT TYPE="submit" VALUE="Submit">
</FORM>
```

When this form is submitted, it passes the ParkName_ID, which is the primary key for the chosen park. This value can be then used by the `example.cfm` page. Other elements of a form can be dynamically created in the same way.

The following section explains how to work with checkboxes and multiple select lists that can be created in dynamic forms.

Using checkboxes and multiple select lists in HTML forms

When an HTML form contains either a list of checkboxes with the same name or a multiple select box, the user's entries are made available as a comma-delimited list with the selected values. These lists can be very useful for a wide range of inputs.

Note If no value is entered for a checkbox or multiple select lists then no variable is created.The SQL INSERT statement will not work correctly if there are no values. To correct this problem, make the form fields required or use Dynamic SQL.

Checkboxes

When you put a series of checkboxes with the same name in an HTML form the variable that is created contains a comma-delimited list of values. This can be done either with numeric values or alphanumeric strings. These two types of values are treated slightly differently.

Searching numeric values

Suppose you want to present a user with a list of organizations. A user is prompted to select one or more organizations using checkboxes. The query retrieves detailed information on the selected organization(s).

```
Select one or more companies to get information on:

<INPUT TYPE="checkbox"
    NAME="SelectedOrgs"
    VALUE="5">
    Mobil Corporation<BR>

<INPUT TYPE="checkbox"
    NAME="SelectedOrgs"
    VALUE="19">
    Shapeware, Inc.<BR>

<INPUT TYPE="checkbox"
    NAME="SelectedOrgs"
    VALUE="13">
    BankBoston<BR>

<INPUT TYPE="hidden"
    NAME="SelectedOrgs_required"
    VALUE="You must select at least one organization.">
```

Note that while the text displayed to the user is the name of the organization, the VALUE attribute of each checkbox corresponds to the underlying database primary key for the organization's record.

If the user checked the Shapeware and BankBoston items, the value of the SelectedOrgs form field would be "19,13." If this parameter were used in the following SQL statement:

```
SELECT *
    FROM Organizations
    WHERE Organization_ID IN ( #SelectedOrgs# )
```

the statement sent to the database would be:

```
SELECT *
    FROM Organizations
    WHERE Organization_ID IN ( 19,13 )
```

This statement retrieves detailed information on Shapeware and Bank of Boston that you can then display to the user with a CFOUTPUT section.

Searching string values

To search for a database field containing string values (instead of numeric), you must modify both the checkbox and CFQUERY syntax.

In the first example, we searched for company information based on a numeric primary key field called "Organization_ID." If instead the primary key was a database field called "OrganizationName" that contained string values, we must make the following two modifications:

Single quotes must be added to the value attributes of the checkboxes.

```
<INPUT TYPE="checkbox"
    NAME="SelectedOrgs"
    VALUE="'Mobil Corporation'">
    Mobil Corporation<BR>

<INPUT TYPE="checkbox"
    NAME="SelectedOrgs"
    VALUE="'ShapeWare, Inc.'">
    Shapeware, Inc.<BR>

<INPUT TYPE="checkbox"
    NAME="SelectedOrgs"
    VALUE="'Bank of Boston'">
    BankBoston<BR>

<INPUT TYPE="checkbox"
    NAME="SelectedOrgs_required"
    VALUE="You must select at least one organization.">
```

If the user checked the Shapeware and BankBoston items, the value of the SelectedOrgs form field would be 'ShapeWare, Inc.','BankBoston'.

You must use the ColdFusion PreserveSingleQuotes function in the SQL statement to prevent ColdFusion from escaping the single quotes in the form field value:

```
SELECT *
    FROM Organizations
    WHERE OrganizationName IN
    (#PreserveSingleQuotes(SelectedOrgs)#)
```

The statement sent to the database would be:

```
SELECT *
    FROM Organizations
    WHERE OrganizationName IN ('ShapeWare, Inc.','BankBoston')
```

Multiple select lists

ColdFusion treats multiple select lists (HTML input type SELECT with attribute MULTIPLE) just like checkboxes. The data made available to your page from any multiple select list is a comma-delimited list of the entries selected by the user. For example, a multiple select list contains three entries: red, green, and blue. The user selects red and green. The value of the form field variable is then 'red', 'green'.

As with checkboxes, two cases exist when querying for information with multiple select lists: searching a database field containing numeric values, and searching a database field containing string values.

Searching numeric values

For example, suppose you want the user to select organizations from a multiple select box. The query retrieves detailed information on the selected organization(s):

```
Select one or more companies to get more information on:
<SELECT Name="SelectOrgs" MULTIPLE>
    <OPTION VALUE="5">Mobil Corporation
    <OPTION VALUE="19">ShapeWare, Inc.
    <OPTION VALUE="13">BankBoston
</SELECT>

<INPUT TYPE="hidden"
    NAME="SelectedOrgs_required"
    VALUE="You must select at least one organization.">
```

If the user selected the Shapeware and BankBoston items, the value of the SelectedOrgs form field would be 19,13.

If this parameter were used in the following SQL statement:

```
SELECT *
    FROM Organizations
    WHERE Organization_ID IN (#SelectedOrgs#)
```

the statement sent to the database would be:

```
SELECT *
    FROM Organizations
    WHERE Organization_ID IN (19,13)
```

Searching string values

Suppose you want the user to select organizations from a multiple select list. The database field to be searched is a string field. The query retrieves detailed information on the selected organization(s):

```
Select one or more companies to get
more information on:
<SELECT Name="SelectOrgs" MULTIPLE>
    <OPTION VALUE="'Mobil Corporation'">Mobil Corporation
    <OPTION VALUE="'ShapeWare, Inc.'">ShapeWare, Inc.
    <OPTION VALUE="'BankBoston'">BankBoston
</SELECT>

<INPUT TYPE="hidden"
    NAME="SelectedOrgs_required"
    VALUE="You must select at least one organization.">
```

If the user selected the Shapeware and BankBoston items, the value of the SelectedOrgs form field would be 'ShapeWare, Inc.','BankBoston'.

As when using checkboxes to search database fields containing string values, the ColdFusion PreserveSingleQuotes function must be used with multiple select boxes:

```
SELECT *
    FROM Organizations
    WHERE OrganizationName IN (#PreserveSingleQuotes(SelectedOrgs)#)
```

The statement sent to the database would be:

```
SELECT *
    FROM Organizations
    WHERE OrganizationName IN ('ShapeWare, Inc.', 'BankBoston')
```

Dynamic SQL

Embedding SQL queries that use dynamic parameters is a powerful mechanism for linking variable inputs to database queries. However, in more sophisticated applications, you will often want user inputs to determine not only the content of queries but also the structure of queries.

Dynamic SQL allows you to dynamically determine (based on runtime parameters) which parts of a SQL statement are sent to the database. So if a user leaves a search field empty, for example, you could simply omit the part of the WHERE clause that refers to that field. Or, if a user does not specify a sort order, the entire ORDER BY clause could be omitted.

Dynamic SQL is implemented in ColdFusion by using CFIF/CFELSEIF/CFELSE tags to control how the SQL statement is constructed.

Syntax

A dynamic SQL statement using the CFQUERY tag takes the following form:

```
<CFQUERY NAME="queryname"
    DATASOURCE="datasourcename">
...Base SQL statement

<CFIF value operator value >
...additional SQL
</CFIF>

</CFQUERY>
```

Example: Using CFIF

For example, in the following code, a series of CFIF tags determine which SQL statements to append to the base SQL SELECT statement:

```
<CFQUERY NAME="GetParkList"
    DATASOURCE="CF 4.0 Examples">
    SELECT *
        FROM Parks
        WHERE 0=0

<CFIF #ParkName# is not "">
    AND ParkName LIKE '%#ParkName#%'
</CFIF>

<CFIF #ParkType# is not "AllTypes">
    AND ParkType = '#ParkType#'
</CFIF>
```

```
<CFIF #Region# is not "AllRegions">
    AND Region = '#Region#'
</CFIF>

<CFIF #State# is not "">
    AND State = '#State#'
</CFIF>

</CFQUERY>
```

Tip The WHERE 0=0 clause has no impact on the query submitted to the database. But if none of the conditions is true, it ensures that the WHERE clause does not result in a SQL syntax error.

Example: Creating a select list

This example shows how to use dynamic SQL to ensure that a multiple select list is created.

```
<CFQUERY NAME="GetParkList"
DATASOURCE="CF 4.0 Examples">
    SELECT *
        FROM Parks
        WHERE 0=0
<CFIF IsDefined("ParkName_ID")>
    AND ParkName_ID IN (#Form.ParkName_ID#)
</CFIF>
</CFQUERY>
```

Transaction Processing (CFTRANSACTION)

You can use the CFTRANSACTION tag to maintain consistency across queries. All queries contained within a CFTRANSACTION tag are treated as a transactional unit. This means that changes made to the database are not permanently committed until all queries in the transaction block execute successfully. If an error occurs in one of the queries, all changes made by previous queries within the transaction block are rolled back.

Syntax and example

The following example illustrates the use of CFTRANSACTION. If an error occurs in the second query, CFTRANSACTION guarantees that a transfer of account funds does not leave the database in an inconsistent state:

```
<CFTRANSACTION>

<CFQUERY NAME="WithdrawCash"
    DATASOURCE="BankDB">
    UPDATE Accounts
```

```
    SET Balance=Balance - #Amount#
        WHERE Account_ID=#AccountFrom#
</CFQUERY>

<CFQUERY NAME="DepositCash"
    DATASOURCE="BankDB">

    UPDATE Accounts
    SET Balance=Balance + #Amount#
        WHERE Account_ID=#AccountTo#

</CFQUERY>

</CFTRANSACTION>
```

If an error occurs during the execution of the "DepositCash" query, the update made in the "WithdrawCash" query is automatically rolled back. Transactions are only supported for the same datasource.

Setting transaction isolation

You can specify an optional ISOLATION attribute with the CFTRANSACTION tag. The ISOLATION attribute provides fine-grained control over how the database engine performs locking during the transaction.

Valid values for the ISOLATION attribute are:

- READ_UNCOMMITTED
- READ_COMMITTED
- REPEATABLE_READ
- SERIALIZABLE

Note VERSIONING is no longer a valid value for this attribute and is no longer supported in ColdFusion.

ODBC driver support for transactions

Not every driver supports transactions and not every driver that supports transactions supports all ISOLATION levels. When you attempt to use a transaction/isolation-level combination for a specific driver, ColdFusion queries the driver for its transaction capabilities. ColdFusion returns an error if the driver indicates it is not capable of implementing the request. Consult your driver's documentation for more information on the ISOLATION levels it supports and on the behavior of the driver for each level.

CHAPTER 12

Building Dynamic Java Forms

This chapter shows you how to use the CFFORM tag to enrich your forms with sophisticated graphical controls, including several Java applet-based controls. These controls can be enabled without the need to code Java directly.

Contents

Creating Forms with the CFFORM Tag

The CFFORM tag allows you to create dynamic forms in CFML and gives you access to a wide range of form controls, such as Java applet-based tree, slider, and grid controls, as well as the standard HTML control types like check boxes, radio buttons, text input boxes, and edit boxes. With CFFORM, you gain the advantage of access to these Java applet-based controls without having to know the Java language, and, you don't have to juggle CFOUTPUT tags and HTML FORM tags to reference ColdFusion variables in your forms.

Forms created using the CFFORM tags are structured just as HTML forms. Within the <CFFORM> and </CFFORM> tags, you place entries for form controls, such as check boxes and radio buttons (using CFINPUT), data grids (using CFGRID), tree controls (CFTREE), or drop-down lists and select boxes (CFSELECT).

The *HTML Reference* contains complete online information on the HTML FORM tag. You can open it from the Window Start menu by clicking Welcome to ColdFusion and then selecting it from the Documentation list. You can also open it from the ColdFusion Studio Help References.

Using HTML in a CFFORM

You can use the HTML FORM tag in combination with the CFFORM tag. ColdFusion generates HTML forms dynamically from CFFORM tags and passes through to the browser any HTML code it finds in the form. You can also replace your existing HTML FORM tags with ColdFusion CFFORM and your forms will work fine.

Advantages of Using Dynamic Forms

Building a form with CFFORM provides the following advantages over building a form using just HTML:

- You no longer have to juggle CFOUTPUT sections and HTML FORM tags in order to use ColdFusion expressions. Expressions can be used directly in tags inside a CFFORM.
- CFFORM elements include a number of Java applet-based controls you can use immediately without having to learn about Java. These dynamic controls add tremendous power to forms.
- Most CFFORM controls offer input validation attributes you can use to validate a user's entry, selection, or interaction. Input validation is often done with JavaScript. You can reference JavaScript programs in your CFFORM.
- Using CFAPPLET, you can simplify the job of embedding custom Java applets of your own in your CFFORM.

For nearly all CFFORM controls, you can specify font characteristics, alignment, size, scrolling properties, as well as a number of other options.

CFFORM controls

When you build a form using CFFORM, you typically use one or more of the following controls:

- CFGRID — A Java applet-based control used to create a data grid you can populate from a query or by defining the contents of individual cells. Grids can also be used to insert, update, and delete records from a data source.
- CFSLIDER — A Java applet-based control used to define a slider.
- CFINPUT — Used to place radio buttons, check boxes, or text input boxes.
- CFTREE and CFTREEITEM — More Java applet-based controls used to define a tree control and individual tree control items.
- CFTEXTINPUT — A Java applet-based control used to define a text input box.
- CFSELECT — Used to define a drop-down list box.
- CFAPPLET — Allows you to embed your own Java applets.

Improving performance with ENABLECAB

The CFFORM ENABLECAB attribute allows you to improve the performance of Java-applet based CFFORM controls. When you use ENABLECAB, ColdFusion prompts the end user to accept a download of the Java classes needed for the CFFORM controls that use them. CAB files are digitally signed using VeriSign digital IDs to ensure file security.

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

Browsers that disable Java

Since each of the Java applet-based controls, CFGRID, CFSLIDER, CFTEXTINPUT, and CFTREE require a Java applet to run, browsers that do not support Java or that have disabled Java execution will not execute the forms that contain these controls. Using the NOTSUPPORTED attribute, ColdFusion allows you to present an error message rather than the blank applet space that appears in the browser. This attribute is available in each of the Java applet-based controls as well as the CFAPPLET tag. You use NOTSUPPORTED to specify the message you want to appear, formatted as HTML, when an application page is loaded by a browser that does not support Java.

Input Validation with CFFORM Controls

The CFINPUT and CFTEXTINPUT tags include the VALIDATE attribute, which allows you to specify a valid data type entry for the control. You can validate user entries on the following data types.

Input Validation Controls	
VALIDATE Entry	**Description**
Date	Verifies US date entry in the form mm/dd/yyy.
Eurodate	Verifies valid European date entry in the form dd/mm/yyyy.
Time	Verifies a time entry in the form hh:mm:ss.
Float	Verifies a floating point entry.
Integer	Verifies an integer entry.
Telephone	Verifies a telephone entry. Telephone data must be entered as ###-###-####. The hyphen separator (-) can be replaced with a blank. The area code and exchange must begin with a digit between 1 and 9.
Zipcode	(U.S. formats only) Number can be a 5-digit or 9-digit zip in the form #####-####. The hyphen separator (-) can be replaced with a blank.
Creditcard	Blanks and dashes are stripped and the number is verified using the mod10 algorithm.
Social_security_number	Number must be entered as ###-##-####. The hyphen separator (-) can be replaced with a blank.

When you specify an input type in the VALIDATE attribute, ColdFusion tests for the specified input type and submits form data only on a successful match. A true value is returned on successful form submission, false if validation fails.

Input Validation with JavaScript

In addition to native ColdFusion input validation using the VALIDATE attribute of the CFINPUT and CFTEXTINPUT tags, the following tags support the ONVALIDATE attribute, which allows you to specify a JavaScript function to handle your CFFORM input validation:

- CFINPUT

- CFGRID
- CFSLIDER
- CFTEXTINPUT
- CFTREE

JavaScript objects passed to the validation routine

The following JavaScript objects are passed by ColdFusion to the JavaScript function you specify in the ONVALIDATE attribute:

- form_object
- input_object
- object_value

Handling failed validation

The ONERROR attribute allows you to specify a JavaScript function you want to execute in the event of a failed validation. For example, if you specify a JavaScript function to handle input validation in the ONVALIDATE attribute you can also specify a JavaScript function in the ONERROR attribute to handle a failed validation, which returns a false value. ONERROR is available in the following CFFORM tags:

- CFGRID
- CFINPUT
- CFSELECT
- CFSLIDER
- CFTEXTINPUT
- CFTREE

When you specify a JavaScript routine in the ONERROR attribute, ColdFusion passes the following JavaScript objects to the specified routine:

- form_object
- input_object
- object_value
- error message text

Example: Form validation

The following sample ColdFusion page includes JavaScript to validate an entry for an email address:

```
<HTML>
<HEAD>
    <TITLE>JavaScript Validation</TITLE>

<SCRIPT>
<!--

function testbox(form) {
    Ctrl = form.inputbox1;
    if (Ctrl.value == "" || Ctrl.value.indexOf ('@', 0) == -1) {
    return (false);
    } else
        return (true);
}

//-->
</SCRIPT>

</HEAD>

<BODY>
<H2>JavaScript validation test</H2>

<P>Please enter your email address:</P>
<CFFORM NAME="UpdateForm"
    ACTION="update.cfm" >
    <CFINPUT TYPE="text"
    NAME="inputbox1"
    REQUIRED="YES"
    ONVALIDATE="testbox"
    MESSAGE="Sorry, invalid entry."
    SIZE="10"
    MAXLENGTH="10">

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

</BODY>
</HTML>
```

See the following Web sites for information on JavaScript validation scripts:

- http://javascript.developer.com
- http://www.dannyg.com/javascript
- http://www.hotwired.com/webmonkey/javascript/

Building Tree Controls with CFTREE

The CFTREE form control is one of the most useful of the Java applet-based tags in ColdFusion. With it, you can create collapsible tree controls populated from data source queries. To build a tree control with CFTREE, you use individual CFTREEITEM

tags to populate the control. Tree controls are very useful for displaying hierarchical information in a space-saving control. You can create shallow or deep tree structures, and you can specify one of six built-in icons to represent individual items in the tree control.

Populating a tree with query data

The following very simple CFTREE example is populated with data from a CFQUERY. It uses a minimum of CFTREE and CFTREEITEM attributes to show how you can create and populate a tree control with just a handful of CFML.

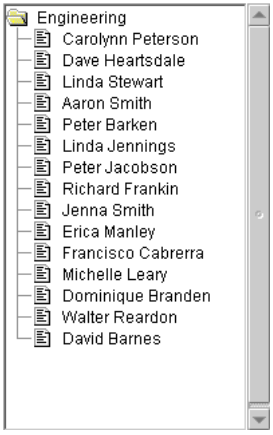
First, the query selects data from the data source:

```
<CFQUERY NAME="Engineering" DATASOURCE="cfsnippets">
    SELECT FirstName + ' ' + LastName AS FullName
    FROM EMPLOYEES
</CFQUERY>
```

Next, the CFTREE is built using data from the query:

```
<CFFORM NAME="form1" ACTION="submit.cfm"
    METHOD="Post">
<CFTREE NAME="tree1" REQUIRED="yes"
    HSCROLL="no" vSCROLL="yes">
    <CFTREEITEM VALUE=FullName
        QUERY="Engineering"
        QUERYASROOT="yes"
        IMG="folder,document">
</CFTREE>
</CFFORM>
```

The resulting tree control looks like this:



This example uses the QUERYASROOT attribute to specify the query name as the root level of the tree control. The QUERYASROOT attribute takes either a yes/no argument, or a name you want to appear as the root level for data returned by a CFQUERY. You

could, for example, populate a CFTREE with data from several different, identical queries or from several different data sources. Using QUERYASROOT allows you to specify each individual query as the source of data for a particular section of the CFTREE.

Grouping output from a query

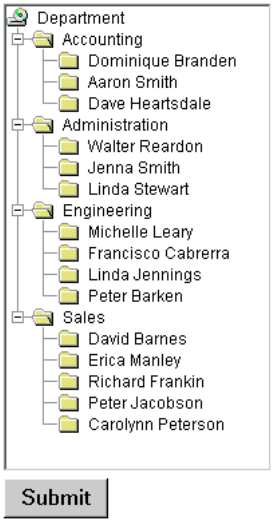
In a similar query, you may want to organize your employees by the department they work in and then display a complete list that reflects organization by department. In this case, ColdFusion provides a very simple means for displaying output from an ordered CFQUERY in a CFTREE. You separate column names with commas in the CFTREEITEM VALUE attribute, and ColdFusion understands you want the tree control to reflect the ordering of the SQL statement:

Example: Grouping query output

```
<!--- CFQUERY with an ORDER BY clause --->
<CFQUERY NAME="myquery" DATASOURCE="cfsnippets">
    SELECT DEPARTMENT, FirstName + ' ' + LastName
    AS FullName
    FROM EMPLOYEES
    ORDER BY DEPARTMENT
</CFQUERY>

<!--- Build the tree control --->
<CFFORM NAME="form1" ACTION="submit.cfm"
    METHOD="Post">

<CFTREE NAME="tree1"
    HSCROLL="no"
    VSCROLL="no"
    BORDER="yes"
    HEIGHT="350"
    REQUIRED="yes">

<CFTREEITEM VALUE="Department, FullName"
    QUERY="myquery"
    QUERYASROOT="Department"
    IMG="cd,folder">

</CFTREE>

<BR><INPUT TYPE="Submit" VALUE="Submit">
</CFFORM>
```

Note how comma-separated items in the IMG and the VALUE attributes correspond. The first column, Department, is represented with one of ColdFusion's built-in CFTREE images, the CD image. The second column, FullName, is represented with another built-in image, the folder. If the IMG attribute is left out altogether, ColdFusion uses the folder image for all levels in the tree.

The resulting tree control looks like this:



If the user selects the name "Peter Jacobsen" in this tree, the following form variables are returned by ColdFusion:

```
form.tree1.node = Peter Jacobsen
form.tree1.path = Department\Sales\Peter Jacobsen
```

You can specify the backslash character used to delimit each element of the path form variable in the CFTREE DELIMITER attribute.

Note The following tree examples all use the result set from the CFQUERY above. The datasource, a Microsoft Access database called cfsnippets.mdb, is installed with CFAS. To run any of the tree examples, just reference “myquery” or drop the query into your test template.

CFTREE form variables

The CFTREE tag allows you to 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 CFTREE ACTION attribute:

- form.*treename*.node — Returns the node of the user selection.
- form.*treename*.path — Returns the complete path of the user selection, in the form: root\node1\node2\node_n*value*

The root part of the path is only returned if you set the COMPLETEPATH attribute of CFTREE to YES; otherwise, the path value starts with the first node.

Input validation with CFTREE

With CFTREE controls, there is no VALIDATE attribute with which, using other CFFORM controls, you can validate a number of data types. However, you can use the REQUIRED attribute in CFTREE to force a user to select an item from the tree control. In addition, you can specify a JavaScript in the ONVALIDATE attribute to perform validation.

Structuring Tree Controls

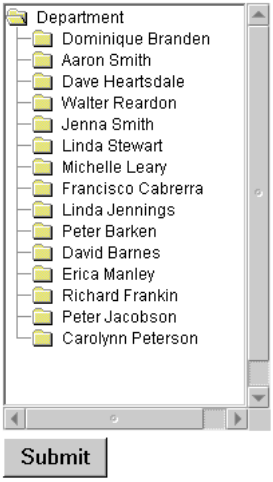
Tree controls built with CFTREE can be very complex. Knowing how to specify the relationship between multiple CFTREEITEM entries will help you handle even the most labyrinthine of CFTREE constructs.

Example: One-level tree control

The following example CFTREE code uses the CFQUERY consists of a single root and a number of individual items:

```
<CFFORM NAME="form1" ACTION="submit.cfm">
    <CFTREE NAME="tree1">
        <CFTREEITEM VALUE="FullName"
        QUERY="myquery"
        QUERYASROOT="Department">
    </CFTREE>

<BR><INPUT TYPE="submit" VALUE="Submit">
</CFFORM>
```

The resulting tree looks like this:



Example: Multilevel tree control

When populating a CFTREE, you manipulate the structure of the tree by specifying a TREEITEM parent. In this example, every TREEITEM, except the top level, specifies a parent. The PARENT attribute allows your CFTREE to show the relationships between elements in the tree control.

```
<CFFORM NAME="form1" ACTION="cfform_submit.cfm"
    METHOD="Post">

<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">
    <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">
    <CFTREEITEM VALUE="QA"
        PARENT="Divisions" IMG="folder">
    <CFTREEITEM VALUE="Product One"
        PARENT="QA">
    <CFTREEITEM VALUE="Product Two" PARENT="QA">
    <CFTREEITEM VALUE="Product Three"
        PARENT="QA">
    <CFTREEITEM VALUE="Support"
        PARENT="Divisions" IMG="fixed">
    <CFTREEITEM VALUE="Product Two"
        PARENT="Support">
    <CFTREEITEM VALUE="Sales"
        PARENT="Divisions" IMG="cd">
    <CFTREEITEM VALUE="Marketing"
        PARENT="Divisions" IMG="document">
    <CFTREEITEM VALUE="Finance"
        PARENT="Divisions" IMG="element">
</CFTREE>

</CFFORM>
```

The resulting tree control looks like this:

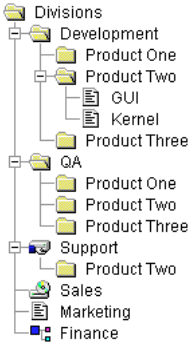


Image names in a CFTREE

When you use the TYPE="Image" attribute, ColdFusion attempts to display an image corresponding to the value in the column, which can be a built in ColdFusion image name, or an image of your choice in the `cfide\classes` directory or subdirectory, referenced with a relative URL.

The built-in image names are:

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

Using commas in CFTREEITEM

Commas are used to separate CFTREEITEM, VALUE, DISPLAY, IMG, and HREF attribute values. You use commas to separate column names in a CFTREE you want to populate with data from a query. As in the following example, two columns returned by the query are specified in the VALUE attribute:

```
<CFTREE NAME="tree1" VSCROLL="no"
    HSCROLL="no" BORDER="no">

    <CFTREEITEM VALUE="Department,LastName"
        QUERY="myquery"
        QUERYASROOT="Company B"
        IMG="folder,folder,document"
        PARENT="Company B">

</CFTREE>
```

In this tree control, Department and LastName are returned by the query. The tree is constructed with LastName entries organized under Department names (assuming the query included an ORDER BY clause on DEPARTMENT). Note also the value of the IMG attribute. Since the tree control consists of three separate levels, you can specify a different built-in or custom image for each level of the tree control by separating each image name with a comma.

Embedding URLs in a CFTREE

The HREF attribute in the CFTREEITEM tag allows you to designate tree items as links. To use this feature in a CFTREE, you simply define the destination of the link in the HREF attribute of CFTREEITEM.

Example: Adding Web links

```
<CFFORM ACTION="submit.cfm">

    <CFTREE NAME="oak"
        HIGHLIGHTHREF="yes"
        HEIGHT="100"
        WIDTH="200"
        HSPACE="100"
        VSPACE="6"
        HSCROLL="no"
        VSCROLL="no"
        BORDER="no"
        DELIMITER="?">

        <CFTREEITEM VALUE="Important Links">
        <CFTREEITEM VALUE="Allaire Home"
            PARENT="Important Links"
            IMG="document"
            HREF="http://www.allaire.com">
        <CFTREEITEM VALUE="Allaire Forums"
            PARENT="Important Links"
            IMG="document"
            HREF="http://forums.allaire.com">

    </CFTREE>
</CFFORM>
```

The resulting tree control looks like this:

The APPENDKEY attribute in CFTREEITEM

When a user selects a tree item and submits the form, the CFTREEITEMKEY variable is appended to the URL passed to the application page specified in the CFFORM ACTION attribute, in the form:

```
http://myserver.com?CFTREEITEMKEY=selected_value
```

You can disable this key by setting the APPENDKEY attribute in the CFTREE tag to No.

The TARGET attribute in CFTREEITEM

Use the TARGET attribute in the CFTREEITEM tag to specify a target for the link you've specified in the HREF attribute. When populating a CFTREE with data from a CFQUERY, you can separate TARGET attribute values in a comma-separated list to correspond with each level of linked tree items in your CFTREE, as shown here:

```
TARGET="FRAME_BODY,_blank,_top"
```

Data Grids with CFGRID

The CFGRID tag, another Java applet-based CFFORM control, allows you to build CFFORM grid controls. A grid control resembles a spreadsheet table and can contain data populated from a CFQUERY or from other sources of data. As with other CFFORM tags, CFGRID offers a wide range of data formatting options as well as the option of validating user selections with a JavaScript validation script.

Other CFGRID features include:

- Alphanumeric sorting of data in a grid
- Data updates, inserts and deletes
- Images can be embedded in a grid

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.

Note If you specify a CFGRID tag with a QUERY attribute defined and no corresponding CFGRIDITEM attribute, the default grid that is created contains all the columns in the query.

CFGRID is used with the CFGRIDCOLUMN tag, much as CFTREE uses CFTREEITEM. In CFGRID, you define a wide range of row and column formatting options, as well as a query name, selection options, and so on. You use the CFGRIDCOLUMN tag to define individual columns in the grid.

Although the CFGRID tag includes a large number of attributes, the basics of building a CFGRID are very straightforward, as you'll see.

Populating a grid from a query

The following example shows a very basic CFGRID populated with data from a CFQUERY:

```
<CFQUERY NAME="getdata" DATASOURCE="cfsnippets">
    SELECT * FROM Employees
</CFQUERY>

<CFFORM NAME="Form1" ACTION="submit.cfm" METHOD="Post">

    <CFGRID NAME="employee_grid" QUERY="getdata"
            SELECTMODE="single">
        <CFGRIDCOLUMN NAME="Employee_ID">
        <CFGRIDCOLUMN NAME="LastName">
        <CFGRIDCOLUMN NAME="Department">
    </CFGRID>

<BR><INPUT TYPE="Submit" VALUE="Submit">
</CFFORM>
```

The resulting CFGRID looks like this:

	Employee Id	Lastname	Department
1	1	Peterson	Sales
2	2	Heartsdale	Accounting
3	3	Stewart	Administration
4	4	Smith	Accounting
5	5	Barken	Engineering
6	6	Jennings	Engineering
7	7	Jacobson	Sales
8	8	Frankin	Sales
9	9	Smith	Administration
10	10	Manley	Sales
11	11	Cabrerra	Engineering
12	12	Leary	Engineering
13	13	Branden	Accounting
14	14	Reardon	Administration
15	15	Barnes	Sales

Submit

Note If you specify a CFGRID tag with a QUERY attribute defined and no corresponding CFGRIDITEM attributes, the default grid that is created contains all the columns in the query.

Hiding columns in a grid

You can use the CFGRIDCOLUMN DISPLAY attribute to hide columns you want to retrieve from a data source but not expose to an end user, such as a customer ID or other primary key column. In the following example, the Employee ID column is retrieved, but not displayed:

```
<CFQUERY NAME="getdata" DATASOURCE="cfsnippets">
    SELECT * FROM Employees
</CFQUERY>

<CFFORM NAME="Form1"
    ACTION="submit.cfm"
    METHOD="Post"
    ENABLECAB="Yes">

    <CFGRID NAME="grid1" QUERY="getdata"
            SELECTMODE="single">
        <CFGRIDCOLUMN DISPLAY="No"
            NAME="Employee_ID">
        <CFGRIDCOLUMN NAME="LastName">
        <CFGRIDCOLUMN NAME="Department">
    </CFGRID>

<BR><INPUT TYPE="Submit" VALUE="Submit">

</CFFORM>
```

Creating an Updateable Grid

You can build grids so that end users are allowed to edit data. These edits can be the basis for changes you make to a data source either by forming queries with CFQUERY or by using the CFGRIDUPDATE tag, which passes edits made to grid data directly to your data source.

Individual cell data can be edited and rows can be inserted, deleted or updated. You enable this facility by specifying SELECTMODE="EDIT" in the CFGRID tag and by enabling the INSERT or DELETE attributes in CFGRID. A grid can now provide data source management, offer data input and review, as well as display tabular data in a grid format.

There are essentially two ways to use an updateable grid to make changes to your ODBC data sources. You can create a page to which you pass the CFGRID form variables and in that page perform CFQUERY operations to update data source records. Or you can pass grid edits to a page that includes the CFGRIDUPDATE tag, which passes data directly to the data source. Although using CFQUERY gives you

complete control over interactions with your data source, CFGRIDUPDATE provides a much simpler interface for operations that do not require the same level of control.

Editing data in a CFGRID

To enable grid editing, you use the SELECTMODE="EDIT" attribute. When enabled, a user can edit cell data and insert or delete grid rows. Users change the content of a cell by clicking on it and editing its contents using simple editing operations. When enabled users can select a row and delete it, or select a row and insert a new row. When a CFFORM containing a CFGRID is submitted, data about changes to grid cells are stored in one-dimensional arrays you can reference like any other ColdFusion array.

The following arrays are created to keep track of edits to grid rows and cells:

Arrays Used to Store Grid Cell Edit Information	
Array reference	**Description**
gridname.colname [*row_index*]	Stores the new value of an edited grid cell
gridname.Original.*colname* [*row_index*]	Stores the original value of the edited grid cell
gridname.RowStatus.Action [*row_index*]	Stores the edit type made against the edited grid cell.

For example, you have an updateable CFGRID called "mygrid" consisting of two displayable columns, col1, col2, and one hidden column, col3. When an end user selects and changes data in a row, arrays are created to store the original values for all columns as well as the new column values for rows that have been updated, inserted, or deleted.

```
mygrid.col1[ row_index ]
mygrid.col2[ row_index ]
mygrid.col3[ row_index ]
mygrid.original.col1[ row_index ]
mygrid.original.col2[ row_index ]
mygrid.original.col3[ row_index ]
```

Where *row_index* is the array index containing the grid data.

If the end user makes a change to a single cell in col2, you can reference the edit operation, the original cell value, and the edited cell value in the following arrays:

```
<CFSET edittype = mygrid.RowStatus.Action[1]><BR>
<CFSET new_value = mygrid.col2[1]><BR>
<CFSET old_value = mygrid.original.col2[1]>
```

Specifying alternate text for the Insert or Delete buttons

If you want the Insert or Delete buttons in an updateable grid to use text other than "Insert" or "Delete," you can specify alternate text in the INSERTBUTTON and DELETEBUTTON attributes.

Multi-row edits

The use of arrays to track changes allows ColdFusion to manage changes to more than one row in a CFGRID. ColdFusion coordinates entries in the arrays used to store edit type (Update, Insert, or Delete), with arrays that store original grid data and edited grid data. For each grid cell edit, an entry is created in the RowStatus array, and corresponding entries are made in the arrays that store the new cell value and the original cell value.

Sorting grid data

The CFGRID SORT attribute allows you to include sort buttons in your grid control. When enabled, sort buttons are automatically added to the grid. When clicked, data is sorted in the selected column. ColdFusion sorts columns either as text or as numeric data.

The following CFGRID attributes are available for defining various options relating to sorting data:

- PICTUREBAR — When Yes, an image button is substituted for the Sort text button.
- SORTASCENDINGBUTTON — You can specify the text to use for the Sort Descending button if you don't want to use the default, which is "`A -> Z`".
- SORTDESCENDINGBUTTON — You can specify the text to use for the Sort Descending button if you don't want to use the default, which is "`Z <- A`".

Note Users must first select a column before clicking a sort button.

Example: Editable grid

This grid example demonstrates an updateable grid, in which the SELECTMODE attribute is "Edit" and the INSERT and DELETE attributes are "yes." When the form is submitted, the `handle_grid.cfm` page displays the type of edits that were made to the data source, Update, Delete, or Insert and interacts with the data source directly to perform the corresponding actions.

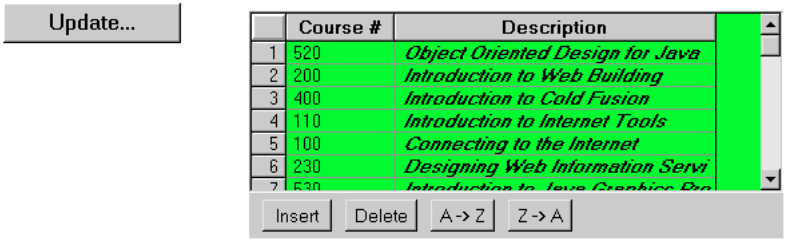
Grid.cfm

```
<HTML>
<HEAD>
    <TITLE>Simple Update Grid Example</TITLE>
</HEAD>

<CFQUERY NAME="CourseList"
```

```
    DATASOURCE="cfsnippets">
    SELECT * FROM Courses
</CFQUERY>

<BODY BGCOLOR="#FFFFFF">

<CFFORM NAME="GridForm"
    ACTION="handle_grid.cfm">

<CFGRID NAME="course_grid"
        HEIGHT=170
        WIDTH=400
        HSPACE=10
        VSPACE=6
        ALIGN="RIGHT"
        SELECTCOLOR="white"
        SELECTMODE="edit"
        ROWHEADERS="YES"
        ROWHEADERWIDTH=25
        ROWHEADERALIGN="right"
        COLHEADERS="YES"
        QUERY="CourseList"
        GRIDDATAALIGN="left"
        BGCOLOR="green"
        INSERT="YES"
        DELETE="YES"
        SORT="YES"
        MAXROWS=60>

    <CFGRIDCOLUMN NAME="course_id"
        HEADER="Course ID"
        WIDTH=80
        ITALIC="NO"
        HEADERALIGN="center"
        HEADERITALIC="NO"
        HEADERBOLD="YES"
        DISPLAY="NO">

<CFGRIDCOLUMN NAME="number"
        HEADER="Course ##"
        WIDTH=80
        ITALIC="NO"
        HEADERALIGN="center"
        HEADERITALIC="NO"
        HEADERBOLD="YES"
        DISPLAY="YES"
        SELECT="YES">

    <CFGRIDCOLUMN NAME="description"
        HEADER="Description"
        WIDTH=240
        ITALIC="No"
        HEADERALIGN="center"
        HEADERITALIC="No"
```

```
        HEADERBOLD="Yes"
        BOLD="Yes"
        ITALIC="Yes"
        DISPLAY="Yes">

    </CFGRID>

<!---
<H3>Editable Grid</H3>

This is a grid that is populated <BR>
from a query. The select mode is <BR>
update. The description column<BR>
is presented in Bold and Italic.<BR><BR> --->

<INPUT TYPE="Submit" VALUE=" Push me... "> <BR>

</CFFORM>

</BODY>
</HTML>
```

The grid and action button look like this:



Changes you make in the course list are reflected in the Courses table in the cfsnippets data source. You can view the table in Studio by going to the DB tab and opening the data source.

Handle_grid.cfm

```
<HTML>
<HEAD>
    <TITLE>Catch submitted grid values</TITLE>
</HEAD>
<BODY>

<H3>Grid values for FORM.Course_grid row updates</H3>

<CFIF IsDefined("form.course_grid.rowstatus.action")>

    <CFLOOP INDEX = "Counter" FROM = "1" TO =
        #ArrayLen(form.course_grid.rowstatus.action)#>
```

```
    <CFOUTPUT>
    The row action for #Counter# is:
    #form.course_grid.rowstatus.action[Counter]#
    <BR><BR>
    </CFOUTPUT>

    <CFIF form.course_grid.rowstatus.action[Counter] IS "D">

    <CFQUERY NAME="InsertNewCourse"
        DATASOURCE="cfsnippets">
        DELETE from courses
        WHERE course_id=#form.course_grid.original.course_id[Counter]#
    </CFQUERY>

<CFELSEIF form.course_grid.rowstatus.action[Counter] IS "U">

    <CFQUERY NAME="UpdateExistingCourse"
        DATASOURCE="cfsnippets">
        UPDATE courses
        SET description='#form.course_grid.description[Counter]#' ,
        "Number"='#form.course_grid.number[Counter]#'
        WHERE
        course_id=#form.course_grid.original.course_id[Counter]#
    </CFQUERY>

<CFELSEIF form.course_grid.rowstatus.action[Counter] IS "I">

    <CFQUERY NAME="InsertNewCourse"
        DATASOURCE="cfsnippets">
        INSERT into courses
        ("Number", description)
        VALUES ('#form.course_grid.number[Counter]#',
        '#form.course_grid.description[Counter]#')
    </CFQUERY>

    </CFIF>
    </CFLOOP>
</CFIF>

</BODY>
</HTML>
```

Using CFGRIDUPDATE

The CFGRIDUPDATE tag allows you to perform updates to a data source directly from a CFGRID. You don't need to form a CFQUERY to perform updates, CFGRIDUPDATE handles the entire transaction, taking grid cell edit information directly from the CFGRID tag. It's a much simpler, but slightly more abstracted method for updating data from a grid control.

In the earlier example, data from an edited grid was passed to a page that used CFQUERY tags in a CFIF construct to update a data source. For many updates, CFGRIDUPDATE is easier.

For example, the following code accepts grid update data from a grid named "Courses" originating in a separate page.

```
<CFGRIDUPDATE GRID="Courses"
    DATASOURCE="CF 4.0 Examples"
    TABLENAME="Courses"
    KEYONLY="NO">
```

The arrays that store information about changes to CFGRID data (as well as original grid cell values) are passed to CFGRIDUPDATE when the form is submitted. CFGRIDUPDATE uses these arrays to build the SQL necessary to perform the data source updates.

The edits originating in the Courses grid could consist of multiple cell edits, row inserts, and row deletions. CFGRIDUPDATE passes all of these edits to the specified data source, saving you the task of having to craft CFQUERY statements to do the same work. Very handy.

The KEYONLY attribute

CFGRIDUPDATE includes the KEYONLY attribute, which allows you to force ColdFusion to compare the original value of the updated fields with the data in the corresponding table field. If they are the same, that is, if no other process has changed the data since the grid was edited, the update passes. If the comparison fails an error is generated. Use KEYONLY="No" when you want to be sure that no other process has updated the same data. Use KEYONLY="Yes" if no other process can potentially change the same data.

Embedding images in a grid

The CFGRIDROW tag allows you to place images in a grid cell. You do this by first using the TYPE="IMAGE" attribute in a CFGRIDCOLUMN tag to tell ColdFusion that you want data in the current column to be interpreted as an image. You can use one of the built-in image names ColdFusion provides (same as those for CFTREEITEM) or specify an image file of your choice. The built-in image names are as follows:

Built-in Image Names	
Image	**Example**
cd	
computer	
document	

Built-in Image Names	
Image	**Example**
element	
floppy	
folder	
fixed	
remote	

Here's part of a CFGRID showing the CFGRIDCOLUMN tag using TYPE="IMAGE" to define the column that will contain images, and the CFGRIDROW tags that populate each column with row data.

```
...
<CFGRIDCOLUMN
    NAME="dept_name"
    HEADER="Dept"
    SELECT="NO"
    DATAALIGN="Center"
    WIDTH=40
    TYPE=IMAGE>

<CFGRIDCOLUMN
    NAME="emp_lname"
    HEADER="Name">

<CFGRIDROW DATA="folder,Jones">
<CFGRIDROW DATA="document,Smith">
...
```

Note In this example, commas are used to separate the image name that appears in the first column from the data that appears in the second column.

Using your own images in a grid

When you want to use your own image files instead of the built-in ColdFusion images, you need to specify the relative path to the directory where the image file can be found, as well as the image file name itself.

When you specify the relative path to the images you want to use, note that the path is relative to the location of the Java class that enables the CFGRID control. Ordinarily, this will be:

web_root\cfide\classes\images

You can specify a location relative to this directory, or you can simply place your files in the cfide\classes\images folder and use the image name without an extension, just as you would use one of the built-in image names.

In the following code chunk, the CFGRIDROW tags use external images found in the *web_root*\images directory.

```
...
<CFGRIDCOLUMN
    NAME="dept_name"
    HEADER="Dept"
    SELECT="NO"
    DATAALIGN="Center"
    WIDTH=40
    TYPE=IMAGE>

<CFGRIDCOLUMN
    NAME="emp_lname"
    HEADER="Name">

<CFGRIDROW DATA="..\..\..\images\icon1.gif,Jones">
<CFGRIDROW DATA="..\..\..\images\icon2.gif,Smith">
...
```

Grid Data Selection Options

You can control how you want users to interact with your grid control. You can limit a user to simply browsing data displayed in the grid control, or you can enable several different selection options by specifying values in the SELECTMODE attribute.

You can specify the following selection behaviors in the SELECTMODE attribute:

- Single — User selections are limited to a single cell in the grid control.
- Column — When a user selects a cell, data from the column containing the selected cell is included in the selection.
- Row — When a user selects a cell, data from the row containing the selected cell is included in the selection.
- Browse — Users cannot select cells in the grid control.
- Edit — Users can edit cell data.

Select mode and form variables

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

- When SELECTMODE="Single" grid data is returned as *grid_name.selectedname* and the selected value.
- When SELECTMODE="Column" grid data is returned as a comma-separated list of all the values for the selected column.

- When SELECTMODE="Row" grid data is returned as *grid_name.colum1_name* and *grid_name.column2_name* and their respective values for the selected row.
- When SELECTMODE="Browse" no selection data is returned.
- When SELECTMODE="Edit" three one-dimensional arrays are created if cell data is changed.

Using the URL attribute

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

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

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

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

The HREF attribute

You can use the HREF attribute to associate a hyperlink with a selected row or cell. ColdFusion interprets the value of the HREF attribute as either a query column that stores the text of a link or the text of the hyperlink itself. The destination specified either literally in the HREF attribute or by reference to the query column is resolved relative to the current application page.

In the following code fragment, the HREF attribute is used in two CFGRIDCOLUMN blocks. Because the HREF attribute refers to a CFGRIDCOLUMN, each row is associated with a different URL, based on the value of the dept_url column in the selected row. Note also that the last CFGRIDCOLUMN is a hidden column used to hide the value of the dept_url field.

```
<CFFORM NAME="GridForm"
    ACTION="catch_grid10.cfm"
    TARGET="Lower">

<CFGRID NAME="grid_ten"
    HEIGHT=170
    WIDTH=400
    HSPACE=10
    VSPACE=6
```

```
    ALIGN="Right"
    SELECTMODE="Row"
    ROWHEADERS="Yes"
    COLHEADERS="Yes"
    QUERY="DeptList"
    GRIDDATAALIGN="Left"
    HIGHLIGHTHREF="No"
    APPENDKEY="No"
    SORT="Yes">

    <CFGRIDCOLUMN NAME="dept_id"
        HEADER="Department"
        WIDTH=80
        ITALIC="No"
        HEADERALIGN="Center"
        HEADERITALIC="No"
        HEADERBOLD="Yes"
        HREF="dept_url"
        TYPE="Numeric">

    <CFGRIDCOLUMN NAME="dept_name"
        HEADER="Name"
        ITALIC="No"
        HEADERALIGN="Center"
        HEADERITALIC="No"
        HEADERBOLD="Yes"
        HREF="dept_url">

    <CFGRIDCOLUMN NAME="dept_url"
        DISPLAY="No">

    </CFGRID>

    <INPUT TYPE="Submit" VALUE="Submit"> <BR>

</CFFORM>
```

The APPENDKEY attribute in CFGRIDKEY

When a user selects a grid item and submits the form, the CFGRIDKEY variable is appended to the URL passed to the application page specified in the CFFORM ACTION attribute, in the form:

```
http://myserver.com?CFGRIDMKEY=selected_value
```

You can disable this key by setting the APPENDKEY="NO".

Building Slider Bar Controls

The CFSLIDER control is one of ColdFusion's Java applet-based CFFORM controls. With it you can create a slider control and define a wide range of formatting options for slider label text, as well as slider scale increments, range, positioning, and behavior.

As with CFTREE and CFGRID, input validation can be serviced with a JavaScript specified in the ONVALIDATE attribute.

Example: CFSLIDER control

The following example shows a simple CFSLIDER control:

```
<CFFORM NAME="Form1" ACTION="submit.cfm"
    METHOD="Post">

    <CFSLIDER NAME="myslider"
        GROOVECOLOR="black"
        BGCOLOR="white"
        TEXTCOLOR="black"
        FONT="Trebuchet MS"
        BOLD="yes"
        RANGE="0,1000"
        SCALE="10"
        VALUE="640"
        FONTSIZE="24"
        LABEL="Slider %value%"
        WIDTH="400">

</CFFORM>
```

The resulting slider looks like this:

CFSLIDER form variable

The value of the form variable passed from a CFSLIDER control to a ColdFusion application page is determined by the position of the slider on the scale. The form variable is passed as:

```
slider_name=slider_value
```

In the earlier example, the form variable would have been passed as:

```
myslider=slider_value
```

Formatting options with CFSLIDER

As with other CFFORM controls, CFSLIDER offers many formatting, positioning, and alignment options. You can specify colors for the groove in which the slider knob moves, as well as label font name, size, boldface, italics, and color.

Building Text Entry Boxes

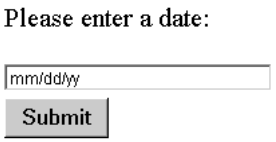
The CFTEXTINPUT tag is a very close relative to the HTML INPUT=text tag. With CFTEXTINPUT, however, you can also specify font and alignment options, as well as enable one of two input validation methods using either a JavaScript or the VALIDATE attribute in CFTEXTINPUT.

Example: CFTEXTINPUT control

The following example shows a basic CFTEXTINPUT control. This example validates a date entry, which means that a user must enter a valid date in the form *mm/dd/yy*.

```
<BR>Please enter a date:
<CFFORM NAME="Form1" ACTION="cfform_submit.cfm" METHOD="Post">

    <CFTEXTINPUT NAME="entertext"
        VALUE="mm/dd/yy"
        MAXLENGTH="10"
        VALIDATE="date"
        FONT="Trebuchet MS">

    <BR><INPUT TYPE="Submit"
            VALUE="Submit">

</CFFORM>
```

The CFTEXTINPUT looks like this:



CFTEXTINPUT form variable

The value of the form variable passed from a CFTEXTINPUT control to a ColdFusion application page is determined by the entry in the CFTEXTINPUT control. The form variable is passed as:

```
textinput_name=textinput_value
```

In the example just above, the form variable would have been passed as:

```
entertext=textinput_value
```

So in the destination application page, the form variable is referenced as #entertext#.

Input validation with CFTEXTINPUT

You can validate user input for the CFTEXTINPUT control on the following data formats:

Input Validation Controls	
VALIDATE Entry	**Description**
`date`	Verifies US date entry in the form mm/dd/yyy.
`Eurodate`	Verifies valid European date entry in the form dd/mm/yyyy.
`Time`	Verifies a time entry in the form hh:mm:ss.
`Float`	Verifies a floating point entry.
`Integer`	Verifies an integer entry.
`Telephone`	Verifies a telephone entry. Telephone data must be entered as ###-###-####. The hyphen separator (-) can be replaced with a blank. The area code and exchange must begin with a digit between 1 and 9.
`Zipcode`	(U.S. formats only) Number can be a 5-digit or 9-digit zip in the form #####-####. The hyphen separator (-) can be replaced with a blank.
`Creditcard`	Blanks and dashes are stripped and the number is verified using the mod10 algorithm.
`social_security_number`	Number must be entered as ###-##-####. The hyphen separator (-) can be replaced with a blank.

Building Drop-Down List Boxes

The drop-down list box you can create with CFSELECT is a close relative of the HTML SELECT tag. CFSELECT gives you more control over user inputs, error handling, and allows you to populate the selection list from a query.

Populating a CFSELECT with query data

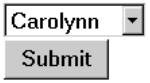
When you populate a CFSELECT with data from a query, you only need to specify the name of the query that is supplying data for the CFSELECT and the query column name for each list element you want to display.

Example: Populate a CFSELECT from a data column:

```
<CFQUERY NAME="myquery"
    DATASOURCE="cfsnippets">
    SELECT * FROM Employees
</CFQUERY>

<CFFORM NAME="Form1" ACTION="submit.cfm"
    METHOD="Post">

    <CFSELECT NAME="myselectbox"
        QUERY="myquery"
        VALUE="Employee_ID"
        DISPLAY="FirstName"
        REQUIRED="yes"
        MULTIPLE="yes"
        SIZE="8">
    </CFSELECT>

    <BR><INPUT TYPE="Submit"
            VALUE="Submit">

</CFFORM>
```

The resulting drop-down list box looks like this:

Note that because the MULTIPLE attribute is used, the user can select multiple entries in the select box. When MULTIPLE is omitted or SINGLE is explicitly used and the SIZE attribute is set to zero, the resulting CFSELECT looks like this:



One other thing to note about this example: since the VALUE tag specifies the primary key for the Employee table, this data is used in the form variable that is passed to the application page specified in ACTION.

Building Form Controls

Like its HTML counterpart the INPUT tag, CFINPUT supports the following form input controls:

- Radio buttons
- Check boxes
- Text entry boxes
- Password entry boxes

CFINPUT also allows is input validation using one of two methods: attributes in the CFINPUT tag, or a JavaScript that you specify.

Example: CFINPUT controls

The following application page contains a form with a variety of CFINPUT controls:

```
<CFFORM NAME="Form1" ACTION="submit.cfm" METHOD="Post">

<TABLE CELLPADDING=5 border=0>
    <TR><TD>Please enter your user login:<BR>
    <CFINPUT TYPE="text" NAME="loginID" VALUE="name"></TD></TR>
    <TR><TD>Please also enter your password:<BR>
    <CFINPUT TYPE="password" NAME="pwd" VALUE="password"></TD></TR>
    <TR><TD>Please select one:<BR>
    <CFINPUT TYPE="radio" NAME="radio1" VALUE="select1">Embodied<BR>
    <CFINPUT TYPE="radio" NAME="radio1" CHECKED="yes" VALUE="select2">
    Disembodied<BR>
    <CFINPUT TYPE="radio" NAME="radio1" VALUE="select3">Don't
    Know</TD></TR>
    <TR><TD>Make your selections here:<BR>
    <CFINPUT TYPE="checkbox" NAME="checkbox1" VALUE="one">Derrida<BR>
    <CFINPUT TYPE="checkbox" NAME="checkbox1" CHECKED="yes"
    VALUE="two">Foucault<BR>
    <CFINPUT TYPE="checkbox" NAME="checkbox1" VALUE="three">
    Kristeva</TD></TR>
    <TR><TD><INPUT TYPE="Submit" VALUE="Submit"></TD></TR>
</TABLE>

</CFFORM>
```

This code generates the following form:

Embedding Java Applets

The CFAPPLET tag allows you to embed Java applets in a CFFORM. To use CFAPPLET, you must first register your Java applet using the ColdFusion Administrator Applets page. In the Administrator, you define the interface to the applet, encapsulating it so that each invocation of the CFAPPLET tag is very simple.

CFAPPLET offers several advantages over using the HTML APPLET tag:

- Return values — Since CFAPPLET requires a form field name attribute, you can avoid having to code 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 — Since the applet's interface is defined in the Administrator, each instance of the CFAPPLET tag in your pages only needs to reference the applet's name and specify a form variable name.
- Parameter options — You can override parameter values you defined in the Administrator by specifying the parameter value pair in CFAPPLET. Unless overridden, ColdFusion uses the parameter value pairs you defined in the Administrator.

When an applet is registered, enter just the applet source and the form variable name:

```
<CFAPPLET APPLETSOURCE="Calculator"
    NAME="calc_value">
```

By contrast, with the HTML APPLET tag, you'd have to invoke all the applet's parameters every time you wanted to use it in a ColdFusion page.

Registering a Java applet

Before you can use a Java applet in your ColdFusion pages, you must first register the applet in the Administrator.

To register a Java applet:

1. Open the ColdFusion Administrator by clicking on the Administrator icon in the ColdFusion Program group and entering the Administrator password (if required).
2. Click the Applets button to open the Registered Applets page.
3. Enter a name for the applet you want to register and click Register New Applet. Enter the information your applet requires, and choose the height, width, vertical and horizontal space, and alignment you want.

Applet registration fields are explained in the following table.

Applet Registration Fields	
Field	**Description**
Codebase	Enter the base URL of the applet: the directory that contains the applet components. The applet class files must be located within the web browser root directory. Example: `http://servername/classes`
Code	This is the name of the file that contains the applet subclass. The filename is relative to the codebase URL. The *.class file extension is not required.
Method	Enter the method name in the applet that returns a string value. You use this method name in the NAME attribute of the CFAPPLET tag to populate a form variable with the method's value. If the applet has no method, leave this field blank.
Height	Enter a measurement in pixels for the vertical space for the applet.
Width	Enter a measurement in pixels for the horizontal space for the applet.
Vspace	Enter a measurement in pixels for the space above and below the applet.

Applet Registration Fields (Continued)	
Field	**Description**
Hspace	Enter a measurement in pixels for the space on each side of the applet.
Align	Choose the alignment you want.
Java Not Supported Message	This message is displayed by browsers that do not support Java applets. If you want to override this message, you specify a different message in the CFAPPLET NOTSUPPORTED attribute.
Parameter Name	Enter a name for a required applet parameter. Your Java applet will typically provide the parameter name needed to use the applet. Enter each parameter in a separate parameter field.
Value	For every parameter you enter, define a default value. Your applet documentation will provide guidelines on valid entries.

Click Create to complete the process.

Using CFAPPLET to embed an applet

Once you've registered 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. Since the applet has been registered, and each applet parameter defined with a default value, you can invoke the applet with a very 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 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:

```
<CFAPPLET APPLETSOURCE="myapplet"
    NAME="applet1_var"
    HEIGHT=400
    WIDTH=200
    VSPACE=125
    HSPACE=125
    ALIGN="left">
```

Overriding parameter values

You can also override the values you assigned to applet parameters in the Administrator by providing new values for any parameter. Note that in order to override a parameter, you must have already defined the parameter and a default value for it in the ColdFusion Administrator Applets page.

```
<CFAPPLET APPLETSOURCE="myapplet"
    NAME="applet1_var"
    Param1="registered parameter"
    Param2="registered parameter">
```

Handling form variables from an applet

The CFAPPLET tag requires you to specify a form variable name for the applet. This variable, referenced like other ColdFusion form variables, `form.`*`variable_name`* holds the value the applet method provides when it is executed in the CFFORM.

Not all Java applets return values. Some, like many graphical widgets, do not return a specific value; they do their flipping, spinning, fading, exploding, and that's that. For this kind of applet, the method field in the Administrator remains empty. Other applets, however, do have a method that returns a value. You can only use one method for each applet you register. If an applet includes more than one method that you want to access, you can register the applet with a unique name an additional time for each method you want to use.

To reference a Java applet return value in your application page:

1. Specify the name of the method in the Register New Applet page of the ColdFusion Administrator.
2. Specify the method name in the NAME attribute of the CFAPPLET tag when you code your CFFORM.

When your page executes the applet, a form variable is created with the name you specified. If you don't specify a method, no form variable is created.

CHAPTER 13

Managing Files on the Server

The CFFILE, CFDIRECTORY, and CFCONTENT tags handle browser/server file management tasks. To perform server-to-server operations, use the CFFTP tag.

Contents

Using CFFILE

The CFFILE tag gives you the ability to work with files on your server in a number of ways:

- Uploading files from a client to the Web server using an HTML form.
- Moving, renaming, copying, or deleting files on the server.
- Reading, writing, or appending to text files on the server.

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.

Uploading Files

File uploading requires that you create two files:

- An HTML form to enter file upload information
- An action page containing the file upload code

Creating a file upload HTML form

HTML forms can be designed in most browsers to give users the ability to upload files. Setting the HTML INPUT tag type to "file" instructs the browser to prepare to read and transmit a file from the user's system to your server. Setting the ENCTYPE FORM attribute to "multipart/form-data" tells the server that the form submission contains an uploaded file.

Example: An HTML form for file upload

```
<FORM ACTION="FileUpload.cfm"
ENCTYPE="multipart/form-data"
METHOD="Post">
<PRE>
    File Name: <INPUT NAME="FileName" TYPE="text">
    File: <INPUT NAME="FileContents" TYPE="file">
<INPUT TYPE="submit" VALUE="Upload File">
</PRE>
</FORM>
```

The user can enter a file path or browse the system and pick a file to send.

Note The FORM attribute ENCTYPE "multipart/form-data" must be included.

Creating a file upload application page

Submitting a file does not save it on the server. When a file is submitted, it is encoded and sent along with the other form data. The CFFILE tag in the file upload application page decodes the file and saves its contents on the server.

Example: Upload a file

The following example shows a CFFILE tag that could be placed in the "FileUpload.cfm" file referred to in the previous example:

```
<CFFILE ACTION="UPLOAD"
    FILEFIELD="FileContents"
    DESTINATION="C:\Web\Uploads\">
```

If the file upload form that requested this application page sent a file named KeyMemo.doc, the file is saved to the server as:

```
c:\Web\Uploads\KeyMemo.doc
```

To save the file under a different name, specify the new file name in the DESTINATION attribute. In this example, “KeyMemo.doc” is saved on the server as “UploadedFile.doc":

```
<CFFILE ACTION="UPLOAD"
    FILEFIELD="FileContents"
    DESTINATION="C:\Web\Uploads\UploadedFile.doc">
```

You could also make any of these attributes dynamic variables. For example, you could set the file name based on information from a database query.

Note The FILEFIELD attribute expects the name of a form field, not the contents of the form field, so you should not enclose the form field in # signs.

Resolving conflicting file names

When a file is saved to the server, there is a risk that another file may already exist with the same name. In the event of this occurrence, there are a number of actions you can take using the NAMECONFLICT attribute.

Example: Resolving a name conflict

The following example will create a unique file name, while retaining the file extension, if there is a name conflict when the file is uploaded:

```
<CFFILE ACTION="Upload"
    FILEFIELD="FileContents"
    DESTINATION="C:\Web\Uploads\"
    NAMECONFLICT="MAKEUNIQUE">
```

Controlling the type of file uploaded

For some applications, you might want to restrict the type of file that is uploaded. For example, you may not want to accept graphic files in a document library.

The ACCEPT attribute is used to restrict the type of file that will be allowed in an upload. When an ACCEPT qualifier is present, the uploaded file's MIME content type must match the criteria specified or an error will occur. ACCEPT takes a comma-separated list of MIME data names, optionally with wildcards.

A file's MIME type is determined by the browser. Common types, like "image/gif" and "text/plain", are registered in your browser.

Example: Restricting file types

This CFFILE specification will only save an image file that is in the GIF format:

```
<CFFILE ACTION="Upload"
    FILEFIELD="UploadFile"
    DESTINATION="c:\uploads\MyImage.GIF"
    NAMECONFLICT="OVERWRITE"
    ACCEPT="image/gif">
```

This CFFILE specification will only save an image file that is either a GIF or a JPEG:

```
<CFFILE ACTION="Upload"
    FILEFIELD="UploadFile"
    DESTINATION="c:\uploads\MyImage.GIF"
    NAMECONFLICT="OVERWRITE"
    ACCEPT="image/gif, image/jpeg">
```

This CFFILE specification will only save an image file, but the format doesn't matter:

```
<CFFILE ACTION="Upload"
    FILEFIELD="UploadFile"
    DESTINATION="c:\uploads\MyImage.GIF"
    NAMECONFLICT="OVERWRITE"
    ACCEPT="image/*">
```

Note Any file will be saved if ACCEPT is omitted, left empty, or contains "*/*".

Setting File and Directory Attributes

File attributes in Windows are defined using the CFFILE ATTRIBUTES attribute. In UNIX, file and directory permissions are defined using the CFFILE and CFDIRECTORY MODE attribute.

UNIX

In UNIX, you can set permissions on files and directories for owner, group, and other. Values for the MODE attribute correspond to octal values for the UNIX `chmod` command:

- 4 = Read only
- 2 = Read/write
- 1 = Read/write/execute

You enter permissions values in the MODE attribute for each type of user: owner, group, other in that order. For example to assign read permissions for all:

```
MODE=444
```

To give a file or directory owner read/write/execute permissions and read only permissions for everyone else:

```
MODE=744
```

Windows

In Windows, you can set the following file attributes:

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

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

Example: Setting file attribute

This example sets the archive bit for the uploaded file:

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

Evaluating the Results of a File Upload

After a file upload is completed, you can retrieve status information using file upload variables. This status information includes a wide range of data about the file, such as the file's name and the directory where it was saved.

The preferred syntax for file upload status variables uses the `CFFILE` prefix, for example, `CFFILE.ClientDirectory`. The `File` prefix is retained for backward compatibility. The file status variables can be used anywhere that ColdFusion variables are used.

The following file upload status variables are available after an upload.

File Upload Variables	
Parameter	**Description**
AttemptedServerFile	Initial name ColdFusion used attempting to save a file, for example, `myfile.txt`. See the "Resolving conflicting file name" section above.
ClientDirectory	Directory location of the file uploaded from the client's system.
ClientFile	Name of the file uploaded from the client's system, such as `myfile.txt`.
ClientFileExt	Extension of the uploaded file on the client's system without a period, for example, `txt` not `.txt`.
ClientFileName	Filename without an extension of the uploaded file on the client's system.
ContentSubType	MIME content subtype of the saved file, such as `gif` for `image/gif`.
ContentType	MIME content type of the saved file, such as `image` for `image/gif`.
DateLastAccessed	Date and time the uploaded file was last accessed.
FileExisted	Indicates (Yes or No) whether or not the file already existed with the same path.
FileSize	Size of the uploaded file.
FileWasAppended	Indicates (Yes or No) whether or not ColdFusion appended the uploaded file to an existing file.
FileWasOverwritten	Indicates (Yes or No) whether or not ColdFusion overwrote a file.

File Upload Variables (Continued)	
Parameter	**Description**
FileWasRenamed	Indicates (Yes or No) whether or not the uploaded file was renamed to avoid a name conflict.
FileWasSaved	Indicates (Yes or No) whether or not ColdFusion saved a file.
OldFileSize	Size of a file that was overwritten in the file upload operation.
ServerDirectory	Directory of the file actually saved on the server.
ServerFile	Filename of the file actually saved on the server.
ServerFileExt	Extension of the uploaded file on the server, without a period, for example, `txt` not `.txt`.
ServerFileName	Filename, without an extension, of the uploaded file on the server.
TimeCreated	Time the uploaded file was created.
TimeLastModified	Date and time of the last modification to the uploaded file.

Use the File prefix to refer to these variables, for example, #File.FileExisted#.

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.

Moving, Renaming, Copying, and Deleting Server Files

With CFFILE, 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 below 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. This makes CFFILE a very powerful tool.

Moving a file (ACTION="MOVE")

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

Example

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

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

Renaming a file (ACTION="RENAME")

CFFILE can be used to rename a file on a server.

Example

The following example renames the file `KeyMemo.doc` to `OldMemo.doc`:

```
<CFFILE ACTION="Rename"
    SOURCE="c:\files\memo\KeyMemo.doc"
    DESTINATION="c:\files\memo\OldMemo.doc">
```

Copying a file (ACTION="COPY")

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

Example

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

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

Deleting a file (ACTION="DELETE")

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

Example

The following example permanently deletes the specified file:

```
<CFFILE ACTION="Delete"
    FILE="c:\files\upload\”oldfile.txt">
```

Reading, Writing, and Appending to a Text File

In addition to managing files on the server, you can use CFFILE to read, create, and modify text files.

This gives you the ability to

- Create log files.
- Generate static HTML documents.
- Use text files to store information that can be brought into Web pages.

Read a text file (ACTION="READ")

You can use CFFILE to read an existing text file. The file is read into a dynamic parameter which 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 find and replace functions to modify the contents.

Example

The following example will create a variable named "Message" which will contain the contents of the file "update.txt:"

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

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

```
<CFOUTPUT>#Message#</CFOUTPUT>
```

Write a text file (ACTION="WRITE")

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

Example

The following example creates a file with the information a user entered into an HTML insert form:

```
<CFFILE ACTION="Write"
    FILE="C:\files\updates\#Form.UpdateTitle#.txt"
    OUTPUT="Created By: #Form.FullName#
    Date: #Form.Date#
    #Form.Content# ">
```

If the user submitted a form in which:

```
UpdateTitle="FieldWork"
FullName="John Lunch"
Date="10/1/98"
Content="We had a wonderful time in Cambridgeport."
```

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

```
Created By: John Lunch
Date: 10/1/98
We had a wonderful time in Cambridgeport.
```

Append to a text file (ACTION="APPEND")

CFFILE can be used to append additional text to the end of an existing text file, for example, when creating log files.

Example

The following example will append the text string "But Davis Square was more fun." to the file FieldWork.txt which was created in the previous example:

```
<CFFILE ACTION="Append"
    DESTINATION="C:\files\updates\FieldWork.txt"
    OUTPUT="<B>But Davis Square was more fun.</B>">
```

Performing Directory Operations

Use the CFDIRECTORY tag to return file information from a specified directory and to create, delete, and rename directories.

As with CFFILE, ColdFusion administrators can disable CFDIRECTORY processing in the ColdFusion Administrator Tags page.

CFDIRECTORY Attributes	
Attribute	**Description**
ACTION	Optional. Defines the action to be taken with directory(ies) specified in DIRECTORY. Valid entries are: • List • Create • Delete • Rename Default is List.
DIRECTORY	Required for all ACTIONs. The name of the directory you want the action to be performed against.

CFDIRECTORY Attributes (Continued)	
Attribute	**Description**
NAME	Required for ACTION="List". Ignored for all other actions. Name of output query for directory listing.
FILTER	Optional for ACTION="List". Ignored for all other actions. Filter to be applied to returned names, for example, "*.cfm"

MODE	Optional. Used only when ACTION="Create" to define the permissions for a directory in Solaris. Ignored in Windows. Valid entries correspond to the octal values (not symbolic) of the Unix chmod command. Permissions are assigned for owner, group, and other, respectively. For example:
	`MODE=644`
	`assigns the owner read/write permissions and group/ other read permission.`
	
	`MODE=666`
	`Assigns read/write permissions for owner, group, and other.`
	
	`MODE=777`
	`Assigns read, write, and execute permissions for all.`

SORT	Optional for ACTION="List". Ignored for all other actions. List of query columns to sort directory listing by. Any combination of columns from query output can be specified in comma separated list. ASC or DESC can be specified as qualifiers for column names. For example, in a CFDIRECTORY tag returning where NAME="mydir", you can sort as follows: `SORT="dirname ASC, filename2 DESC, size, datelastmodified"` Where colname is the name of any column of data returned in the CFDIRECTORY operation.
NEWDIRECTORY	Required for ACTION="Rename". Ignored for all other actions. The new name of the directory specified in the DIRECTORY attribute.

Returning file information (ACTION="LIST")

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

- Name — Directory entry name.

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

Example

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

```
<CFDIRECTORY
    DIRECTORY="c:\winnt\system32"
    NAME="mydirectory"
    SORT="size ASC, name DESC, datelastmodified">

<CFOUTPUT QUERY=”mydirectory”>
    Name: #mydirectory.name# <BR>
    Size: #mydirectory.size# <BR>
    Type: #mydirectory.type# <BR>
    Date last modified: #mydirectory.datelastmodified# <BR>
    Attributes: #mydirectory.attributes#<BR>
    Mode: #mydirectory.mode#<BR>
</CFOUTPUT>
```

CHAPTER 14

Performing File Operations with CFFTP

The CFFTP tag allows you to perform tasks on remote servers via the File Transfer Protocol (FTP). CFFTP allows you to cache connections for batch file transfers.

For server/browser operations, use the CFFILE, CFCONTENT, and CFDIRECTORY tags.

Contents

Establishing a Connection

Using CFFTP involves two distinct types of operations, connecting and transferring files.

Note CFFTP is a COM object and is not supported in Microsoft Windows NT 3.51.

Example: FTP logon

This example shows an FTP connection being opened to a server that requires a username and password.

```
<CFFTP CONNECTION=FTP
    USERNAME="betauser"
    PASSWORD="monroe"
    SERVER="beta.company.com"
    ACTION="Open"
    STOPONERROR="No">
```

Since the USERNAME and PASSWORD attributes are required for CFFTP, you establish an anonymous connection by entering "anonymous" as the username and an email address (by convention) for the password.

File and Directory Operations

Once a connection has been established you can perform file and directory operations to and from the connected FTP server. Use the following CFFTP attributes to do file and directory operations once a CFFTP connection has been established.

CFFTP File and Directory Operation Attributes	
Attribute	**Description**
ACTION	Required if connection is not already cached. If connection caching is used, the ACTION attribute is not required. Determines the FTP operation to perform. Can be one of the following: • ChangeDir • CreateDir • ListDir • GetFile • PutFile • Rename • Remove • GetCurrentDir • GetCurrentURL • ExistsDir • ExistsFile • Exists **Note**: Names of objects (files and directories) are case-sensitive. Thus a ListDir on `test.log` will not find `test.LOG`.
USERNAME	Required if the FTP connection is not already cached. If connection caching is used, the USERNAME attribute is not required. User name to pass in the FTP operation.
PASSWORD	Required if the FTP connection is not already cached. If connection caching is used, the PASSWORD attribute is not required. Password to log the user.
NAME	Required for ACTION="ListDir". Specifies the query name to hold the directory listing.

CFFTP File and Directory Operation Attributes (Continued)	
Attribute	**Description**
SERVER	Required if the FTP connection is not already cached. If connection caching is used, the SERVER attribute is not required. The FTP server to connect to.
TIMEOUT	Optional. Value in seconds for the timeout of all operations, including individual data request operations. Defaults to 30 seconds.
PORT	Optional. The remote port to connect to. Defaults to 21 for FTP.
CONNECTION	Optional. The name of the FTP connection. Used to cache the current FTP connection or to reuse connection information from a previous connection of the same name. All calls to CFFTP with the same connection name will reuse the same FTP connection information.
ASCIIEXTENSIONLIST	Optional. A semicolon delimited list of file extensions that force ASCII transfer mode when TRANSFERMODE="Autodetect". Default extension list is: `txt;htm;html;cfm;cfml;shtm;shtml;css;asp;asa`
TRANSFERMODE	Optional. The FTP transfer mode you want to use. Valid entries are ASCII, Binary, or Autodetect. Defaults to Autodetect.
AGENTNAME	Optional. Application or entity conducting transfer.
FAILIFEXISTS	Optional. Yes or No. Defaults to Yes. Specifies whether a GetFile operation will fail if a local file of the same name already exists.
DIRECTORY	Required for ACTION=ChangeDir, CreateDir, ListDir, and ExistsDir. Specifies the directory on which to perform an operation.
LOCALFILE	Required for ACTION=GetFile, and PutFile. Specifies the name of the file on the local file system.
REMOTEFILE	Required for ACTION=GetFile, PutFile, and ExistsFile. Specifies the name of the file on the FTP server's file system.

CFFTP File and Directory Operation Attributes (Continued)	
Attribute	**Description**
ATTRIBUTES	Optional. Defaults to "Normal." A comma delimited list of file attributes. Specifies the file attributes for the local file in a GetFile. Can be any combination of the following: • ReadOnly • Hidden • System • Archive • Directory • Compressed • Temporary • Normal File attributes differ according to environment.
ITEM	Required for ACTION=Exists, and Remove. Specifies the object, file or directory, of these actions.
EXISTING	Required for ACTION=Rename. Specifies the current name of the file or directory on the remote server.
NEW	Required for ACTION=Rename. Specifies the new name of the file or directory on the remote server.
RETRYCOUNT	Optional. Number of retries until failure is reported. Default is one (1).
STOPONERROR	Optional. Yes or No. When Yes, halts all processing and displays an appropriate error. Default is No. When No, three variables are populated: • CFFTP.Succeeded — Yes or No. • CFFTP.ErrorCode — Error number (See STOPONERROR variables, below.) • CFFTP.ErrorText — Message text explaining error condition.

Connection Caching

Once you've established a connection with CFFTP, you can reuse the connection to perform additional FTP operations. To do this, you use the CONNECTION attribute when establishing a first connection to define and name an FTP connection object

that stores information about the connection. Any additional FTP operations that use the same CONNECTION name automatically make use of the information stored in the connection object. This facility helps save the time necessary to connect and logon to an FTP server and improves file transfer operation performance.

When you access an already active FTP connection, you don't need to re-specify the following connection attributes:

- USERNAME
- PASSWORD
- SERVER

In this case, developers have to make sure that when they use frames and multiple requests come in only one frame is going to use the connection object.

Caching connections across multiple pages

CFFTP caching is maintained only in the current page unless you explicitly assign a CFFTP 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 the most sense.

You cache a connection object for a session by assigning the connection name to a session variable:

Example: Caching a connection

```
<CFFTP ACTION=connect
    USERNAME="anonymous"
    PASSWORD="me@home.com"
    SERVER="ftp.eclipse.com"
    CONNECTION="Session.myconnection">
```

In this example, the connection cache remains available to other pages within the current session. Of course, you need to be sure that you've enabled session variables in your application first.

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

Connection caching actions and attributes

The following table shows which CFFTP attributes are required for CFFTP actions when employing connection caching. If connection caching is not used, the connection attributes USERNAME, PASSWORD, and SERVER must be specified.

CFFTP Required Attributes by Action			
Action	**Attributes**	**Action**	**Attributes**
Open	none	Rename	EXISTING NEW
Close	none	Remove	SERVER ITEM
ChangeDir	DIRECTORY	GetCurrentDir	none
CreateDir	DIRECTORY	GetCurrentURL	none
ListDir	NAME DIRECTORY	ExistsDir	DIRECTORY
GetFile	LOCALFILE REMOTEFILE	ExistsFile	REMOTEFILE
PutFile	LOCALFILE REMOTEFILE	Exists	ITEM

Example: An FTP session

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

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

<--- get current directory name --->
<CFFTP CONNECTION=FTP
    ACTION="GetCurrentDir"
    STOPONERROR="Yes">

<--- output directory name --->
<CFOUTPUT>
```

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

<--- get directory info --->
<CFFTP CONNECTION=FTP
    ACTION="listdir"
    DIRECTORY="/*."
    NAME="q"
    STOPONERROR="Yes">

<--- output dirlist results --->
<HR>
<P>FTP Directory Listing:</P>

<CFTABLE QUERY="q" HTMLTABLE>
    <CFCOL HEADER="<B>Name</B>" TEXT="#name#">
    <CFCOL HEADER="<B>Path</B>" TEXT="#path#">
    <CFCOL HEADER="<B>URL</B>" TEXT="#url#">
    <CFCOL HEADER="<B>Length</B>" TEXT="#length#">
    <CFCOL HEADER="<B>LastModified</B>"
    TEXT="Date(Format#lastmodified#)">
    <CFCOL HEADER="<B>IsDirectory</B>"
        TEXT="#isdirectory#">
</CFTABLE>
```

CFFTP Variables

Variables returned for CFFTP operations are as follows:

- CFFTPResult.ReturnValue
- Three variables populated when the value of the STOPONERROR attribute is "No."
- CFFTP query object when the value of the ACTION attribute is "ListDir."

Sections that follow describe each of these variable types.

CFFTPResult.ReturnValue variable

The value of the CFFTPResult.ReturnValue variable is determined by the result of the ACTION attribute used in CFFTP.

CFFTPResult.ReturnValue Variable	
CFFTP ACTION	**Value of CFFTPResult.ReturnValue**
GetCurrentDir	String value of the current directory
GetCurrentURL	String value of the current URL

CFFTPResult.ReturnValue Variable (Continued)	
CFFTP ACTION	**Value of CFFTPResult.ReturnValue**
`ExistsDir`	Yes or No
`ExistsFile`	Yes or No
`Exists`	Yes or No

STOPONERROR variables

The following variables are created when the STOPONERROR attribute is No:

- CFFTP.Succeeded — Yes or No
- CFFTP.ErrorCode — Error number (See CFFTP.ErrorCode values below)
- CFFTP.ErrorText — Message text explaining error condition

CFFTP.ErrorCode values

The following table lists the error codes that can be returned in the CFFTP.ErrorCode variable:

CFFTP.ErrorCode Values	
Error Code	**Description**
0	Operation succeeded
1	System error (OS or FTP protocol error)
2	An Internet session could not be established
3	FTP session could not be opened
4	File transfer mode not recognized
5	Search connection could not be established
6	Invoked operation valid only during a search
7	Invalid timeout value
8	Invalid port number
9	Not enough memory to allocate system resources
10	Cannot read contents of local file
11	Cannot write to local file

CFFTP.ErrorCode Values (Continued)	
Error Code	**Description**
12	Cannot open remote file for reading
13	Cannot read remote file
14	Cannot open local file for writing
15	Cannot write to remote file
16	Unknown error
17	reserved
18	File already exists
19	reserved
20	reserved
21	Invalid retry count specified

CFFTP query object properties

When you use CFFTP with the ListDir action, you must also specify a value for the NAME attribute. The value of the NAME attribute is used to hold the results of the ListDir action in a query object. The query object consists of columns you can reference in the form:

```
queryname.columname[row]
```

Where *queryname* is the name of the query as specified in the NAME attribute and *columnname* is one of the columns returned in the query object as shown in the following table. *Row* is the row number for each file/directory entry returned by the ListDir operation. A separate row is created for each entry.

CFFTP Query Object Columns	
Column	**Description**
Name	Filename of the current element
Path	File path (without drive designation) of the current element
URL	Complete URL for the current element (file or directory)
Length	Number indicating file size of the current element

CFFTP Query Object Columns (Continued)	
Column	**Description**
LastModified	Unformatted date/time value of the current element
Attributes	String indicating attributes of the current element
IsDirectory	Boolean value indicating whether object is a file or directory

The example at the beginning of this chapter includes a LISTDIR action and the output of the query.

CHAPTER 15

Accessing Remote Servers with HTTP

This section describes how ColdFusion wraps the complexity of Hypertext Transfer Protocol communications in a simplified tag syntax that allows you to easily extend your site's offerings across the Web.

Contents

Using CFHTTP to Interact with the Web

The CFHTTP tag is one of the more powerful tags in the CFML tag set. It can be used to:

- GET pages and query results from the Web for local rendering.
- POST data to a remote server, CFML template, or CGI application and process the returned data.

HTTP, the Hypertext Transfer Protocol, manages a variety of communications between Web clients and servers. The method by which a transaction is executed determines both what the server processes and what is delivered to the browser. In addition to HTML documents, HTTP supports content types such as binary files, graphics, audio, and video through the MIME (Multipurpose Internet Mail Extensions) standard.

The HTTP standard specifies that information about a client request and a server response, as well as the status of a transaction are generated in request and response headers as part of an HTTP session. The HTTP protocol has several communication methods used to exchange information between a client and a server. The CFHTTP tag implements the most commonly used methods, GET and POST. CFHTTP syntax is quite simple but offers a number of options for specifying output, resolving links, and building queries from delimited text files.

Allaire Alive

A video titled, "Creating Web Agents" is available at http://alive.allaire.com. It gives an overview of HTTP and covers the use of CFHTTP for creating automated processes such as:

- Search agents
- Transaction agents
- Messaging agents

The video is part of Allaire Alive, an educational service that offers Web videos on topics specific to ColdFusion development and application deployment as well as broader industry issues. The titles are available free for online viewing or download.

Using Secure Sockets Layer (SSL) with CFHTTP

There is a limitation on ColdFusion's ability to handle SSL transactions with CFHTTP on Windows NT. See Knowledge Base article #1096 at http://www.allaire.com/Support/KnowledgeBase/SearchForm.cfm for details.

CFHTTP Tag Syntax

The basic syntax for the CFHTTP tag is listed below:

```
<CFHTTP URL="hostname"
    USERNAME="username"
    PASSWORD="password"
    NAME="queryname"
    COLUMNS="query_columns"
    PATH="path"
    FILE="filename"
    METHOD="get_or_post"
    DELIMITER="character"

    TEXTQUALIFIER="character"
    RESOLVEURL="Yes/No"
    PROXYSERVER="hostname">
</CFHTTP>
```

Note A closing tag is only required for CFHTTP Post operations.

Resolving links in retrieved pages

To maintain relative links in pages you retrieve from a server via CFHTTP, set the RESOLVEURL attribute to Yes. The following HTML tags, which can contain links, are resolved when RESOLVEURL=Yes:

- IMG SRC
- A HREF
- FORM ACTION
- APPLET CODE
- SCRIPT SRC
- EMBED SRC
- EMBED PLUGINSPACE
- BODY BACKGROUND
- FRAME SRC
- BGSOUND SRC
- OBJECT DATA
- OBJECT CLASSID
- OBJECT CODEBASE
- OBJECT USEMAP

Note If a URL on a retrieved page points to a binary file, the page is not displayed. You can store the file locally by entering a path as the value of the PATH attribute.

Using the CFHTTP Get Method

Use Get to retrieve text and binary files from a specified server. The examples below illustrate a few common GET operations. The Get method is a one-way transaction in which CFHTTP retrieves an object. By comparison, the Post method is a two-way transaction in which CFHTTP passes variables to a ColdFusion page or CGI program which then returns data, usually processing what was received.

Example: Retrieving to a variable

The following example uses the Get method to perform a simple file request on the default page of the Yahoo Web site (as of this writing of course) and stores the contents of the file in a variable. Using CFOUTPUT, the page is rendered from the contents of the variable and displayed in the browser.

```
<CFHTTP METHOD="Get"
    URL="http://www.yahoo.com/index.htm"
    RESOLVEURL="Yes">

<CFOUTPUT>
    #CFHTTP.FileContent# <BR>
</CFOUTPUT>
```

When FILE and PATH attributes are omitted, ColdFusion stores the contents of the file `index.htm` in the CFHTTP.FileContent variable. Note that, without the RESOLVEURL attribute set to "Yes," relative links in the downloaded page would be broken. When set to Yes, RESOLVEURL turns relative links into absolute links that resolve correctly.

Example: Retrieving to a file

The following example also uses the Get method to perform a simple file request, as above, but the addition of the PATH and FILE attributes results in the retrieved file being saved to the server as a variable.

```
<CFHTTP
    METHOD = "get"
    URL="http://www.yahoo.com/index.htm"
    PATH="c:\temp"
    FILE="yahooindex.htm">
```

Note that when the PATH and FILE attributes are used, the RESOLVEURL attribute is ignored, even if present. The contents of the retrieved file can be referenced in the CFHTTP.FileContents variable.

Example: Retrieving a binary file

Like the previous example, this example retrieves a file from a server and saves it to the location specified in the PATH and FILE attributes. The only difference is the MIME type of the retrieved file.

```
<CFHTTP
    METHOD="Get"
    URL="http://maximus/downloads/quakestuff/q2_test.zip"
    PATH="c:\quake2\install"
    FILE="quake2beta.zip">

<CFOUTPUT>
    #CFHTTP.MimeType#
</CFOUTPUT>
```

The CFOUTPUT block at the end displays the MIME type of the downloaded file, which in this case, is interpreted as `application/zip`.

Creating a Query from a Text File

Using the CFHTTP Get operation, you can create a query object from a delimited text file. This is a powerful means for processing and handling generated text files. Once the query object is created, it is very simple to reference columns in the query and perform other ColdFusion magic on the data.

Text files are processed in the following manner:

- You specify a delimiter with the DELIMITER attribute. If data in a field includes the delimiter character, it must be quoted or qualified with some other character, which you specify with the TEXTQUALIFIER attribute.
- By default, the first row of the text file is interpreted as column heading text. You can replace this heading text by specifying alternate heading text in the COLUMNS attribute, making sure that you supply an alternate for every column of data in the text file.
- 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_".

Example: Creating a query from a text file

In this example, a query object is created from a comma-delimited text file. The CFOUTPUT tag is used to output specific columns in the query. The text file consists of six columns, separated by commas. The first row of the file looks like this:

```
OrderID,OrderNum,OrderDate,ShipDate,ShipName,ShipAddress
```

This example accepts the first row of the text file as the column names:

```
<CFHTTP METHOD="Get"
    URL="http://127.0.0.1/orders/june/orders.txt"
    NAME="juneorders"
    DELIMITER=","
    TEXTQUALIFIER="""">
```

```
<CFOUTPUT QUERY="juneorders">
    OrderID: #OrderID#<BR>
    Order Number: #OrderNum#<BR>
    Order Date: #OrderDate#<BR>
</CFOUTPUT>
```

You can substitute different column names by using the COLUMNS attribute:

```
<CFHTTP METHOD="Get"
    URL="http://127.0.0.1/orders/june/orders.txt"
    NAME="juneorders"

    COLUMNS="ID, Number,Date"
    DELIMITER=","
    TEXTQUALIFIER="""">

<CFOUTPUT QUERY="juneorders">
    Order ID: #ID#<BR>
    Order Number: #Number#<BR>
    Order Date: #Date#<BR>
</CFOUTPUT>
```

Using the CFHTTP Post Method

Use the Post method to send cookie, form, CGI, URL, and file variables to a specified ColdFusion page or CGI program. For Post operations, the CFHTTPPARAM tag must be used for each variable you want to post. Unlike the Get method, Post passes data to a specified ColdFusion page or to some 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 program to which form data will be passed. Using the Post method in CFHTTP is exactly the same.

Example: Pass variables to a ColdFusion page

The following example passes the five supported variable types to the page specified in the URL attribute. The page that receives this data is also shown. It returns the value of the variables which appears in the client's browser. This example uses the CFFILE tag in the page that receives the Posted variables to upload the contents of the file variable to `c:\temp\junk.`

The CFOUTPUT section in `posttest.cfm` references the `CFHTTP.FileContent` variable, which is used to display the output from the `server.cfm` file. If the `CFHTTP.FileContents` variable were left out, the browser output would be limited to the contents of the `posttest.cfm` file.

Sample file: `posttest.cfm`

```
<CFHTTP METHOD="Post"
    URL="http://housebeat/cfdocs/server.cfm"
    USERNAME="user1"
    PASSWORD="user1pwd">

    <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="wbfreuh@allaire.com"
        NAME="emailaddress">
    <CFHTTPPARAM TYPE="File"
        NAME="myfile"
        FILE="c:\temp\cyberlogo.gif">
</CFHTTP>

<CFOUTPUT>
    #CFHTTP.filecontent#
    #CFHTTP.mimetype#
</CFOUTPUT>
```

Sample file: `server.cfm`

```
You have POSTed to me.<BR>
<CFFILE DESTINATION="c:\temp\junk"
    NAMECONFLICT="Overwrite"
    FILEFIELD="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.myformfield#. <BR>
</CFOUTPUT>
```

This example uses the CFFILE tag to upload the contents of the file variable to `c:\temp\junk`.

Example: Returns results of CGI program

The following example runs a CGI program, `search.exe`, that searches the site and returns the hits on the value specified in VALUE.

```
<CFHTTP METHOD="Post"
    URL="http://www.thatsite.com/search.exe"
    RESOLVEURL="Yes">

    <CFHTTPPARAM TYPE="Formfield"
        NAME="search"
        VALUE="hello">

</CFHTTP>

<CFOUTPUT>
    #CFHTTP.MimeType#<BR>
    Length: #len(cfhttp.filecontent)# <BR>
    Content: #htmlcodeformat(cfhttp.filecontent)#<BR>
</CFOUTPUT>
```

CHAPTER 16

Sending and Receiving Email

You can add interactive email features to your ColdFusion applications, providing complete two-way interface to mail servers via the CFMAIL tag and the CFPOP tag. The boom in Internet mail services makes ColdFusion's enhanced email capability a vital link to your users.

Contents

Using ColdFusion with Mail Servers

Adding email to your ColdFusion applications lets you respond automatically to user requests. You can use email in your ColdFusion applications in many different ways. These are just a few examples:

- Trigger email messages based on users' requests or orders.
- Allow users to request and receive additional information or documents through email.
- 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 email into your applications. For sending email, you generally use the Simple Mail Transfer Protocol (SMTP). For receiving mail, you use the Post Office Protocol (POP) to retrieve email from the mail server. To use email messaging in your ColdFusion applications you must have access to an SMTP server, as well as a valid POP account and password.

In your ColdFusion application pages, you use the CFMAIL and CFPOP tags to send and receive mail respectively. The following sections describe ColdFusion email features and offer examples of these tags.

Sending Email Messages (SMTP)

Before you set up ColdFusion to send email messages, you must have access to an SMTP email server. Also, before you run application pages that refer to the email server, you must configure the ColdFusion Administrator to use the SMTP server.

To configure ColdFusion for email:

1. Open the Mail page in the ColdFusion Administrator.
2. In the Mail Server box, enter the address of the SMTP mail server you want ColdFusion to use.
3. Leave the Server Port and Connection Timeout settings at their default values.
4. Click Apply to save the settings.
5. To verify server settings, click the Verify button to make sure ColdFusion can access your mail server.

See the *Administering ColdFusion Server* book for more information on the Administrator's mail settings.

Sending SMTP mail with CFMAIL

The CFMAIL tag provides support for sending SMTP (Simple Mail Transfer Protocol) email from within ColdFusion applications. The CFMAIL tag is similar to the CFOUTPUT tag, except that CFMAIL outputs the generated text as SMTP mail

messages rather than to the display screen. All attributes and commands you use with CFOUTPUT you can also use with CFMAIL.

Here's an example of a basic use of the CFMAIL tag, using form variables to resolve the From, To, Subject, and message body components of the mail message.

```
<CFMAIL QUERY="GetList"
    TO="#Email#"
    FROM="Sales DepartmentList Manager"
    SUBJECT="#Form.Subject#">

    #FORM.Body#

</CFMAIL>
```

More detailed examples follow in subsequent sections. Also, see the *CFML Language Reference* for a full catalog of the CFMAIL tag's attributes and values.

SMTP Examples with CFMAIL

An application page with the CFMAIL tag dynamically generates email messages based on the tag's settings. Some of the things you can accomplish with CFMAIL are:

- Send a mail message whose recipient and contents are determined by data the user enters in an HTML form.
- Use a query to send a mail message to a database-driven list of recipients.
- Use 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.
- Send a MIME file attachment along with a mail message.

The CFMAIL tag behaves like the CFOUTPUT tag. It shares some attributes used by CFOUTPUT - QUERY, GROUP, STARTROW, and MAXROWS - and it employs several mail-specific attributes to handle mail header information and MIME and HTML files.

See the *CFML Language Reference* for a more information.

Sending form-based email

In the example below, the contents of a customer inquiry form submittal are forwarded to the marketing department. Note that the same application page could also insert the customer inquiry into the database.

```
<CFMAIL FROM="#Form.EMailAddress#"
    TO="marketing@allaire.com"
    SUBJECT="Customer Inquiry">

A customer inquiry was posted to our Web site:

Name: #Form.FirstName# #Form.LastName#
```

```
Subject: #Form.Subject#

#Form.InquiryText#

</CFMAIL>
```

Sending query-based email

In the example below, a query ("ProductRequests") is run to retrieve a list of the customers who have inquired about a product over the last seven days. This list is then sent, with an appropriate header and footer, to the marketing department:

```
<CFMAIL QUERY="ProductRequests"
    FROM="webmaster@allaire.com"
    TO="marketing@allaire.com"
    SUBJECT="ColdFusion status report">

Here is a list of people who have inquired about
Allaire ColdFusion over the last seven days:

<CFOUTPUT>
#ProductRequests.FirstName# #ProductRequests.LastName#
(#ProductRequests.Company#) - #ProductRequests.EMailAddress#
</CFOUTPUT>

Regards,
The WebMaster
webmaster@allaire.com

</CFMAIL>
```

Note the use of the nested CFOUTPUT tag to present a dynamic list embedded within a normal CFMAIL message. The text within the CFOUTPUT is repeated for each row in the "ProductRequests" query, while the text above and below it serve as the header and footer (respectively) for the mail message.

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

Sending email to multiple recipients

In the following example, a query ("CFBetaTesters") is run to retrieve a list of people who are beta testing ColdFusion. This query is then used to send a notification to each of these testers that a new version of the beta release is available:

```
<CFMAIL QUERY="CFBetaTesters"
    FROM="beta@allaire.com"
    TO="#TesterEMail#"
    SUBJECT="ColdFusion Beta Four Available">

To all ColdFusion beta testers:

ColdFusion Beta Four is now available for downloading
from the Allaire site.The URL for the download is:

http://beta.allaire.com

Regards,
ColdFusion Technical Support
beta@allaire.com

</CFMAIL>
```

Note that in this example, the contents of the CFMAIL tag are not dynamic, that is, they don't use any # delimited dynamic parameters. What is dynamic is the list of email addresses to which the message is sent. Note the use of the "TesterEMail" column from the "CFBetaTesters" query in the TO attribute.

Customizing Email for Multiple Recipients

In the following example, a query ("GetCustomers") is run to retrieve the contact information for a list of customers. This query is then used to send an email to each customer asking them to verify that their contact information is still valid:

```
<CFMAIL QUERY="GetCustomers"
    FROM="service@allaire.com"
    TO="#EMail#"
    SUBJECT="Contact Info Verification">

Dear #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: #Company#
Contact: #FirstName# #LastName#

Address:
    #Address1#
    #Address2#
    #City#, #State# #Zip#

Phone: #Phone#
Fax: #Fax#
Home Page: #HomePageURL#
```

```
Please let us know if any of the above
information has changed, or if we need to
get in touch with someone else in your
organization regarding this request.

Thanks,
Allaire Customer Service
service@allaire.com

</CFMAIL>
```

Note that in the TO attribute of CFMAIL, the #Email# query column causes one message to be sent to the address listed in each row of the query. Also note the use of the other query columns (FirstName, LastName, etc.) within the CFMAIL section to customize the contents of the message for each recipient.

Attaching a MIME file

In the following example, a MIME-encoded file is sent along with an email message:

```
<CFMAIL FROM="info@allaire.com"
    TO="jdoe@supercomputer.com"
    SUBJECT="File you requested"
    MIMEATTACH="c:\cfmanual.doc">

Dear Joe,

Here is a copy of the file you requested.

Regards,
The Allaire Team

</CFMAIL>
```

For more information about the attributes and values for the CFMAIL tag, see the *CFML Language Reference*.

Advanced Sending Options

The ColdFusion implementation of SMTP mail uses a spooled architecture. This means that when a CFMAIL tag is processed in an application page, the messages generated are not sent immediately. Instead, they are spooled to disk and processed in the background. This architecture has two distinct advantages:

1. End users of your application are not required to wait for SMTP processing to complete before a page returns to them. This is especially useful when a user action causes more than a handful of messages to be sent.
2. Messages sent using CFMAIL are delivered reliably, even in the presence of unanticipated events like power outages or server crashes.

In most cases, spooled messages are processed immediately by ColdFusion and delivery occurs almost instantly. If, however, ColdFusion is either extremely busy or has a large existing queue of messages, delivery could occur some time after the request is submitted.

Sending mail as HTML

Most newer Internet mail applications are capable of reading and interpreting HTML code in a mail message. The CFMAIL tag allows you to specify the message type as HTML. The TYPE attribute, which only accepts HTML as an argument, informs the receiving email client that the message has embedded HTML tags that need to be processed. This feature is only useful when sending messages to mail clients that understand HTML.

Overriding default SMTP server settings

You can use the following optional CFMAIL attributes to override the default SMTP Server settings set with the ColdFusion Administrator:

CFMAIL Tag	
Optional Attributes	**Description**
SERVER	The address of the SMTP server to use for sending messages.
PORT	The TCP/IP port on which the SMTP server listens for requests. This is almost always 25.
TIMEOUT	The time, in seconds, before timing out the connection to the SMTP server.

Error logging and undelivered messages

All errors that occur during the processing of SMTP messages are logged to the file `errors.log` in the ColdFusion log directory. Error log entries contain the date and time of the error as well as diagnostic information on why the error occurred.

All messages not delivered because of an error are written to the `\cfusion\mail\undelivr` directory. The error log entry corresponding to the undelivered message contains the name of the file written to the `undelivr` directory.

See *Administering ColdFusion Server* for more information about the mail logging settings in the ColdFusion Administrator.

Receiving Email Messages (CFPOP)

CFPOP, the Post Office Protocol tag, expands the ColdFusion developer's ability to add Internet mail client features and email consolidation to applications. While a conventional mail client provides an adequate interface for personal mail, there are many cases where an alternative interface to some mailboxes is desirable. CFPOP is a tool to develop targeted mail clients to suit the specific needs of a wide range of applications.

Use CFPOP in applications when you want to receive email. Here are two instances where implementing POP mail makes sense:

- If your site has generic mailboxes that are read by more than one person (*sales@yourcompany.com*), it may be more efficient to construct a ColdFusion mail front-end to supplement individual user mail clients.
- In many applications, the processing of mail can be automated when the mail is formatted to serve a particular purpose. For example, when subscribing to a list server.

Use the CFPOP tool in the CFML Advanced toolbar in ColdFusion Studio to add CFPOP tags to your pages.

See the *CFML Language Reference*for more information on CFPOP syntax and variables.

CFPOP Example

This example shows the basic syntax of CFPOP:

```
<CFPOP SERVER="my.mailserver.com"
    USERNAME=#username#
    PASSWORD=#pwd#
    ACTION="GetHeaderOnly"
    NAME="getmsghdrs">

<BODY>
<CFOUTPUT>
You have #getmsghdrs.RecordCount# messages to read.
</CFOUTPUT>

<CFOUTPUT QUERY="getmsghdrs">
    <P><B>Date: </B>#getmsghdrs.date#</P>
    <P><B>From: </B>#getmsghdrs.from#</P>
</CFOUTPUT>
</BODY>
```

Using CFPOP

To implement the CFPOP tag in your ColdFusion application:

1. Choose which mail boxes you want to access within your ColdFusion application.

2. Determine what mail message components you need to process: message header, message body, attachments, etc.
3. Decide if you need to store the retrieved messages in a database.
4. Decide if you need to delete messages from the POP server once you've retrieved them.
5. Incorporate the CFPOP tag in your application and create a user interface for accessing a given mailbox.
6. Build an application page to handle the output. Retrieved messages can include ASCII characters that do not display properly in the browser.

   Use the CFOUTPUT tag with the HTMLCodeFormat and HTMLEditFormat functions to control output to the browser. Note the use of these functions in the examples.

CFPOP query variables

Two variables are returned for each CFPOP query that provide record number information:

- RecordCount: The total number of records returned by the query.
- CurrentRow: The current row of the query being processed by CFOUTPUT in a query-driven loop.

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>
```

Handling POP Mail

You use the ACTION attributes of the CFPOP tag to describe the data set to retrieve. You can get all data (GetAll), get header data only (GetHeaderOnly), or delete mail (DELETE).

This section gives an example of each usage:

- Retrieving only message headers
- Retrieving a message body
- Retrieving attachments
- Deleting messages

Note that for any of the examples below to work, you'll need to reference an existing POP mail server with a valid user and password. Save the code example to a directory using the indicated filenames. Then open each CFM file to run the example.

Returning only message headers

When you use the CFPOP tag with ACTION="GetHeaderOnly", ColdFusion returns only the mail message header information to the query specified in the NAME attribute. The following columns are returned:

- DATE
- FROM
- MESSAGENUMBER
- REPLYTO
- SUBJECT
- CC
- TO

Header information

Mail message header information returned by CFPOP may be enclosed in HTML coding. You can use the ColdFusion function HTMLCodeFormat to replace HTML tags with escaped characters, such as `>` for > and `<` for <.

In addition, the date returned by CFPOP can be processed with ParseDateTime, which accepts an argument for converting POP date/time objects into GMT (Greenwich Mean Time).

See the *CFML Language Reference*for information on these ColdFusion functions.

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>
```

Example of retrieving message headers

Users can retrieve and optionally delete mail messages from a POP mail server. This first example, `hdronly.cfm`, illustrates how to retrieve message header information. You can use the code as is after you've supplied your POP mail server name, user name and password.

This code retrieves the message headers and stores them in a CFPOP result set called *Sample*.

Sample file: `hdronly.cfm`

```
<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=#username#
    PASSWORD=#password#
    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>
</CFOUTPUT>

</BODY>
</HTML>
```

Returning an entire message

When you use the CFPOP tag with ACTION="GetAll", ColdFusion returns the same columns returned with GETHEADERONLY, as well as two additional columns, BODY and HEADER.

Example

In this sample file, `hdrbody.cfm`, the ACTION attribute is set to GetAll, so it retrieves the message body as well as its header.

Sample file: `hdrbody.cfm`

```
<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=#username#
    PASSWORD=#password#
```

```
    ACTION="GetAll"
    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>
    Body: #HTMLCodeFormat(Sample.BODY)# <BR>
    Header: #HTMLCodeFormat(Sample.HEADER)# <BR>
</CFOUTPUT>

</BODY>
</HTML>
```

Returning attachments with messages

When you use the CFPOP tag with ACTION="GetAll", and add the ATTACHMENTPATH attribute, ColdFusion returns two additional columns:

- ATTACHMENTS contains a tab-separated list of all source attachment names.
- ATTACHMENTFILES contains a tab-separated list of the actual temporary filenames written to the server. Use the CFFILE tag to delete the temporary files.

Not all messages have attachments. If a message has no attachments, both ATTACHMENTS and ATTACHMENTFILES will be equal to an empty string.

Managing attachment filenames

To avoid any problems with duplicate filenames, ColdFusion creates temporary files for all attachments. If you retrieve files with attachments, you can use the ATTACHMENTPATH attribute to choose where the attachment files go. It is the developer's responsibility to manage these files so they can be accessed unambiguously.

Also note that it is the developer's responsibility to clean up any temporary files written as a result of executing a CFPOP tag.

Example

This retrieval example, `attach.cfm`, retrieves all parts of the message including attachments.

Sample File: `attach.cfm`

```
<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=#username#
    PASSWORD=#password#
    ACTION="GetAll"
    ATTACHMENTPATH="c:\attachdir"
    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>
    Body: #HTMLCodeFormat(Sample.BODY)# <BR>
    Header: #HTMLCodeFormat(Sample.HEADER)# <BR>
</CFOUTPUT>

</BODY>
</HTML>
```

Deleting messages

By default, retrieved messages are not deleted from the POP mail server. If you want to delete retrieved messages, you must set the ACTION attribute to Delete.

The MESSAGENUMBER attribute returned by all CFPOP retrievals contains the message number you need to pass back to the POP mail server to have the corresponding message deleted. A few notes:

- Once a message is deleted, it's gone for good.
- Message numbers are reassigned at the end of every POP mail server communication that contains a delete action. For example, if four messages are retrieved from a POP mail server, the message numbers returned will be 1,2,3,4. If two messages are then deleted within a single CFPOP tag, messages 3 and 4 will be assigned message numbers 1 and 2, respectively.

Example

Sample file: `msgdel.cfm`

```
<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>
```

For more information

For more information on using the CFMAIL and CFPOP tags, see the *CFML Language Reference*.

CHAPTER 17

Indexing and Searching Data

You can provide a full-text search capability for documents and data sources on a ColdFusion site by enabling the Verity search engine.

Contents

Searching a ColdFusion Web Site

Verity, Inc. SEARCH'97 engine is bundled with ColdFusion to provide full-text indexing and searching functionality. This feature allows you to enhance your ColdFusion applications with a powerful search engine, giving your users intelligent access to content.

The ColdFusion online documentation employs Verity for searches against the installed document set. The first time you click the Search button on the ColdFusion Documentation home page, you will be prompted to index the set. When that is completed, you can run searches.

Here are some of the possible uses for Verity in ColdFusion:

- Index your Web site and provide a generalized search mechanism, such as a form interface, for executing searches.
- Index specific directories containing documents for subject-based searching.
- Index CFQUERY result sets, giving your end users the ability to search against the data. Since collections are made up of data optimized for retrieval, this method is much faster than performing multiple database queries to return the same data.
- Index CFLDAP and CFPOP query results.
- Manage and search collections generated outside of ColdFusion using native Verity tools. This additional capability requires only that the full path to the collection be specified in the index command.
- Index email generated by ColdFusion application pages and create a searching mechanism for the indexed messages.
- Build collections of inventory data and make those collections available for searching from your ColdFusion application pages.
- Support international users in a range of languages from both the CFINDEX and CFSEARCH tags.

Advantages of using Verity

Verity can index the output from queries so that you or an end user can search against the result sets. This has a clear advantage in speed of execution because pointers to the result sets are stored in a Verity index that is optimized for searching. You can reduce the programming overhead of query constructs by allowing users to construct their own queries and execute them directly. You need only be concerned with presenting the output to the client browser.

Verity can index database text fields, such as notes and product descriptions, that cannot be effectively indexed by native database tools.

When indexing collections containing documents in Adobe Acrobat (PDF) format, Verity scans for the document title (if one has been entered in Acrobat Exchange). The document title displays in the search results list.

Indexing Web pages returns the URL for each document. This is a valuable document management feature.

Online Verity training

A video titled "Creating Search Engines with Verity" is available at http://alive.allaire.com. The video gives an overview of the Verity implementation in ColdFusion and illustrates the development process with sample code.

The video is part of Allaire Alive, an educational service that offers Web videos on topics specific to ColdFusion development and application deployment as well as broader industry issues. The titles are available free for online viewing or download.

Verity collections

The Verity engine performs searches against *collections*. A collection is a special database created by Verity that contains pointers to the indexed data that you specify for that collection. ColdFusion's Verity implementation supports collections of three basic data types:

- Text files such as HTML pages and CFML templates.
- Binary documents (see the Supported Document Types list below).
- Result sets returned from CFQUERY, CFLDAP, and CFPOP queries.

You can build a collection from individual documents or an entire directory tree. Collections can be stored anywhere, so you have a lot of flexibility in accessing indexed data. This adds enormous value to any content-rich Web site.

The first step in implementing Verity is to create a collection. This can be done either through the ColdFusion Administrator or programmatically. The next step is to populate the collection, that is, to select the data and generate the index. You now have a searchable data source. Designing a search interface and a results page complete the process.

Following is a brief description of the Verity tags.

CFCOLLECTION

Provides a programming interface for the commands available on the ColdFusion Administrator Verity page.

CFINDEX

Used primarily to populate collections and to update the index.

CFSEARCH

Used to define the search. The tag's attributes give you precise control of search results.

Supported File Types

The ColdFusion Verity implementation supports a wide array of document types. This means you can index Web pages, ColdFusion applications, and many binary document types and produce search results that include summaries of these documents.

The following table lists the supported document types.

Supported Document Types	
Documents	**Versions**
Text files	
HTML, CFML, DBM, SGML, XML,	N/A
ANSI, ASCII, Plain Text	N/A
Word processors	
Adobe Acrobat (PDF)	All
Adobe FrameMaker (MIF)	All
Aplix Words	4.2
Corel WordPerfect for Windows	5.x 6, 7, 8
Corel WordPerfect for Macintosh	2, 3
Lotus AMI Pro	2, 3
Lotus AMI Pro Write Plus	all
Lotus Word Pro	96, 97
Microsoft Office	95, 97
MS Rich Text Format (RTF)	1.x, 2.0
MS Word for Windows	2, 6, 95, 97
MS Word for DOS	4, 5, 6
MS Word for Macintosh	4.0, 5.0, 6.0
MS Notepad, WordPad	all
MS Write, MS Works	all
XYWrite	4.12
Spreadsheets	
Corel QuattroPro	7, 8
Lotus 1-2-3 for DOS/Windows	2.0, 3.0, 4.0, 5.0, '96, '97
Lotus 1-2-3 for OS/2	2
MS Excel	3, 4, 5, '95, '97
MS Works	all

Supported Document Types (Continued)	
Documents	**Versions**
Presentation	
Corel Presentations	7.0, 8.0
Lotus Freelance	96, 97
MS PowerPoint	4.0, 95, 97

Support for International Languages

The ColdFusion International Language Search Pack can be purchased and installed to index data in any the following languages:

- Danish
- Dutch
- Finnish
- French
- German
- Italian
- Norwegian
- Portuguese
- Spanish
- Swedish

The default language for Verity collections is English. To index data in one of the supported languages, you must select the language from the drop-down list when you create a collection on the ColdFusion Administrator Verity page. You must then enter the selected language as a value of the LANGUAGE attribute in both the CFINDEX and CFSEARCH tags used against that collection.

To order the Language Pack, contact Allaire Customer Service or visit our online store at http://www.allaire.com/store. The default installation directory for the dictionaries is in \cfusion\verity\common.

Creating a Collection

ColdFusion provides two interfaces for creating a Verity collection. You can make selections on the ColdFusion Administrator Verity page or code the CFCOLLECTION tag.

Running the ColdFusion Administrator

Open the ColdFusion Administrator Verity page. If you checked the option to install the ColdFusion Documentation, the documentation collection is listed by default. The Verity engine is used to search our online documents.

To create a new collection:

1. In the Add a Collection section, type in a name for the collection.
2. Type in a path for the location of the new collection. By default, new collections are added to `\Cfusion\Verity\Collections\`.
3. If you have an International Language Search Pack installed, you can select a language for the collection from the drop-down list.
4. Click Create a new collection. When the collection is created, the name and full path of the new collection appear in the Verity Collections list at the top of the page.

You can easily enable access to a collection on the network by creating a local reference (an alias) for that collection. It only needs to be a valid Verity collection; it doesn't matter whether it was created within ColdFusion or another tool.

To add an existing collection:

1. In the Add a Collection section, type in the collection alias.
2. Enter the full path to the collection.
3. Select Language if needed.
4. Click Map an existing collection.
5. Click Apply.

If the collection is subsequently moved, the alias path must be updated. The Delete command, when used on a mapped collection, only deletes the alias.

Coding the CFCOLLECTION tag

Creating and maintaining collections from a CFML application eliminates the need to access the ColdFusion Administrator. This can be an advantage when you need to schedule these tasks or to allow users to perform them without exposing the Administrator.

With the introduction of CFCOLLECTION, Verity tasks can now be grouped in a more logical way. CFCOLLECTION can be used for collection creation and maintenance and CFINDEX can be used to populate and update collections.

The valid values for the ACTION attribute are:

- **Create** — Creates a new collection using the specified path and optionally specified language.
- **Repair** — Attempts to fix corrupted data in the specified collection.

- **Delete** — Deletes the specified collection.
- **Optimize** — Reorganizes the collection data for greater efficiency, similar to a DBMS optimize operation.
- **Map** — Assigns an alias to an existing collection.

The CREATE and MAP actions require a collection name and path. The MAP action additionally requires an alias name. The LANGUAGE attribute is required for data in languages other than English. Only the ACTION and COLLECTION attributes are required for maintenance tasks.

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

Example: Collection action page

This example processes input from a form. Notice the use the new CFSWITCH tag to control page flow.

```
<HTML>
<HEAD>
    <TITLE>CFCOLLECTION</TITLE>
</HEAD>

<BODY>
<H1>CFCOLLECTION</H1>

<CFOUTPUT>

    <CFSWITCH EXPRESSION=#FORM.CollectionAction#>

        <CFCASE VALUE="Create">

            <CFCOLLECTION ACTION="Create"
            COLLECTION="#FORM.CollectionName#"
            PATH="C:\CFUSION\Verity\Collections\">
        </CFCASE>

        <CFCASE VALUE="Repair">

            <CFCOLLECTION ACTION="REPAIR"
            COLLECTION="#FORM.CollectionName#">
            <P>Collection repaired.
        </CFCASE>

        <CFCASE VALUE="Optimize">

            <CFCOLLECTION ACTION="OPTIMIZE"
            COLLECTION="#FORM.CollectionName#">
            <P>Collection optimized.
        </CFCASE>
```

```
        <CFCASE VALUE="Delete">

            <CFCOLLECTION ACTION="DELETE"
            COLLECTION="#FORM.CollectionName#">
            <P>Collection deleted.
        </CFCASE>

    </CFSWITCH>

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

The CFCOLLECTION tag operates at the collection level. To add content to a collection, use the CFINDEX tag.

Indexing a Collection

At this point, the new collection is just an empty shell. In fact, if you hold it up to your ear you can hear the ocean. You can use the CF Administrator or CFINDEX to populate the collection with indexed data.

Note You can index and search against Verity collections created outside of ColdFusion by using the EXTERNAL attribute of CFINDEX and CFSEARCH.

Selecting an indexing method

Use the following guidelines to determine which method to use.

Using the CF Administrator or CFINDEX	
Use the Administrator if	**Use the CFINDEX tag if**
You want to index document files.	You want to index ColdFusion query results.
The collection won't be updated very frequently.	You need to dynamically populate or update a collection from a ColdFusion application page.
You want to generate the collection without writing any CFML code.	Your collection needs to be updated frequently.
You want to generate a one-time collection.	Your collection needs to be updated by other people.

The following sections give instructions for each of the indexing methods.

Populating a Collection from Document Files

This section covers the two methods for indexing files.

Indexing files with the ColdFusion Administrator

To index a collection from the Verity page:

1. Select a collection name in the Verity Collections box.
2. Click Index to open the index page. The selected collection name appears at the top of the page.
3. Enter a single file type or multiple file types separated by commas.
4. Type in the directory path for the collection or click Browse Server and navigate to the directory in which to begin the index.
5. Check the Recursively index subdirectories box if you want to extend the indexing operation to all directories below the selected path.
6. Optionally, you can enter a Return URL to prepend to all indexed files. This allows you to easily create a link to any of the files in the index. A typical entry might be something like `http://localhost/wwwroot/`.
7. If the International Language Search Pack is installed, you can select one of the supported languages.
8. Click Update to begin the indexing process. The time required to generate the index depends on the number and size of the selected files in the path.

As you can see, this interface allows you to easily build a very specific index based on the file extension and path information you enter. In most cases, your server file structures need not be changed to accommodate the generation of indices.

In your ColdFusion application, you can populate and search multiple collections, each of which can be designed to focus on a specific group of documents or queries, according to subject, document type, location, or any other logical grouping. Searches can be performed against multiple collections, giving you lots of flexibility in designing your search interface.

Indexing files with CFINDEX

To programmatically index files, set the index parameters in CFSET tags, then specify those values in CFINDEX attributes. To illustrate these steps, we'll use a section of an indexing template generated by the Verity Wizard in ColdFusion Studio. To run the wizard, click File > New and select the Verity Wizard from the CFML tab of the New Document dialog.

This collection is a set of draft documents and supporting files used during the review process of the ColdFusion 4.0 documentation.

```
<CFSET IndexCollection = "Review Docs">
<CFSET IndexDirectory = "C:\Projects\CF40\Doc Source\">
<CFSET IndexRecurse = "YES">
<CFSET IndexExtensions = ".htm, .doc, .xls">
<CFSET IndexLanguage = "English">
```

The collection parameters listed here mirror those on the Administrator Verity Index page. The extensions list for this index includes HTML files, Microsoft Word documents, and Excel worksheets. To revert to the default extensions, simply enter double quotes with no space between. Other wildcards, such as *.* have no effect.

The indexing attributes and values are then entered.

```
<CFINDEX COLLECTION="#IndexCollection#"
    ACTION="REFRESH"
    TYPE="PATH"
    KEY="#IndexDirectory#\"
    EXTENSIONS="#IndexExtensions#"
    RECURSE"#IndexRecurse#"
    LANGUAGE="#IndexLanguage#">
```

Below this you can enter other CFML code as needed and HTML page elements for the search interface.

Type attribute options

Generally, a server path is entered as the value for the TYPE attribute, but you can use the TYPE="FILE" option under special circumstances, such as indexing a database table containing a list of file names. For more information on this topic, see the Allaire Knowledge Base article, "Using Indirection with CFINDEX TYPE=FILE" (ID# 1083) on our Web site.

Populating a Collection from a Query

Indexing the result set from a ColdFusion query involves an extra step not required when indexing documents. You need to code the query and output parameters, then point the CFINDEX tag at the result set from a CFQUERY, CFLDAP, or CFPOP query.

To index a ColdFusion query:

1. Create the collection on the ColdFusion Administrator Verity page.
2. Execute a query and output the data.
3. Populate the collection using the CFINDEX tag.

Indexing database query results

To populate a collection from a CFQUERY you specify a KEY, which corresponds to the primary key of the data source, and the BODY, the column in which you want to

conduct searches. The following extract shows only the CFQUERY and CFINDEX parts of the process.

```
<!--- Select the entire table --->
<CFQUERY NAME="Messages"
    DATASOURCE="CF 4.0 Examples">
    SELECT *
        FROM Messages
</CFQUERY>

<!--- Output the result set --->
<CFOUTPUT QUERY="Messages">
    #Message_ID#, #Subject#, #Title#, #MessageText#

</CFOUTPUT>

<!--- Index the result set --->
<CFINDEX COLLECTION="DBINDEX"
    ACTION="UPDATE"
    TYPE="CUSTOM"
    BODY="MessageText"
    KEY="Message_ID"
    TITLE="Subject"
    QUERY="Messages">
```

This CFINDEX statement specifies the MessageText column as the core of the collection and names the table's primary key, the Message_ID column, as the KEY value. Note that the TITLE attribute names the Subject column. The TITLE attribute can be used to designate an output parameter.

Indexing multiple columns

To index more than one column in a collection, enter a comma-separated list of column names for values of the BODY attribute, such as:

```
BODY=FirstName,LastName,Company
```

Custom fields

The CFINDEX tag supports two custom attributes, CUSTOM1 and CUSTOM2, that you can use to store data during an indexing operation. The query columns you specify as values for these attributes are returned in a CFSEARCH of the collection. For more information on this topic, see the Allaire Knowledge Base article, "Custom1, Custom2 and Summary Fields" (ID# 1081) on our Web site.

Advantages of indexing a data source

The main advantage of performing searches against a Verity collection over using CFQUERY alone is that the database is indexed in a form that provides faster access. Use this technique instead of CFQUERY in the following cases:

- You want to index textual data. Verity collections containing textual data can be searched much more efficiently with CFINDEX than searching a database with CFQUERY.
- You want to give your users access to data without interacting directly with the data source itself.
- You want to improve the speed of queries.
- You want your end users to run queries but not update a database table.
- You do not want to expose your data source.

Indexing CFLDAP Query Results

The widespread use of the Lightweight Directory Access Protocol to build searchable directory structures, both internally and across the Web, provides ColdFusion developers with new opportunities to add value to the sites they create. Contact information or other data from an LDAP-accessible server can be indexed and searched by users. Remember to create the collection in the Administrator.

Two things to remember when creating an index from an LDAP query:

- Because LDAP structures vary greatly, you must know the server's directory schema and the exact name of every LDAP attribute you intend to use in a query.
- The records on an LDAP server can be subject to frequent change. You may want to re-index the collection before processing a search request.

In the example below, the search criterion is records with a telephone number in the 617 area code. Generally, LDAP servers use the Distinguished Name (dn) attribute as the unique identifier for each record, so that is used as the KEY value for the index.

```
<!--- Run the LDAP query --->
<CFLDAP NAME="OrgList"
    SERVER="my.ldapserver.com"
    ACTION="query"
    ATTRIBUTES="o, telephonenumber, dn, mail"
    SCOPE="onelevel"
    FILTER="(|(O=a*) (O=b*))"
    SORT="o"
    START="c=US">

<!--- Output query result set --->
<CFOUTPUT QUERY="OrgList">
    DN: #dn# <BR>
    O: #o# <BR>
    TELEPHONENUMBER: #telephonenumber# <BR>
    MAIL: #mail# <BR>
==============================<BR>
</CFOUTPUT>

<!--- Index the result set --->
```

```
<CFINDEX ACTION="update"
    COLLECTION="ldap_query"
    KEY="dn"
    TYPE="custom"
    TITLE="o"
    QUERY="OrgList"
    BODY="telephonenumber">

<!--- Search the collection --->
<!--- Use the wildcard * to contain the search string --->
<CFSEARCH COLLECTION="ldap_query"
    NAME="s_ldap"
    CRITERIA="*617*">

<!--- Output returned records --->
<CFOUTPUT QUERY="s_ldap">
    #Key#, #Title#, #Body# <BR>
</CFOUTPUT>
```

Indexing CFPOP Query Results

The contents of mail servers are generally quite volatile; specifically, the MessageNumber is reset as messages are added and deleted. To avoid mismatches between the unique MessageNumber identifiers on the server and in the Verity collection, it's a good idea to re-index the collection before processing a search.

As with the other query types, you need to provide a unique value for the KEY attribute and enter the data fields to index in the BODY attribute.

```
<!--- Run POP query --->
<CFPOP ACTION="getall"
    NAME="p_messages
    SERVER="mail.mycompany.com"
    USERNAME="user1"
    PASSWORD="user1">

<!--- Output POP query result set --->
<CFOUTPUT QUERY="p_messages">
    #MESSAGENUMBER# <BR>
    #FROM# <BR>
    #TO# <BR>
    #SUBJECT# <BR>
    #BODY# <BR>
==========================<BR>

<!--- Index result set --->
<CFINDEX ACTION="update"
    COLLECTION="pop_query"
    KEY="messagenumber"
    TYPE="custom"
    TITLE="subject"
```

```
    QUERY="p_messages"

    BODY="body">

<!--- Search messages for the word "action" --->
<CFSEARCH COLLECTION="pop_query"
    NAME="s_messages"
    CRITERIA="action">

<!--- Output search result set --->
<CFOUTPUT QUERY=" s_messages">
    #Key#, #Title# <BR>
</CFOUTPUT>
```

The CFSEARCH code in the examples above uses the basic attributes needed to search a collection. The next section expands on the capabilities of this tag for more detailed input and output options.

Managing Collections

As with any data source, the maintenance requirements of a Verity collection are dictated by the amount, frequency, and type of changes that occur in the records. You can run maintenance routines directly from either the CFCOLLECTION or CFINDEX tags or via the Administrator Verity page. The newer For more information on this topic, see the Allaire Knowledge Base article, "Maintaining Collections" (ID# 1080) on our Web site.

Maintenance options

Choose an option based on the following function descriptions.

- Repair — Runs internal Verity routines to fix corrupted records. If you suspect a collection has become corrupted, it is probably safest to re-populate it.
- Optimize — Packs the indexed data for better performance. This is similar to database optimization. This procedure can be used as part of routine maintenance. The Optimize action is deprecated for CFINDEX except to maintain legacy code; the newer CFCOLLECTION tag is recommended instead. For more information on this command, see the Allaire Knowledge Base article, "How To Optimize Your Verity Collection" (ID# 416) on our Web site.
- Purge — Removes all data from a collection.
- Delete (when used as a CFINDEX ACTION) — Deletes the specified KEY value, or comma-separated values, from the collection.
- Delete (when used on the Administrator Verity page or in CFCOLLECTION) — Deletes the entire collection.
- Update — Re-populates the collection with changed records and new records and adds a key if one is not part of the collection. This operation does not delete records that have been deleted from the data source. To update a collection

from the Administrator Verity main page, select a collection on the list, click Index, then click Update on the index page.

- Refresh (CFINDEX ACTION only) — Deletes all data and re-populates the collection.

Scheduling collection maintenance

The easiest way to perform collection management tasks is to create a ColdFusion template that runs the operations, then add the task on the Administrator Scheduler page. The page presents a wide range of scheduling options.

Securing a collection

A couple of possible scenarios for restricting access to a Verity collection are:

- The ColdFusion Administrator may need to specify developer access to collections.
- A public site may need to limit user access to collections.

To restrict access to a collection, follow these steps:

1. Open the Advanced Server Security page of the ColdFusion Administrator and click the Use Advanced Server Security box.
2. Click the Security Contexts button.
3. Enter a name for the secured collection and click Add.
4. Optionally enter a description for the secured collection.
5. Click Collections on the Enable Security for Resource Types list.
6. Click Apply.

An appropriate authentication interface can then be developed to allow access to the secured collection. See the *Advanced ColdFusion Development* book for information on advanced user security and authentication.

Building a Search Interface

The CFSEARCH tag provides users with a set of operators and modifiers to create sophisticated query expressions. We'll explore these options in detail below, but first let's take a look at getting a basic search application up and running.

The Verity wizard

To quickly create a search application for an existing collection, click the File > New command in ColdFusion Studio and select the Verity Wizard in the CFML tab of the

New Document dialog. The wizard creates a set of application pages based on the entries you make in the wizard dialogs.

You can customize the search interface by adding instructional text for users and applying styles to the form pages.

Operators and modifiers

The power of the CFSEARCH tag is in the control it gives you over the Verity search engine. The engine offers users a high degree of specificity in setting search parameters.

Operators

An operator represents logic to be applied to a search element. This logic defines the qualifications a document must meet to be retrieved. Operators are used to refine your search or to influence the results in other ways. For example, you could construct an HTML form for conducting searches. In the form, a user could perform a search for a single term: server. You can refine your search by limiting the search scope in a number of ways. Operators are available for limiting a query to a sentence or paragraph, and you can search words based on proximity. The following operator types are available:

- Evidence operators — Used to specify basic and intelligent word searches.
- Proximity operators — For specifying the relative location of words in a document.
- Relational operators — Search fields in a collection.
- Concept operators — Used to identify a concept in a document by combining the meanings of search elements.
- Score operators — Allow you to manipulate the score returned by a search element. The score percentage display can optionally be set to as many as four decimal places.
- Natural language operators — Allow the use of natural language expressions in forming queries.

Ordinarily, you use operators in explicit searches. They are used in the following manner:

```
"<operator>search_string"
```

Modifiers

Modifiers can be used with operators to further refine query expressions. You can specify case sensitivity in a query, or force the output to be ranked by relevancy. Modifiers include:

- CASE — Sets case sensitivity. Verity searches are case-insensitive unless mixed case characters are entered as CRITERIA values.

- MANY — Results are ranked by relevancy, which is determined by the number of times the search value is found in a document.
- NOT — Eliminates documents containing the specified words.
- ORDER — Returns documents only if they contain words in the listed order.

Basic search operations

Conducting a basic search is very straightforward using the CFSEARCH tag. In the following example, a simple search for the word "database" is performed on a collection created from a directory of HTML files called "DOCS."

```
<CFSEARCH NAME="DocSearch"
    COLLECTION="DOCS"
    TYPE="Simple"
    CRITERIA="database">
```

You use the CFOUTPUT tag to present the results of your search to your user. The following example shows a fully defined URL:

```
<CFOUTPUT QUERY="DocSearch">
    <A HREF="#DocSearch.url#">
    #DocSearch.Title#</A><BR>
</CFOUTPUT>
```

The DocSearch.Title variable presents the contents of the HTML TITLE tag. This facility gives you excellent indexing options for managing large numbers of HTML documents. With judicious use of the TITLE tag, you could offer excellent searching options to users.

Result columns

Every search conducted with the CFSEARCH tag returns up to seven result columns you can reference in your CFOUTPUT:

- URL — Returns the value of the URLPATH attribute defined in the CFINDEX tag used to populate the collection. This value is always empty when you populate the collection with CFINDEX with TYPE=CUSTOM.
- KEY — Returns the value of the KEY attribute defined in the CFINDEX tag used to populate the collection.
- TITLE — Returns the value of the TITLE attribute defined by the <TITLE> HTML tag in any HTML or ColdFusion application page file that was indexed by CFINDEX. If the collection was TYPE=CUSTOM, TITLE returns the value of the TITLE attribute defined by the CFINDEX tag. If the collection was TYPE=FILE, TITLE also returns the value of the TITLE attribute defined by the CFINDEX tag.
- SCORE — Returns the relevancy score of the document based on the search criteria.
- CUSTOM1 — The value returned from the CUSTOM1 attribute in CFINDEX.
- CUSTOM2 — The value returned from the CUSTOM2 attribute in CFINDEX.

- SUMMARY — The contents of the automatic summary generated by CFINDEX.

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

```
#DocSearch.URL#
#DocSearch.KEY#
#DocSearch.TITLE#
#DocSearch.SCORE#
```

Summarization

As part of the indexing process, Verity automatically produces a summary of every document file or query result set. The default summarization selects the best sentences, based on internal rules, up to a maximum of 500 characters. Summarization information is returned by default with every CFSEARCH operation. For more information on this topic, see the Allaire Knowledge Base article, "Custom1, Custom2 and Summary Fields" (ID# 1081) on our Web site.

To access the summary, invoke the property in the following form:

```
#search_query.Summary#
```

For example, in a search operation where the value of the NAME attribute is "mysearch" the following CFML outputs the summary of the search results:

```
<CFOUTPUT QUERY="mysearch">
    #Summary#<BR>
</CFOUTPUT>
```

For information on an advanced summarization technique, see the Allaire Knowledge Base article, "Synchronizing information stored in Verity Collection Document Fields with Corresponding Data from a Database" (ID# 1161) on our Web site.

CFSEARCH properties

Three properties are generated for each CFSEARCH query that provide information about a particular query:

- RecordCount — The total number of records returned by the query.
- CurrentRow — The current row of the query being processed by CFOUTPUT.
- RecordsSearched — The total number of records in the index that were searched. If no records were returned in the search, this property returns a null value.

You can use CFSEARCH properties in much the same way you use other ColdFusion system variables. For example, you can easily determine how many records were returned in a search with the following code:

```
<CFOUTPUT>
    <P>Your search returned
    #queryname.RecordCount# records.</P>
</CFOUTPUT>
```

```
<--- Present a message to the user if no records are returned --->
<CFIF #cfuser.RecordCount# LTE 0>
    <CFOUTPUT>
        <P>Sorry no instances of "#Form.searchquery#" found.</P>
    </CFOUTPUT>

<CFELSEIF #cfuser.RecordCount# GT 0>
    <CFOUTPUT QUERY="cfuser">
        <A HREF="#cfuser.url#">#cfuser.Title#</A><BR>
    </CFOUTPUT>
</CFIF>
```

Using Query Expressions

When you search a Verity collection, you use the CFSEARCH tag in a ColdFusion application page. Use the CRITERIA attribute to specify the query expression you want to pass to the search engine.

You can build two types of query expressions: simple and explicit. A simple query expression is typically a word or words. An explicit query expression can employ a number of operators and modifiers to refine the search. Although an explicit query can employ operators and modifiers, all aspects of the search must be explicitly invoked. A simple query expression is somewhat more powerful since it employs operators by default. You can assemble an explicit query expression programmatically or simply pass a simple query expression to the search engine directly from an HTML input form.

The Verity query language provides many operators and modifiers for composing queries. The following search techniques can be used in searching a Verity collection:

- Word searches
- Proximity searches
- Concept–based
- Field searches in which documents are match based on matching predefined custom attributes
- Scoring operators

Simple query expressions

Simple queries allow end users to enter simple, comma-delimited strings and use wildcard characters. By default, a simple query searches for words, not strings. For example, entering the word "All" will find documents containing the word "all" but not "allegorical." You can use wildcards, however to broaden the scope of the search. "All*" will return documents containing both "all" and "alliterate." Case is ignored.

You can enter multiple words separated by commas: software, Microsoft, Oracle. The comma in a Simple query expression is treated like a logical OR. If you omit the

commas, the query expression is treated as a phrase, so documents would be searched for the phrase "software Microsoft Oracle."

Ordinarily, operators are employed in explicit query expressions. Operators are normally surrounded by angle brackets < >. However, you can use the AND, OR, and NOT operators in a simple query without using angle brackets: software AND (Microsoft OR Oracle). To include an operator in a search, you surround it with double quotation marks: software "and" Microsoft. This expression searches for the phrase "software and Microsoft."

A simple query employs the STEM operator and the MANY modifier. STEM searches for words that derive from those entered in the query expression, so that entering "find" will return documents that contain "find," "finding," "finds," etc. The MANY modifier forces the documents returned in the search to be presented in a list based on a relevancy score.

Explicit query expressions

Explicit queries can be constructed using a variety of operators, including evidence, proximity, relational, concept, and score operators. Most operators in an explicit query expression are surrounded by angle brackets < >. You can use the AND, OR, and NOT operators without angle brackets.

Expression syntax

You can use either simple or explicit syntax when stating simple query syntax. The syntax you use determines whether the search words you enter will be stemmed, and whether the words that are found will contribute to relevance-ranked scoring.

Simple syntax

When you use simple syntax, the search engine implicitly interprets single words as if they were modified by the MANY and STEM operators. By implicitly applying the MANY modifier, the search engine calculates each document's score based on the density of the search term in the searched documents. The more frequent the occurrence of a word in a document, the higher the document's score.

As a result, the search engine ranks documents according to word density as it searches for the word you specify, as well as words that have the same stem. For example, "films," "filmed," and "filming" are stemmed variations of the word "film." To search for documents containing the word "film" and its stem words, you can enter the word "film" without modification. When documents are ranked by relevance, they appear in a list with the most relevant documents at the top.

Explicit syntax

When you use explicit syntax, the search engine interprets the search terms you enter as literals. For example, by entering the word "film" (including quotation marks) using

explicit syntax, the stemmed versions of the word "film", "films," "filmed," and "filming" are ignored.

The following table shows all operators available for conducting searches of ColdFusion Verity collections.

Verity Search Operators		
<	CONTAINS	PHRASE
<=	ENDS	SENTENCE
=	MATCHES	STARTS
>	NEAR	STEM
>=	NEAR/N	SUBSTRING
Accrue	OR	WILDCARD
AND	PARAGRAPH	WORD

Special characters

A number of characters are handled in particular ways by the search engine.

Special Search Characters	
Characters	**Description**
, ( ) [	These characters end a text token.
= > < !	These characters also end a text token. They are terminated by an associated end character.
' @ ' < { [!	These characters signify the start of a delimited token. They are terminated by an associated end character.

A backslash (\) removes special meaning from whatever character follows it. To enter a literal backslash in a query, use two in succession, such as this examples:

```
<FREETEXT>("\"Hello\", said Packard.")
"backslash (\\)"
```

Precedence Evaluation

The following rules apply for composing search expressions.

Precedence rules

While an expression is read from left to right, some operators carry more weight than others. For example, AND operators take precedence over OR operators. To ensure that an OR operator is interpreted prior to an AND operator, you can use parentheses to enclose the OR operator:

```
(a OR b) AND c
```

Terms enclosed by parentheses are read first.

There must be at least one space between operators and words used in the expression.

When the search engine encounters nested parentheses, it starts with the innermost term:

```
(a AND (b OR c)) OR d
```

This expression means: Look for documents that contain b or c as well as a, or that contain d.

Prefix and infix notation

Search strings that use any operator other than evidence operators can be defined in prefix notation or infix notation.

Prefix notation specifies that the operator comes before the search string:

```
AND (a,b)
```

When prefix notation is used, precedence is handled explicitly within the expression. The following example means: "Look for documents that contain b and c first, then documents that contain a":

```
OR (a, AND (b,c))
```

Infix notation specifies that the operator is to be specified between each term within the expression. The following example means: "Look for documents that contain a and b or documents that contain c":

```
a AND b OR c
```

When infix notation is used, precedence is implicit in the expression. For example, the AND operator takes precedence over the OR operator.

Commas in expressions

If an expression includes two or more search terms within parentheses, a comma is required as a separator between each element. The following example means: Look for documents that contain any combination of a and b together. Note that in this example, angle brackets are used with the OR operator.

```
<OR> (a, b)
```

Delimiters in expressions

Angle brackets < >, double quotation marks " ", and backslashes \ are used to delimit various elements in a query expression.

Angle brackets for operators

Left and right angle brackets < > are reserved for designating operators and modifiers. They are optional for the AND, OR, and NOT operators, but required for all other operators.

Double quotation marks in expressions

You use double quotation marks to search for a word that is otherwise reserved as an operator, such as AND, OR, and NOT.

Backslashes in expressions

To include a backslash \ in a search, insert two backslashes for each backslash character you want to search for, such as `C:\\CFUSION\\BIN`.

Searching with Wildcards

This table shows the wildcard characters for searching Verity collections.

Verity Wildcard Characters	
Wildcard	**Description**
?	Question. Specifies any single alphanumeric character.
*	Asterisk. Specifies zero or more alphanumeric characters. Avoid using the asterisk as the first character in a search string. Asterisk is ignored in a set, [ ] or an alternative pattern { }.
[ ]	Square brackets. Specifies one of any character in a set, as in "sl[iau]m" which locates "slim," "slam," and "slum." Square brackets indicate an implied OR.
{ }	Curly braces. Specifies one of each pattern separated by a comma, as in "hoist{s, ing, ed}" which locates "hoists," "hoisting," and "hoisted." Curly braces indicate an implied AND.

Verity Wildcard Characters (Continued)	
Wildcard	**Description**
^	Caret. Specifies one of any character not in the set as in "sl[^ia]m" which locates "slum" but not "slim" or "slam."
-	Hyphen. Specifies a range of characters in a set as in "c[a-r]t" which locates every word beginning with "c," ending with "t," and containing any letter from "a" to "r."

Searching for wildcards as literals

To search for a wildcard character in your collection, you need to escape the character with a backslash (\). For example:

To match a literal asterisk, you precede the * with two backslashes: "a*"

To match a question mark or other wildcard character: "Checkers\?"

Searching for special characters as literals

The following non-alphanumeric characters must be preceded by a backslash character (\) in a search string:

- comma (,)
- left and right parentheses ( )
- double quotation mark (")
- backslash (\)
- at sign (@)
- left curly brace ({)
- left bracket ([)
- less than sign (<)
- backquote (‘)

In addition to the backslash character, you can use paired backquotes (‘ ‘) to interpret special characters as literals. For example, to search for the wildcard string "a{b" you can surround the string with backquotes, as follows:

```
‘a{b‘
```

To search for a wildcard string that includes the literal backquote character (‘) you must use two backquotes together and surround the whole string in backquotes:

```
‘*n‘‘t‘
```

Note that you can use either paired backquotes or backslashes to escape special characters. There is no functional difference in the use of one or the other. For example, you can query for the term: <DDA> in the following ways:

```
\<DDA\> or `<DDA>`
```

Operators and Modifiers

This chapter presents details on the search operator and modifier types.

Evidence operators

Evidence operators can be used to specify either a basic word search or an intelligent word search. A basic word search finds documents that contain only the word or words specified in the query. An intelligent word search expands the query terms to create an expanded word list so that the search returns documents that contain variations of the query terms.

Documents retrieved using evidence operators are not ranked by relevance unless you use the MANY modifier.

Verity Evidence Operators	
Operator	**Description**
STEM	Expands the search to include the word you enter and its variations. The STEM operator is automatically implied in any SIMPLE query. For example, the EXPLICIT query expression `<STEM>believe` yields matches such as, "believe," "believing," "believer".'
WILDCARD	Matches wildcard characters included in search strings. Certain characters automatically indicate a wildcard specification, such as * and?. For example, the query expression `spam*` yields matches such as, "spam," "spammer," "spamming."
WORD	Performs a basic word search, selecting documents that include one or more instances of the specific word you enter. The WORD operator is automatically implied in any SIMPLE query.

Proximity operators

Proximity operators specify the relative location of specific words in the document. Specified words must be in the same phrase, paragraph, or sentence for a document to be retrieved. In the case of NEAR and NEAR/N operators, retrieved documents are

ranked by relevance based on the proximity of the specified words. Proximity operators can be nested; phrases or words can appear within SENTENCE or PARAGRAPH operators, and SENTENCE operators can appear within PARAGRAPH operators.

The following table describes each operator.

Verity Proximity Operators	
Operator	**Description**
NEAR	Selects documents containing specified search terms. The closer the search terms are to one another within a document, the higher the document's score. The document with the smallest possible region containing all search terms always receives the highest score. Documents whose search terms are not within 1000 words of each other are not selected.

NEAR/*N*	Selects documents containing two or more search terms within *N* number of words of each other, where *N* is an integer between 1 and 1024 where NEAR/1 searches for two words that are next to each other. The closer the search terms are within a document, the higher the document's score.
	You can specify multiple search terms using multiple instances of NEAR/*N* as long as the value of N is the same:
	`commute <NEAR/10> bicycle <NEAR/10>`
	`train <NEAR/10>`

PARAGRAPH	Selects documents that include all of the words you specify within the same paragraph. To search for three or more words or phrases, you must use the PARAGRAPH operator between each word or phrase.

PHRASE	Selects documents that include a phrase you specify. A phrase is a grouping of two or more words that occur in a specific order. Examples of phrases:
	`mission oak`
	`"mission oak"`
	`mission <PHRASE> oak`
	`<PARAGRAPH> (mission, oak)`

SENTENCE	Selects documents that include all of the words you specify within the same sentence. Examples:
	`jazz <SENTENCE> musician`
	`<SENTENCE> (jazz, musician)`

Relational operators

Relational operators search document fields that have been defined in the collection. Documents containing specified field values are returned. Documents retrieved using relational operators are not ranked by relevance, and you cannot use the MANY modifier with relational operators.

Numeric and date relational operators

The following operators are used for numeric and date comparisons.

Verity Numerical and Date Relational Operators	
Operator	**Description**
=	Equals
>	Greater than
>=	Greater than or equal to
<	Less than
<=	Less than or equal to

Text comparison operators

Text comparison operators match words and parts of words. The following operators are used for text comparisons.

Verity Comparison Operators	
Operator	**Description**
CONTAINS	Selects documents by matching the word or phrase you specify with the values stored in a specific document field. Documents are selected only if the search elements specified appear in the same sequential and contiguous order in the field value. For example, specifying "god" will match "God in heaven," "a god among men," or "good god" but not "godliness," or "gods."
MATCHES	Selects documents by matching the query string with values stored in a specific document field. Documents are selected only if the search elements specified match the field value exactly. If a partial match is found, a document is not selected. For example, specifying "god" will match a document field containing only "god" and will not match "gods," "godliness," or "a god among men."
STARTS	Selects documents by matching the character string you specify with the starting characters of the values stored in a specific document field.
ENDS	Selects documents by matching the character string you specify with the ending characters of the values stored in a specific document field.
SUBSTRING	Selects documents by matching the query string you specify with any portion of the strings in a specific document field. For example, specifying "god" will match "godliness," "a god among men," "godforsaken," etc.

Document fields

The values you specify for the CFINDEX attributes TITLE, KEY, URL, and CUSTOM can be specified as document fields for use with relational operators in the CRITERIA attribute. Document fields are referenced in text comparison operators. They are identified as:

- CF_TITLE
- CF_KEY
- CF_URL

- CF_CUSTOM1
- CF_CUSTOM2

For more information on this topic, see the Allaire Knowledge Base article, "Using Document Fields To Narrow Down Searches" (ID# 1082) on our Web site.

The SUBSTRING operator

You can use the SUBSTRING operator to match a character string with data stored in a specified data source. In the following example, a data source called TEST1 contains the table YearPlaceText, which itself contains three columns: Year, Place, and Text. Year and Place make up the primary key. The following table shows the TEST1 schema.

YearPlaceText		
Year	**Place**	**Text**
1990	Utah	Text about Utah 1990
1990	Oregon	Text about Oregon 1990
1991	Utah	Text about Utah 1991
1991	Oregon	Text about Oregon 1991
1992	Utah	Text about Utah 1992

The following application page matches records that have 1990 in the TEXT column and are in the Place Utah. The search is performed against the collection that contains the TEXT column and then is narrowed further by searching the string "Utah" in the CF_TITLE document field. Recall that document fields are defaults defined in every collection corresponding to the values you define for URL, TITLE, and KEY in the CFINDEX tag.

```
<CFQUERY NAME="GetText"
    DATASOURCE="TEST1">
    SELECT Year+Place
        AS Identifier, text
        FROM YearPlaceText
</CFQUERY>

<CFINDEX COLLECTION="testcollection"
    ACTION="Update"
    TYPE="Custom"
    TITLE="Identifier"
    KEY="Identifier"
    BODY="TEXT"
    QUERY="GetText">

<CFSEARCH NAME="GetText_Search"
    COLLECTION="testcollection"
```

```
    TYPE="Explicit"
    CRITERIA="1990 and CF_TITLE <SUBSTRING> Utah">
<CFOUTPUT>
    Record Counts: <BR>
    #GetText.RecordCount# <BR>
    #GetText_Search.RecordCount# <BR>
</CFOUTPUT>

<CFOUTPUT>
    Query Results --- Should be 5 rows <BR>
</CFOUTPUT>

<CFOUTPUT QUERY="Gettext">
    #Identifier# <BR>
</CFOUTPUT>

<CFOUTPUT>
    Search Results -- should be 1 row <BR>
</CFOUTPUT>

<CFOUTPUT QUERY="GetText_Search">
    #GetText_Search.TITLE# <BR>
</CFOUTPUT>
```

Concept operators

Concept operators combine the meaning of search elements to identify a concept in a document. Documents retrieved using concept operators are ranked by relevance. The following table describes each concept operator.

Verity Concept Operators	
Operator	**Description**
AND	Selects documents that contain all of the search elements you specify.
OR	Selects documents that show evidence of at least one of the search elements you specify.
ACCRUE	Selects documents that include at least one of the search elements you specify. Documents are ranked based on the number of search elements found.

Score operators

Score operators govern how the search engine calculates scores for retrieved documents. The maximum score a returned search element can have is 1.000. The score percentage display can optionally be set to as many as four decimal places.

When a score operator is used, the search engine first calculates a separate score for each search element found in a document, and then performs a mathematical operation on the individual element scores to arrive at the final score for each document.

Note that the document's score is available as a result column. The SCORE result column can be referenced to trap the relevancy score of any document retrieved. For example:

```
<CFOUTPUT>
    <A HREF="#Search1.URL#">#Search1.Title#</A><BR>
    Document Score=#Search1.SCORE#<BR>
</CFOUTPUT>
```

The following table lists the score operators.

Verity Score Operators	
Operator	**Description**
YESNO	Forces the score of an element to 1 if the element's score is non-zero: `<YESNO>mainframe` If the retrieval result of the search on "mainframe" is 0.75, the YESNO operator forces the result to 1. You can use YESNO to avoid relevance ranking.
PRODUCT	Multiplies the scores for documents matching a query. To arrive at a document's score, the search engine calculates a score for each search element and multiplies these scores together: `<PRODUCT>(computers, laptops)` The resulting score for each document is multiplied together.
SUM	Adds together the scores for documents matching a query, up to a maximum value of 1: `<SUM>(computers, laptops)` The resulting scores are added together.
COMPLEMENT	Calculates scores for documents matching a query by taking the complement (subtracting from 1) of the scores for the query's search elements. The new score is 1 minus the search element's original score. `<COMPLEMENT>computers` If the search element's original score is.785, the COMPLEMENT operator recalculates the score as.215.

Search modifiers

Modifiers are combined with operators to change the standard behavior of an operator in some way. For example, you can use the CASE modifier with an operator to specify that you want to match the case of the search word.

Modifiers are as follows.

Verity Search Modifiers	
Modifier	**Description**
CASE	Specifies a case-sensitive search: <CASE>J[AVA, ava] Searches for "JAVA" and "Java." If a search contains a mixed-case string, the search request will be case-sensitive.
MANY	Counts the density of words, stemmed variations, or phrases in a document and produces a relevance-ranked score for retrieved documents. Can be used with the following operators: WORD WILDCARD STEM PHRASE SENTENCE PARAGRAPH <PARAGRAPH><MANY>javascript <AND> vbscript The MANY modifier cannot be used with the following: AND OR ACCRUE Relational operators

Verity Search Modifiers (Continued)	
Modifier	**Description**
NOT	Used to exclude documents that contain the specified word or phrase. Used only with the AND and OR operators. `Java <AND> programming <NOT> coffee`
ORDER	Used to specify that the search elements must occur on the same order in which they were specified in the query. Can be used with the following operators: PARAGRAPH SENTENCE NEAR/*N* Place the ORDER modifier before any operator: `<ORDER><PARAGRAPH>("server", "Java")`

Collection Examples

The following code examples demonstrate a very basic approach to populating and searching a collection of documents and a collection of data from a ColdFusion query. To run the example, save the code examples into the indicated file names all in the same directory. Then open start.cfm to run the example.

Example: Choose the collection

This first sample, `start.cfm`, provides a simple starting point for populating a collection. You can use the code as is, with a few minor changes. Note that for this sample to work, you'll first need to create two collections: one called `DBINDEX` and the other, `DOCS`.

```
Start.cfm
<HTML>
<HEAD>
    <TITLE>Verity Samples</TITLE>
</HEAD>

<H2>Pick which index you want to build</H2>
<P>Select the collection you want to populate:</P>

<FORM METHOD="POST" ACTION="index.cfm">
    <INPUT TYPE=radio NAME=collection VALUE=DBINDEX checked>
        Message table from CF 4.0 Examples<BR>
    <INPUT TYPE=radio NAME=collection VALUE=DOCINDEX>
        Documents in the Web server root directory tree<BR><BR>
    <INPUT TYPE=SUBMIT NAME=doindex VALUE="Populate">
```

```
</FORM>

<H3>Skip populating the collection, go right
to <A HREF="search.cfm"> searching a collection</A></H3>
</BODY>
</HTML>
```

Example: Populate the collection

Now populate the collection you chose:

```
Index.cfm
<HTML>
<HEAD>
    <TITLE>Verity Custom Tag Tests</TITLE>
</HEAD>

<CFQUERY NAME="Messages"
    DATASOURCE="CF 4.0 Examples">
    SELECT *
        FROM Messages
</CFQUERY>

<BODY>
<CFIF #form.collection# IS "DBINDEX">
    <CFINDEX COLLECTION="DBINDEX"
        ACTION="UPDATE"
        TYPE="CUSTOM"
        BODY="Body"
        KEY="Message_ID"
        TITLE="UserName"
        QUERY="Messages">
<CFELSE>
    <CFINDEX COLLECTION="DOCS"
        TYPE="PATH"
    KEY="c:\inetpub\wwwroot"
    URLPATH="http://127.0.0.1/"
    EXTENSIONS=".htm, .html, .cfm, .cfml"
    RECURSE="YES">
</CFIF>

<H2>Collection Successfully Generated</H2>
<H3><A HREF="search.cfm">Search the collection</A></H3>

</BODY>
</HTML>
```

Example: Search the collection

With the collections populated, you can now offer a form for searching a collection:

```
Search.cfm

<HTML>
<HEAD>
    <TITLE>CFINDEX and CFSEARCH samples</TITLE>
</HEAD>

<BODY>
<H2>Search</H2>
<P><B>Select search type:</B></P>
<FORM METHOD="POST" ACTION="vfp_search.cfm"><P>
    <INPUT TYPE=radio
        NAME=type
        VALUE=Simple checked> Simple<BR>
    <INPUT TYPE=radio
        NAME=type
        VALUE=Explicit> Explicit<P>
<P><B>Select data source:</B></P>
    <INPUT TYPE=radio
        NAME=source
        VALUE=DBINDEX checked>
Messages Table<BR>
    <INPUT TYPE=radio
        NAME=source
        VALUE=DOCS>
<P>Web doc root directory</P>
<P>Search string:</P>
    <INPUT TYPE=text
        NAME=searchstring SIZE=50><P>
    <INPUT TYPE=SUBMIT
        NAME=search1
        VALUE="Search">
    <INPUT TYPE=reset
        VALUE="Reset">
</FORM>

</BODY>
</HTML>
```

Example: Present the search results

The following example processes the search results.

```
vfp_search.cfm
<HTML>
<HEAD>
    <TITLE>Search output template</TITLE>
</HEAD>

<CFIF #form.source# IS "DBINDEX">
    <CFSET #type#="messages">
<CFELSE>
    <CFSET #type#="files">
</CFIF>
```

```
<BODY>

<CFSEARCH NAME="Search1"
    COLLECTION="#form.source#"
    FORM TYPE="#form.type#"
    CRITERIA="#form.searchstring#">

<H2>Search Results</H2>

<CFOUTPUT>
    #Search1.RecordCount# #type# found out of
    #Search1.RecordsSearched# #type# searched.
</CFOUTPUT>

<HR NOSHADE>

<CFIF #form.source# IS "DBINDEX">
    <CFOUTPUT QUERY="Search1">
        <A HREF="cf_tag_verity_detail.cfm?ID=#Search1.KEY#">
        Message #Search1.KEY# posted by #Search1.TITLE#</A><BR>
        Score = #Search1.SCORE#<BR>
    </CFOUTPUT>
<CFELSE>
    <CFOUTPUT QUERY="Search1">
        <A HREF="#Search1.URL#">#Search1.Title#</A><BR>
    </CFOUTPUT>
</CFIF>

<HR NOSHADE>
</BODY>
</HTML>
```

Example: Output from searching the DBINDEX collection

This application page presents each individual message.

cf_tag_verity_detail.cfm

```
<CFQUERY NAME="MessageDetail"
    DATASOURCE="CF 4.0 Examples">
    SELECT UserName, Subject, Body
        FROM Messages
        WHERE Message_ID = #ID#
</CFQUERY>

<CFOUTPUT QUERY="MessageDetail">
<PRE>
    Message #URL.ID# <BR>
    Posted by: #MessageDetail.UserName# <BR>
    Subject: #MessageDetail.Subject# <BR>
    #MessageDetail.Body#
</PRE>
</CFOUTPUT>
```

CHAPTER 18

Managing Files

This chapter outlines options and commands for viewing and managing local and remote files.

Contents

Working with Local Files

The Local Files tab gives you easy access to files stored on your PC or network drive. The top pane displays available drives, the middle pane shows the directory tree for the selected drive, and the bottom pane presents the file list for the selected folder.

To access files in the Files Resource tab:

1. Click the Local Files tab at the bottom of the Resources area.
2. Navigate to the drive and directory you want to access.
3. Double-click on a folder or click on the + sign to expand the view.
4. Click on a folder to display its files in the bottom pane.
5. Double-click on a file to open it in the editor.

Note When you open a read-only files, it is marked with a red dot on the document tab. Right-click on the file in the file list and select Properties to change a file's attributes. This is not recommended if you are using a source control application to manage read and write privileges.

To open a page from a Web site:

1. Choose File > Open From the Web.
2. Enter the URL for the document or select a file from your Bookmarks or Favorites list.
3. If the remote site is accessed via a proxy server, click Proxy and enter the server name and port number.
4. You can optionally set a time-out limit for the connection.

Note Only HTML files can be opened with this command. Opened files cannot be saved back to the server. For remote file management procedures, see the Working with Remote Files section below.

To filter the file list:

You can set the files list to show specific file types. The filter is a global setting for all directories.

1. Right-click in the File pane and choose Filter.
2. Select a file type from the menu. The file list refreshes to apply the filter.

Open the Options > Settings (F8) File Settings tab to check the current list of file extensions for the Web Documents menu item. You can edit this list as needed. The Web Images option displays Web-supported graphic formats.

Working with Remote Files

Support for remote development is tightly integrated with Studio, eliminating the need for complicated network configurations or additional applications to access remote files. Studio provides two ways to work with remote files:

- **FTP** — Create, open, edit, and save files on a remote server using the File Transfer Protocol.
- **RDS** — Remote Development Services — Connect to a Cold Fusion Application Server to access files and databases via HTTP.

To work with remote files:

1. Add FTP and RDS servers.
2. Connect to the remote server.

Adding an FTP server

You add servers in the Remote Files tab in the Resources area.

To add an FTP server:

1. Click the Remote Files Resource tab.
2. Right-click in the file pane and select Add FTP Server.
3. Complete the fields in the Configure FTP Server dialog:
    - Description — Required. The text that displays in the server list.
    - Host Name — Required. The server domain name, such as `www.allaire.com` or an IP address.
    - Initial Directory — Optional. The root directory of the FTP server.
    - User Name — Required. Your login name or "anonymous" for anonymous FTP servers.
    - Password — Optional. Password with user name.
    - Root URL — Optional. Root URL for link verification.
    - Remote Port — The port on the server used by the FTP server. Use the default unless specified by server administrator.
    - Request Timeout — Set a value for the amount of time Studio waits for a server connection to complete.
    - Prompt for Password — Check this box to require a user password at login.
    - Use passive mode — Required for servers that use passive connections.
    - Assume UTC file times — Check this box if the FTP server uses the UTC time format.

4. Click OK to complete the dialog.

Note To edit server settings, right-click on a server name in the Remote Server list and select Server Properties to open the Properties dialog.

To add an RDS server:

1. Click the Remote Files tab.
2. Right-click in the file list and select Add RDS Server.
3. Complete the Remote Host fields in the Configure RDS Server dialog:
    - Description — Required. The name that appears in the server list.
    - Host Name — Required. The server domain name, such as `www.allaire.com` or an IP address.
    - User Name — Required if the server is password-protected. Open the CF Administrator Server page to edit the Studio password.
    - Password — Optional. If the server is password-protected you must use a password. Open the CF Administrator Server page to edit the Studio password.
    - Port — Required. The port on the server used by the Web server. Use default unless specified by server administrator.
    - Use Secure Sockets Layer (SSL) — Optional. Allows exchanges between Studio and the server to be encrypted via SSL.
4. Complete the ColdFusion RDS Security fields:
    - User Name — Required if the server is password-protected. Open the CFAS Administrator Server page to edit or disable the Studio password.
    - Password — Optional. If the server is password-protected you must use a password. Open the CFAS Administrator Server page to edit or disable the Studio password.
5. Click Prompt for Password to enable these entries.
6. Click OK to complete the dialog.

Accessing a remote server

Once you've added a remote server connection, it is easy to open that server and work with the files.

To open files on a remote server:

1. Click the Remote Files tab.
2. Double-click on a server name or right-click and choose Connect.
3. Double-click on drives and directories to locate files.

4. Double-click on a file to open it in the editor. Remote files are indicated by a blue dot next to their names on the file name tab at the bottom of the editor window.

You can work with remote files just as you do with local ones. When you save files, changes are saved to the remote server.

To add a new file to a remote server:

1. Open the file in the editor.
2. Select File > Save Remote Copy.
3. In the Remote Save dialog, double-click on the server, drive, and directory to open the directory where you can to save the file. You can right-click in the directory list to add a new directory.
4. Enter a file name with a recognized Studio extension. Click the File Extensions tab of the Options > Settings dialog (F8) to check the current entries.
5. Click Save.

To add new directories to a remote server:

1. Click the Remote Files tab.
2. Double-click on a server name or right-click and choose Connect.
3. Double-click on drives and directories to locate files.
4. Right-click in the file pane and choose New Folder.
5. Enter a name for the new folder. The folder displays in the current directory.

CHAPTER 19

Creating and Editing Application Pages

This chapter covers the basics of writing and editing pages in ColdFusion Studio, with pointers to features that help you be as productive as possible.

Studio is used by professional Web developers, designers, production and content teams creating sophisticated Web applications, as well as by people creating individual Web pages and sites.

If you're new to creating application pages, see the Quick Start lesson in Chapter 3, "ColdFusion Studio Quick Start," on page 19, which offers a basic guide to building a Web application page.

Contents

Creating Application Pages

Studio gives you control over your tags and content, while adding shortcuts to help you create pages more quickly — as well as advanced features that help teams and savvy Web developers increase their productivity.

New Users

If you're new to ColdFusion Studio, you'll want to begin with the Quick Start lesson, a mini-tutorial that walks you through the steps to create and view dynamic pages. Then you can follow the procedures and examples in this section.

See the *CFML Language Reference* for details on all CFML tags and functions.

Creating new pages

To create a new application page in Studio, you choose File > New to open the New Document dialog box. Here you can choose to

- Create a blank document, with no tag codes.
- Use the default template to create a new file with the basic HTML document tags: DOCTYPE, HTML, HEAD, TITLE, and BODY.
- Choose a Studio wizard that walks you through the steps for creating various types of HTML and CFML pages.

To create a new page:

1. Launch Studio and click the New tool at the top left corner of the Standard toolbar.

   The basic codes for an HTML page appear in the editor window.

   The New tool creates files based on the default template stored at `\Allaire\Studio4\Wizards\HTML\Default Template.htm`. You can customize this file to change the content of your default template.

2. In the TITLE tag, enter a descriptive title in the place of "Untitled."

   The text you enter in the TITLE tag appears in the browser's window title, in a search engine's results list, as well as in a user's bookmarks or favorites list when someone bookmarks your page.

3. Click below the opening BODY tag to position the cursor in the document text area. Now you're ready to enter CFML tags and application page content.

Using the Edit, Design, and Browse views

Studio offers three views for each document:

Edit View is where you enter and edit text, add tags, and code your pages. This is the primary work space in Studio, where you create and edit tags and content. Use the Edit toolbar to close the current document, navigate among open documents, control word wrap, and turn the gutter on and off.

Design View offers you the additional option of visual editing. You can edit the page while seeing how the results would look in a browser. Design view is most useful if you're a new user, if you're designing the major elements that fit on a page, or if you're creating a complex table.

Browse View shows how your page appears in a Web browser. You can view, but not edit, the current page.

In this manual, most procedures and descriptions apply to the Edit view, unless otherwise noted.

Opening existing files

You open existing files using the standard Windows File commands or using the Files pane of the Resources tab in Studio:

- The easiest way to open a file is to double click the file's icon in the File tab in the Resources area.
- You can also open files by pressing CTRL and dragging a file from the Windows Explorer into Studio.
- Choose File > Open or use the Open tool on the Standard toolbar. These commands open the Open dialog box, where you can use the standard Windows File Open choices on the Local pane.

  Use the Remote tab of the File Open dialog box to access files on remote servers.
- To open files from the Web, choose File > Open from the Web.

Tip To open files you've used recently, choose File > Recent Files.

Setting File Open preferences

In the General page of the Settings dialog box, you can choose to always restore the last open file when you start Studio. This feature is particularly useful if you frequently open the same file every time you start Studio.

By default, Studio always opens to the folder you had open when you last closed Studio. But you can change Studio's default start up folder in the File Locations pane of the Settings dialog box (F8).

Using the Edit toolbar

The Edit toolbar offers a command to show all the open files. You can also move from one open file to the first, previous, next, or last open file using the arrows on the Edit toolbar.

Editing Application Pages

Studio is designed to suit many different editing styles. Here are a few of the most common editing tasks:

- Entering tags and content from scratch.
- Adding CFML Expressions
- Adding CFML Queries.
- Importing content from other sources and applying new tags.
- Changing existing tags and attributes.

Each type of task calls for different features and customizations. This section describes the basic editing techniques and then offers pointers to features and settings to help you work in the style that suits you best.

Entering and editing text

Use the tag and command tools on the toolbars and Quick Bar as you edit tags and text.

To add text and tags from scratch:

1. Open a new document and click below the opening <BODY> tag.
2. In the Quick Bar, select a tag tool, such as the Paragraph tool on the Standard toolbar.

   The opening and closing <P> tags appear, with your cursor blinking in between. If Tag completion is turned on, the closing tag appears when you finish typing the opening tag.
3. Enter the tag's contents and press Enter.
4. Right-click in the Edit window, and choose Insert Tag to open the Tag Chooser.
5. Choose a tag and click Select. If you choose a tag that has a tag editor, such as H2, for example, the Tag Editor appears. Edit the tag entries as needed and click OK to insert the tag code in your document.

   If you choose a tag with no attributes, such as CENTER, the tag code appears in the Edit window.

Inserting CFML Expressions

ColdFusion Studio has an Expression Builder that helps you build and add CFML expressions to your ColdFusion application pages. The Expression Builder is a visual tool for choosing and combining functions, operators, and values into CFML Expressions.

To build expressions:

1. Place the cursor at the point in the document where you want to insert the expression.
2. Right-click and select Insert Expression or choose Tools > Expression Builder. You can open and close the list of Expression Elements to show or hide the functions, constants, operators, and variables.
3. In the Functions list, choose an expression type to display the expression elements in the adjoining pane. For example, select Date and Time to see all the ColdFusion functions for manipulating date and time values.
4. Double-click an element to add it to the element list.
5. Add operators by clicking on them in the operator toolbar.
6. Click Insert to add the expression in the current document.

See the Functions and Expressions chapter in the *Advanced ColdFusion Development* book for information about ColdFusion functions and expressions.

Adding links

To add hypertext links:

1. Select the text you'd like to link and click the Anchor tool on the Standard toolbar.
2. In the Anchor dialog box, enter the destination of the link, usually a URL or filename, in the HREF box.

   The selected text becomes a hyperlink.

Adding text from other files

To add and edit text from another file:

1. To add content from another file, choose File > Insert File. Choose a file and click Open.

   The contents of the file are inserted in the current document.
2. Select text you want to edit and use the Edit > Selection commands and editing tools to add tag codes as needed. For example, to add
 tags for line breaks, choose Edit > Selection > Add Line Breaks.
3. Use the Edit menu commands to indent text, or convert tag case as needed. Use the Edit > Selection commands to convert text to lists or to a table.

Editing individual tag blocks

You can use shortcuts to quickly select and edit tag blocks:

- To select an entire CFML tag block — codes and content — press CTRL and double-click.
- To select a range of tag and text, click to place the cursor at the start of the selection, and press SHIFT-CTRL at the end of the selection block.
- To drag and drop HTML from Internet Explorer 4.0, press CTRL while selecting text in Internet Explorer and drag it into Studio.

Using Code Snippets

The Code Snippets feature offers a quick way to store and re-use code. In the Snippets pane of the Resources tab in ColdFusion Studio, you can create and store snippets for your own use, and share code snippets with other developers.

The first procedure creates a code snippet that helps you quickly enclose ColdFusion variables or other parameters in pound signs.

To create and use code snippets:

1. Open the Snippets pane of Resource tab, right-click and choose Create Folder.
2. Provide a name for the snippets folder, for example, *MySnippets*.
3. Right-click in the Snippets pane and choose Add Snippet. The Snippet dialog box appears, with windows for you to enter Start Text and End Text.
4. Enter a name for the code snippet, for example, *PoundSigns*.

   Note that snippet names cannot contain characters that are illegal in file names, such as slashes, special characters, double quotes, etc.
5. In the Start Text window, enter the beginning of your snippet, for example, #.
6. In the End Text window, enter the ending text, for example, #. Click OK
7. Select a variable or expression in your CFML code and double-check the Pound Signs snippet to enclose the selected text in pound signs.

   Right-click on a snippet to edit or delete it.

Shared Snippets

You can share code snippets with other developers on your network.

To create shared snippets:

1. Choose Options > Settings or press F8 to open the Settings dialog box.
2. In the File Locations pane, choose a folder for Shared Snippets.

3. Use the Browse button to navigate to your ColdFusion Studio installation, `\UserData\Snippets` folder and choose a Snippets folder to share.
4. In the Snippets pane of the Resource tab, right-click and choose Create Shared Folder. The folder icon changes color to show it's shared.

Editing Tag Attributes and Values

Studio offers a variety of editing tools that help you enter and edit tags more quickly and accurately:

- **Tag Insight** — Popup menus that appear as you type tag and attribute names. You turn it on and off in the Tag Help pane of the Settings dialog box (F8).
- **The Tag Inspector** — A property sheet-style display, that offers an interactive display of the current tag's attributes and values. Press F4 to open the Tag Inspector for the current tag, or choose the Tag Inspector pane in the Resources area.
- **The Tag Tree** — An expandable tree view at the top of the Tag Inspector. It shows the tag hierarchy of all tags in the current document, a specific selection of tags (only CFML tags or only HTML 4.0 tags), or a customized selection you configure in an outline profile (only Table tags, for example).
- **Tag Chooser** — Selection window for creating and editing tags. Choose Tools > Tag Chooser or right-click in the Edit window and choose Insert Tag.
- **Tag Editors** — Dialog boxes for editing Tag attributes and values. To open a tag editor window, select the tag or click inside it, right-click and choose Edit Tag.

To edit tags and content:

1. Click inside an existing CFML tag, or create a new tag that takes attributes, for example, CFQUERY, and put the cursor before the closing angle bracket (<).
2. Press the space bar. When the Tag Insight is turned on, a popup menu appears showing the valid attributes for the current tag.
3. Choose an attribute, such as Datasource. The attribute appears, with your cursor waiting between quotation marks for you to enter a value.

   Tag Insight is especially useful when you're not sure of the available attributes or don't care to type out long attribute names.
4. You can add additional attributes by pressing the space bar inside a tag.

Using Search and Replace

To find text or phrases in the current document or a set of documents, use the Search menu commands. See Chapter 22, "Maintaining Web Applications," on page 315 for details on search and replace in Studio, including using regular expressions in your search strings.

Using the Tag Inspector and Tag Tree

You can use the Tag Inspector to edit existing attribute values or to add new ones. It is also very useful for editing tags that you create or modify — see *Customizing the Development Environment* for information on creating your own tag editors.

To use the Tag Inspector:

1. Click inside a CFML tag and open the Tag Inspector in the Resources area.
2. Click in an attribute field and enter a value for the attribute.

   When you click outside the current field, the new tag value appears in your code.
3. In the Tag Tree above the Tag Inspector, click on the + signs to expand the tag tree display. Use the Tag Tree to inspect and navigate the document's hierarchy.

Entering special characters

Because angle brackets and special characters are used in HTML coding, to display special characters on a Web page you must use character entities to display special characters on Web pages.

Studio automatically converts special characters to their corresponding character entity. You can turn this feature on and off in the Tag Help pane of the Settings dialog box (F8).

Also, you can use the Special Characters palette to add special characters to your documents. Choose View > Special Characters or select the Special Characters tool on the View toolbar to open this palette at the bottom of your Studio window.

Tag completion

When you're typing HTML and CFML tag codes, you can choose to have Studio "complete" the tag by adding a closing tag when you press the > key at the end of the opening tag.

This feature is most helpful when you are typing HTML and CFML codes and text at the same time, as opposed to editing isolated text and tags.

To turn Tag Completion on and off, use the Tag Completion check box in the Tag Help pane of the Settings dialog box (F8).

You can also refine the way the Tag Completion feature works by clicking the Tag Completion Settings button in the Tag Help pane and editing the list of tags on the tag completion list. You can also choose whether to include ASP tags in tag completion and whether to automatically close double quotes in attribute values.

Running the CodeSweeper to Format Code

CodeSweeper automates the task of getting your code properly formatted. It can be very useful in a number of situations:

- Visually editing page elements in Design view can change code formatting. You can set CodeSweeper to apply its formatting rules when you leave Design view.
- You can painlessly enforce a code style guide for multiple developers by simply having them use the same settings.
- You can easily clean up the code formatting of an existing project as you review the documents.

Click the CodeSweeper button on the Edit toolbar to run the selected CodeSweeper on the current document.

Configuring CodeSweeper

Select Options > CodeSweeper Settings to choose from the list of available CodeSweepers. ColdFusion Studio installs a CodeSweeper optimized for HTML, one that works on both HTML and ColdFusion Markup Language (CFML) tags, and one that you can use for testing. When you have selected a CodeSweeper, you can then specify the formatting rules you want to use.

CodeSweeper settings

You can set the following formatting rules for all tags:

- Set case for tag and attribute names. You can choose all upper or lower case or preserve case, which leaves the case unchanged.
- The Format event names setting contains an additional option — Mixed Case — which applies to case-sensitive JavaScript event handler names like OnMouseOver. If this code is correct in your document, select Preserve Case for this setting.
- Set the use of quotes for values.
- The Trim white space between tags setting cleans up spacing produced by some code generation tools. It is enabled by default. We recommend leaving it on and disabling it for individual tags as needed.

Click the Update button to save these settings.

Tag-specific settings

A wide range of options is available for the individual tags contained in each CodeSweeper. Note that if you set formatting for the "All Other Tags" entry in the tag list, those rules apply to every tag that appears in a document that is not in the list.

- Insert a newline command for start and end tags.
- Set indent by tabs or spaces — a newline is inserted automatically for each indented line when this option is checked.
- Enable indenting of nested sub-tags.
- Override the general trim white space setting by preventing trimming of white space around the selected tag.
- Leave the selected tag unchanged when invoking CodeSweeper.
- Strip tag from document — this is handy for getting rid of superfluous and unwanted tags inserted by code generation tools.

To set rules for a tag:

1. Select a tag from the list.
2. Change the specific settings.
3. Click Update Tag to save the settings for that tag.

To add a tag to a CodeSweeper:

1. Click Add Tag.
2. Enter the tag name and click OK.
3. Change tag settings.
4. Click Update Tag to save the settings.

Check the "Apply Current CodeSweeper when switching from Design view" box to automatically run CodeSweeper on the active document when you leave Design view.

Adding a new CodeSweeper

You can easily create custom CodeSweepers to fit specific document types or coding styles.

To add a new CodeSweeper:

1. Click the Add button.
2. Enter a name for the new CodeSweeper.
3. Change global and tag settings.
4. Click Update to save the settings.

Editing Shortcuts

You can customize the Studio editor to suit your preferences for writing and editing code. For example, you can use color coding to distinguish HTML and CFML code

from text or scripting languages. The following features control the appearance of tags and text in the editor.

The following features and shortcuts can help you work more productively in ColdFusion Studio.

Turning word wrap on and off

To control whether text wraps in the Edit window as you type and enter content, choose Options > Word Wrap or select the Word Wrap icon on the Editor toolbar on the left hand side of the Edit window.

Using the Edit window gutter

You can toggle on and off a gutter on the left hand side of the Edit window. Use the gutter to:

- Keep track of line numbers
- Set breakpoints for the interactive debugger

See the Debugging and Troubleshooting chapter for information on using the interactive debugging tools.

Controlling tag case

You can control whether HTML and CFML tags appear in all upper or lower case. Use the Tags settings in the General pane of the Settings dialog box (F8).

To change the case of all tags in the current document, use Edit > Convert Tag Case command.

Using hexadecimal color values

To specify that all colors be inserted as hexadecimal numbers in Studio, use the Tags settings in the General pane of the Settings dialog box (F8).

Color-coding tag names

Studio displays tag codes, attributes, and values in different colors to help you distinguish tags from content. The document's file type and file extension determine which color scheme is employed.

To edit or change the color scheme for each file extension, use the Color Coding pane of the Settings dialog box (F8).

Indenting code

Indenting your code in the Studio editor greatly increases the readability of the code, even though it does not affect display in the browser (except for preformatted text).

Studio offers a way to automatically indent tags underneath each other. If you select the Auto Indent button in the Edit pane of the Settings dialog box (F8), when you hit the Enter key, Studio aligns your cursor with the indent level of the previous line.

Note that you can override this automatic matching of the previous line's indent level by placing the cursor manually or by using the arrow keys.

Keyboard Shortcuts

Studio offers keyboard shortcuts for many editing commands. You can also customize these shortcuts. Choose Options > Customize and open the Keyboard Shortcuts pane to see the current keyboard mappings and edit them to suit your preferences.

For more information on using keyboard shortcuts, see Chapter 3, "ColdFusion Studio Quick Start," on page 19.

Code Templates

Use the Code Templates feature as a shortcut to entering words that you type frequently. On the Code Templates pane of the Settings dialog box (F8), you can add and change these shortcuts, mapping abbreviations to words or phrases that you type often.

To use the Code Templates feature while editing text, enter the abbreviation, for example, **dt4**, and press CTRL-J. The abbreviation expands to the full phrase, for example, `<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">`.

You can also use the Snippets feature to store frequently-used code blocks, such as navigation bars, and complex table code that you use often, and so on. See Chapter 22, "Maintaining Web Applications," on page 315 for information on creating templates.

Saving CFM files

When you're working on a file with unsaved changes, a mark appears next to the file name on the file tab of the Edit window in ColdFusion Studio. Always save files before adding links or images, to make sure the relative paths of the current document and the source or destination file are resolved correctly.

To save files:

1. Click the Save tool or use the CTRL-S keyboard shortcut.
2. The first time you save a new file in Studio, you are prompted to provide a name for the file. Enter a unique filename.

   Usually, Web filenames appear in URLs that users have to type or bookmark. Keep your filenames short and descriptive. Using all lowercase helps to minimize typing errors and ensures cross-platform portability.
3. To save all the open files that have unsaved changes, choose Save > Save All.

Note To control whether files are saved in UNIX, PC, or Macintosh format, use the File Settings pane of the Settings dialog box (F8).

Previewing Application Pages

As you develop applications in ColdFusion Studio, you can preview dynamic pages against the ColdFusion Server. Studio offers an internal browser, but you can also configure it to launch your browser in a separate window.

Note that in order to preview dynamic pages you must

- Configure external browsers.
- Have a Web server running on a server or workstation, and access to a ColdFusion server.
- Establish mappings that associated the physical directory where a file is stored with a server mapping and a URL. These mappings can be to a local server or to an RDS server.

Remote Development Services server mappings

If you are building pages and want to preview them against an RDS server, you must translate ColdFusion Studio paths to URLs. These URLs are resolved using the following logic:

- If you are previewing a file opened using the RDS remote file access, the URL resolution uses only the mappings for the specific RDS server.
- For all other files (local, mapped drives, UNC\NetworkNeighborhood), the URL is resolved against the cumulative summary of mappings for all RDS servers. When more than a single mapping resolved to a URL, a dialog prompts the user to select which resolution they want to use.
- When a browsed file cannot be resolved to a URL, it is viewed as a local copy.

Whenever URL resolutions are used, you need to save the file for the changes to appear.

For more detailed information on understanding RDS mappings, see the Debugging and Troubleshooting chapter in this book.

Viewing pages in the internal browser

In the Browse pane of the Settings dialog box (F8), you can choose to use Microsoft Internet Explorer 3.01 or higher as the internal browser for ColdFusion Studio.

Note Because of the way Netscape Navigator is built, it cannot be used as the internal browser in ColdFusion Studio.

To preview a page in the internal browser:

1. To see the current open document in Studio's internal browser, click the Browse tab in the editor window.

   The Browse view shows how your code, text, and graphics look in a Web browser.

2. To return to edit mode, choose the Edit tab.

Viewing pages in your external browser

You can choose to view the current document in an external browser. Studio automatically detects which browsers you have installed and lists them in a drop-down menu when you choose the External Browser tool on the View toolbar.

To configure Studio to use your external browser:

1. Choose Options > Configure External Browsers to open the External Browsers dialog box.

   You can add, edit, and delete browser listings, and use the arrow tools to re-order the list of browsers.

2. To add your browser, click Add and in the Browser dialog box, enter its name and the location of the *.exe file. Click OK.

3. Choose whether Studio should browse the page using a temporary copy, (meaning you do not have to save the page first), prompt you to save your changes, or automatically save changes whenever you launch an external browser.

4. Click OK.

   Now when you press the External Browser tool on the View menu, your external browsers are configured according to your preferences.

Visual editing in the Design view

Studio provides an optional visual editing view, called Design View, which lets you edit HTML code while seeing how the results would look in a browser.

The Design view is most useful if you're a new user, if you're designing the major HTML elements that fit on a page, or if you're creating a complex table and want to create the table's form visually without coding the TABLE tags.

Note The Design view feature requires Microsoft's Internet Explorer 4.01 or later. If you don't have this browser installed, or if you don't intend to use the Design view, you can disable the Design view in the Design page of the Settings (F8) dialog box.

To edit a page in Design view:

1. With a document open in the Edit window, save your changes and select the Design tab in the editor. The Design view opens, showing a browser view of the document that you can also use to edit text, styles, and graphics.

2. Use the Design view toolbars to edit text, change styles, add colors, or create HTML tables and form elements.
3. If you've opened Design view and now prefer to cancel any changes, select the Cancel Design View button to discard all changes and return to the Edit view.
4. Use the Edit and Browse tabs to return to the Edit and Browse modes.

Productivity Tips

Studio has dozens of features designed to help you work quickly and efficiently. Here are some tips on how to increase your productivity coding pages in Studio.

Set up the user interface to suit your preferences

You can customize the Studio user interface by adding and moving toolbars and windows. The procedures in this section refer to the default location of toolbars and commands. If you fiddle with the setup and want to return to the original layout, choose Options > Customize, select the Toolbars pane, and click the Reset to Defaults button.

See Chapter 3, "ColdFusion Studio Quick Start," on page 19 for information on configuring Studio's interface to suit your work style.

Manage files in Projects

Use the Projects feature to associate files as a cohesive project in Studio. When you associate files together as a project, you can upload the entire project to your Web site.

See Chapter 20, "Using Projects for Site Management," on page 301 for information on creating projects in Studio.

Use site visualization

You can use the Site Visualization feature to see a graphical representation of the pages and files that make up your site. See Chapter 22, "Maintaining Web Applications," on page 315 for more information.

Use Snippets for frequently-used code

The Snippets feature lets you store reusable code blocks, such as navigation bars, page layout code, and so on. To create a tag snippet, open the Snippets tab in the Resources area, right-click to create a snippets folder, and right-click again to create a tag snippet.

You can have private snippets and shared snippets, accessible using keyboard shortcuts. To activate shared snippets, you need to specify a Shared Snippets folder in the File Locations page of the Settings dialog box.

See the Using Code Snippets section for more information.

Create custom templates

Most Web sites benefit from design templates that set up the basic page and site structure. Any HTML or CFML page can serve as a template in Studio. When you save a file as a template, it is stored in the Studio\Wizards\Custom directory by default.

To build a page from a custom template, choose File > New to open the New File Wizard, and select a template from the Custom page of the New Document dialog box.

For more information on how to create and use templates in Studio, see Chapter 22, “Maintaining Web Applications,” on page 315.

Customize your development environment

Studio offers several ways for you to set up the interface to suit your preferences, to create your own help files, and to add custom buttons, dialogs, and toolbars. For example, you can create your own tag editors for Custom Tags and Extensions.

See *Customizing the Development Environment* for more information.

CHAPTER 20

Using Projects for Site Management

This chapter describes how to use the Cold Fusion project management tools to create, maintain, and use projects to manage application development.

Contents

Why Use Projects?

Organizing your documents and supporting files in a ColdFusion Studio project can significantly increase your productivity in a number of ways:

- Easily keep track of multiple projects in the Projects list.
- Projects provide a quick and easy way to add and remove files on your site.
- You can use Edit, Browse, and Design modes from the Projects tab.
- You can link external files to project files and link files within a project.
- Maintenance chores such as search and replace operations and verifying links can be performed on all the files in a project.
- An entire project can be deployed on the network or to a remote FTP server.
- Projects can be integrated with version source control. You can use source control as part of team development or to keep track of changes to your site content.

The Projects Tab

Projects are created and used entirely through the Projects Resource tab. The Project Resource tab has three panes:

- The top pane lists all of your projects.
- The middle pane shows the project folders.
- The bottom pane lists the files in the selected project. Use this pane to access files in a project.

Project Commands

The Projects menu

Menu commands are active in the Local Files, Remote Files, and Project tabs. You can access them in Edit, Browse, or Design modes.

- Create a new project
- Open a project
- Reopen an existing project
- Close the current project

The Projects Toolbar

The toolbar appears at the top of the Projects Resource tab and contains the following tools:

- Click the New Project tool button to set up a project. By default, projects are saved with the Allaire Projects (.apj) extension.

- Click the Open Project toolbar and navigate to an existing project.

- Click the Deploy Project tool button to upload the project files to a server.

Managing Files in a Project

Projects are created and maintained within your existing directory structure, making it easy to control project contents. This allows you several options:

- Create a new project in an existing directory, optionally including sub-directories.
- Build a new directory structure specifically for the project and work directly in it or add folders and files as the project develops.
- Build a project locally, on the network, or on a remote server.

To create a project:

1. Click Projects > New Project or click the New Project tool button.
2. Enter a project name.
3. Enter a directory path or server in the Location box or click the Browse button and select a location from either the Local or Remote tab.
4. Optionally include sub-folders by clicking the checkbox.
5. Enter file types for the project or select from the dropdown list.
6. Click OK. The project file (.apj) is created in the project root directory.

Note We recommend that you avoid using spaces in project folder and file names. Many servers do not recognize them properly.

To add a file to a project:

1. Save the file to the project directory or move an existing file there.
2. Select a project from the projects drop-down list.

3. Right-click on the project and select Add Files to Project.
4. Select files from the list. You can filter the display in the Add Files dialog by selecting from the Files of Type list.
5. Click OK to add the file. The file pane automatically refreshes.

To remove a file from a project:

1. Select one or more files in the file pane.
2. Right-click and select Remove from Project.

Note Using the Remove from Project command does not delete a file, it just removes the file from the project configuration file.

Working on Project Files

Edit, Browse, and Design modes are all active in the Projects tab.

To open a project file:

- Double-click on a file in the file pane to open it in the editor

To open all project files:

- Right-click in the Folder pane and select Open All Documents in Project

To save project files:

1. Save the file to a project folder.
2. Right-click on the current project folder and select Add Files to Project.
3. Files saved to the current project directory are displayed in the dialog.
4. Select the files you want to add and click OK. The project configuration file is automatically updated.

Previewing Pages in a Project

One of the most convenient features of projects is that you can set a Web server mapping for a project directory. You can then route pages through the Web server for preview.

To define a server mapping for a project:

1. Open the Options > Settings (F8) Browse tab.
2. Click the Enable server mappings checkbox.
3. Enter the local directory path of your project.

4. Enter a complete URL, such as `http://www.myserver/files/`, for the Web server to use for processing HTML pages.
5. Click OK.

Deploying a Project

Since you can create and manage a project locally, on a network drive, or on a remote server, deploying a project really means saving the project files to the host machine. In a development environment that uses multiple servers, for example, development, testing, staging, and production servers, you move the project through these processes using the deployment options.

The project configuration file (.apj) remains on the host machine when project files are deployed.

To deploy a project:

1. Open the project in the Projects tab.
2. Click the Deploy project tool button.
3. Select a location for the project files.

Verifying Links in a Project

Links in project files can be checked in a batch process.

To check the links in project files:

1. Right-click in the project folder pane and select Verify Links.
2. Select options for the files you want check.
3. Run the verification.

See “Verifying Links,” in Chapter 22 for more information on using the link checker.

CHAPTER 21

Adding Source Control for Development Projects

This chapter describes how to use a source control application in Studio to manage your ColdFusion pages and facilitate team development.

Contents

Why Use Source Control?

Version source control for ColdFusion applications is an essential component for coordinating team development of complex projects. A source control system adds a layer of file management responsibility but it offers clear advantages for developers, managers, and support staff, including:

- Share files on a LAN without overwriting work or accidentally modifying files simultaneously.
- Track versions of files and modifications as files are changed.
- Control the deployment of applications.
- Replicate applications to a local workstation for development and testing.

Source control systems really are about control; they are designed to control file management in application development and related work. Common terms such as check in, check out, lock, and unlock accurately describe the security procedures required for an effective source control system.

Note A commitment by network administrators, managers, development teams, and staff to support the system is essential to its success. Collaborative planning and implementation of a source control system ensures that user needs are truly met.

Implementing a Source Control System

There are several possible scenarios for adding source control to your application development work:

- A new installation of both Studio and a source control application
- Adding source control to an existing Studio installation
- Adding Studio to an existing source control system

Note The order of the steps required to use source control for Studio projects is determined by which of these scenarios applies to your development environment, but all of the following steps must be taken:

- Install and configure the source control application.
- Create a Studio project.
- Establish a working directory.
- Create a source control project or add an existing project to source control.
- Add and manage project files under source control.

Standard source control commands

The standard source control commands are available to Studio projects from the main menu and the context menus in the Projects Resource tab:

- Create a project and set its working directory.
- Add and remove files.
- Check-out a file to have exclusive editing rights.
- Check-in a file to make it available to another user.
- Get the latest version of a file without checking it out. You can view the file and edit it but the changes are not saved to the file in source control.
- Open the source control application to set options.

Choosing a Source Control Provider

Studio uses the source-code control (SCC) interface to connect with a wide range of standard source control products including Microsoft Visual SourceSafe, Intersolv PVCS, and StarBase Versions. Studio can work with both client-based and server-based systems.

Studio automatically populates a list of source control applications detected on your system. You can then select the appropriate provider from the list.

The interface and command structure for source control applications varies from vendor to vendor, so check the product documentation of your source control software for specific procedures and options.

Creating a Studio Project for Source Control

A Studio project is a collection of files that make up a development project. The actual physical location of project files and the types of files included in a project are determined by the needs of the development team. Typically, application code, documentation, media, testing materials, and other supporting files are included.

When you add a Studio project to source control for the first time, you are prompted to choose a source control provider based on the source control applications Studio detects on your system.

See Chapter 20, “Using Projects for Site Management,” on page 301 for information on building Studio projects.

Establishing a Working Directory

The *working directory* is the point of interaction for files in a project. If you create a project from an existing directory, that directory is automatically set as the working

directory. You can build a working directory and create a project from it or create a new project first and then build the working directory.

The source control system is the central repository for your files once they are added to it. If you delete files from the working directory, they will not be deleted from source control unless you specifically remove them.

Note Once you set the working directory for a project, you cannot change it. To change the working directory, you have to create a new project and move your files to that project.

Generally, you can create a working directory on your workstation to develop and test your Web applications. The following configuration is required on your system:

- The working directory should be in the root of your Web server directory.
- Your workstation must be running the ColdFusion Application Server (single-user or full version) and a Web server. The Application Server should be configured to run the application that you are developing. This may mean connecting to network ODBC data sources.
- To preview pages in a project, set the Project server mapping so that the working directory is mapped to the URL for your local machine.

When you get or check files out of source control, they go into that project's working directory on your local machine.

Adding a Studio project to source control

All source control work is done in Studio through the project management tools in the Projects Resource tab. Once you have a source control system installed and a working directory configured for a Studio project, you can add that project to source control.

To add a Studio project to source control:

1. Open the context menu in the Project pane of the Project Resource tab.
2. Select Add Project to Source Control.
3. Choose a working directory if the project is not already assigned to a directory. If you build a project from an existing directory, this directory will be the default working directory.
4. Add a new project or select an existing one.

When you have added a project to source control, you can add the project files to the source control database.

To add the files to source control:

1. Click on a folder in the Project Resource tab directory pane to open it.
2. Select the project files you want to add, right-click and select Source Control > Add to Source Control.
3. Add the project.

4. Associate the project with source control.
5. Repeat this process as needed for all directories and files.

Managing Files in Source Control

Although every source control system offers an array of options for managing source files, all systems provide three basic functions for files that are in the system:

- **Get** — Copies the most recent version of a file from source control to the working directory. The default attribute is usually read-only.
- **Check In** — Returns the file to source control. You can set options for handling the files in your working directory, such as removing them or setting them to read-only. These options provide a level of protection if you try to work on a file that is not checked out.
- **Check Out** — Overwrites the copy of the file on your working directory (if you chose to leave a copy there) with a read/write copy from source control. Most source control systems lock files that are checked out so that no one else can open them.

Check in options

Some source control systems provide options for protecting the local copy of a file after it is added or checked in. The system can flag the local file as Read-Only or remove it from the local directory. If a local copy of a file is edited after it is checked in, the changes are overwritten when the file is checked out again. Check your source control system's documentation for available options.

Command options

Studio provides a variety of ways to access source control commands:

- Right-click on a file in the file pane, select Source Control on the context menu and select a command. The context menu contains the complete source control command set.
- Select a file in the file pane, click Tools > Source Control on the main toolbar and select a command.
- Double-click on a file to open the local copy of the file as read-only, check it out, or update the local file with the current version in source control and open it as read-only.

The availability of commands for the selected file is based on the file's current status in the source control system.

Adding files and subdirectories

As your development project progresses, you can add files and subdirectories to source control.

To add a new file to source control:

1. Save the new file in the appropriate sub-directory of the working directory.
2. Add the file to the Studio project.
3. You will be prompted to add the file to source control.

To create a new subdirectory:

1. Add the subdirectory to your working directory.
2. Create a new folder in the correct location in the Studio project.
3. Add a file to the subdirectory and then add the file to the appropriate folder in the Studio directory.
4. Add the file to source control. When you add the file, it automatically creates a new subproject for the new folder in the Studio project.

Synchronizing files

A significant feature of source control for team development is the ability to update your working directory as files are added, changed, and removed from source control by team members. You can run these maintenance procedures based on the level of source control activity of your team.

To synchronize project files with source control:

1. Right-click on a project in the Project Folder pane to open the context menu.
2. Select Synchronize with Source Control. The dialog contains tabs for adding and removing local files based on their status in source control.

To add files from source control to your working directory:

1. Click the Add tab in the Synchronize Projects dialog.
2. The list shows files in source control that are not in your local project.
3. Click the list check box of the files you want to add, click Add, and click Close to update the local project.

To remove local files:

1. Click the Remove tab in the Synchronize Projects dialog.
2. The list shows local files that are not in the source control project.

3. Click the list check box of the files you want to remove, click Remove, and click Close to update the local project.

CHAPTER 22

Maintaining Web Applications

As your Web site develops, you will inevitably need to make changes and test its accuracy, completeness, and efficiency. Studio provides a full set of tools to accomplish these necessary tasks.

Contents

Using Search and Replace

Studio provides two levels of search and replace to help you maintain your Web pages — basic and extended.

Running a basic search

To run a basic search in the current document, choose Search > Find to locate a specific string in the current document or Search > Replace to replace all matches.

- The basic find command (CTRL + F) searches only in the current document.
- Clicking the Find Next button in the Find dialog box sequentially highlights each match in the current document.
- When the search dialog box is closed, you can choose Search > Find Next (F3) to resume the last search, starting where the cursor is positioned.
- Use the Search > Replace command (CTRL+ R) to replace matches selectively or to replace all matches.

Using the extended search and replace feature

For more complex search and replace operations across documents, use the Search > Extended Find command. Or, you can use the Search > Extended Replace command to specify the replacement text as well as the string you're looking for.

The extended search and replace commands offer a number of options to refine your search:

- You can run the Extended Search (CTRL + SHIFT + F) and Extended Replace (CTRL + SHIFT + R) features on the current document, all open documents, or entire folders, including sub-folders.
- When searching through folders, you can restrict searches by selecting file extensions from the File Types drop down list. You can also backup files before making replacements at the folder level.
- Select the Match Case option for case-sensitive searches.
- Select Regular Expressions to enable parsing of regular expression entries. See the section on Searching with Regular Expressions for details on using regular expressions in Studio.
- Select the Skip Tags While Searching option to search the page content only, excluding the tags themselves.

When you choose Replace operations, you are prompted to save all untitled open documents.

The Results pane displays a list of locations where the matched string was replaced. Double-click on a match in the list to highlight it in the document. Right-click in the Results pane to clear the pane or close it.

Note The Extended Replace command skips read-only files.

Replacing special characters

Use the Search > Replace Special Characters command to either replace extended characters with their HTML equivalents, or replace HTML tags with the equivalent extended characters. This command works only in the current document.

Replacing double-spaced lines

Because of the way different operating systems treat carriage returns, text files saved on UNIX or Macintosh systems may become double-spaced when opened in Studio. Use the Search > Replace Double Spacing with Single Spacing command to collapse double-spaced lines to single-spaced lines.

Searching with Regular Expressions

Studio supports searching with *regular expressions* (or *regexes*) to match patterns in character strings in the Extended Find and Replace commands. Regular expressions allow you to specify all the possible variants in a search and to precisely control replacements. Ordinary characters are combined with special characters to define the pattern for the search. The regex parser evaluates the selected files and returns each matching pattern.

In the Find command, the matching pattern is added to the find list. In the Replace operation, it triggers insertion of the replacement string. When replacing a string, it is just as important to ensure what is not found as what is. Simple regular expressions can be concatenated into complex search criteria.

Note The rules listed in this section are for creating regular expressions in Studio. The rules used by other regex parsers may differ.

Special characters

Because special characters are the operators in regular expressions, in order to represent a special character as an ordinary one, you need to precede it with a double backslash (\\)

Single-character regular expressions

This section describes the rules for creating regular expressions. You can use regular expressions in the Search > Extended Find and Replace commands to match complex string patterns.

The following rules govern one-character regexes that match a single character:

- Special characters are: `+ * ? . [ ^ $ ( ) { | \`
- Any character that is not a special character matches itself.
- 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, for example, ".umpty" matches either "Humpty" or "Dumpty."
- A set of characters enclosed in brackets ([]) is a one-character RE that matches any of the characters in that set. For example, "[akm]" matches an "a", "k", or "m".
- Any regular expression can be followed by one of the following suffixes: {m,n} forces a match of m through n (inclusive) occurrences of the preceding regular expression. The suffix {m,} forces a match of at least m occurrences of the preceding regular expression. The syntax {,n} is not allowed.
- A range of characters can be indicated with a dash. For example, "[a-z]" matches any lowercase letter. However, if the first character of the set is the caret (^), the regex 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.
- All regular expressions can be made case insensitive by substituting individual characters with character sets, for example, [Nn][Ii][Cc][Kk].

Character classes

You can specify a character by using one of the POSIX character classes. You enclose the character class name inside two square brackets, as in this example:

```
REReplace("Allaire's Web Site","[[:space:]]","*","ALL")
```

This code replaces all the spaces with *, producing this string:

```
Allaire's*Web*Site
```

The following table shows the POSIX character classes that Studio supports.

Supported Character Classes	
Character Class	**Matches**
alpha	Matches any letter. Same as [A-Za-z].
upper	Matches any upper-case letter. Same as [A-Z].
lower	Matches any lower-case letter. Same as [a-z].
digit	Matches any digit. Same as [0-9].

Supported Character Classes (Continued)	
Character Class	**Matches**
alnum	Matches any alphanumeric character. Same as [A-Za-z0-9].
xdigit	Matches any hexadecimal digit. Same as [0-9A-Fa-f].
space	Matches a tab, new line, vertical tab, form feed, carriage return, or space.
print	Matches any printable character.
punct	Matches any punctuation character, that is, one of ! ‘ # S % & ‘ ( ) * + , - . / : ; < = > ? @ [/] ^ _ { | } ~
graph	Matches any of the characters defined as a printable character except those defined to be part of the *space* character class.
cntrl	Matches any character not part of the character classes [:upper:], [:lower:], [:alpha:], [:digit:], [:punct:], [:graph:], [:print:], or [:xdigit:].

Multi-character regular expressions

You can use the following rules to build a multi-character regular expressions:

- Parentheses group parts of regular expressions together into grouped sub-expressions 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 sub-expressions followed by an asterisk (*) matches zero or more occurrences of the regular expression. For example, [a-z]* matches zero or more lower-case characters.
- A one-character regular expression or grouped sub-expressions followed by a plus (+) matches one or more occurrences of the regular expression. For example, [a-z]+ matches one or more lower-case characters.
- A one-character regular expression or grouped sub-expressions followed by a question mark (?) matches zero or one occurrences of the regular expression. For example, xy?z matches either “xyz” or “xz”.
- 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”.
- Braces ({}) are used to indicate a range of occurrences of a regular expression, 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".

Backreferences

Studio supports backreferencing, which allows you to match text in previously matched sets of parentheses. A slash followed by a digit n (\n) is used to refer to the n^{th} parenthesized sub-expression.

One example of how backreferencing can be used is searching for doubled words -- for example, to find instances of 'the the' or 'is is' in text. The following example shows the syntax you use for backreferencing in regular expressions:

```
("There is is coffee in the the kitchen",
"([A-Za-z]+)[ ]+\1","*","ALL")
```

This code searches for words that are all letters ([A-Za-z]+) followed by one or more spaces [ ]+ followed by the first matched sub-expression in parentheses. The parser detects the two occurrences of *is* as well as the two occurrences of *the* and replaces them with an asterisk, resulting in the following text:

```
There * coffee in * kitchen
```

Anchoring a regular expression to a string

All or part of a regular expression can be anchored to either the beginning or end of the string being searched:

- If a caret (^) is at the beginning of a (sub)expression, the matched string must be at the beginning of the string being searched.
- If a dollar sign ($) is at the end of a (sub)expression, the matched string must be at the end of the string being searched.

Expression examples

The following examples show some regular expressions and describe what they match.

Regular Expression Examples	
Expression	**Description**
`[\?&]value=`	A URL parameter value in a URL.
`[A-Z]:(\\[A-Z0-9_]+)+`	An uppercase DOS/Windows full path that (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.

Regular Expression Examples (Continued)	
Expression	**Description**
`([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.
`(\+|-)?[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}`	Two to four occurrences of 'a': aa, aaa, aaaa.
`(ba){3,}`	At least three 'ba' pairs: bababa, babababa, ...

Resources

An excellent reference on regular expressions is *Mastering Regular Expressions* by Jeffrey E.F. Friedl, published by O'Reilly & Associates, Inc.

Spell Checking

You can check the spelling of your document content and, optionally, code syntax by using either the internal spell checker or the Microsoft Office spell checker, which is enabled if Office 95 or later is detected during installation. Open the Options > Settings (F8) > Spelling tab to set options.

You can download additional dictionaries for international languages and legal and medical terms from the Allaire Web site.

To run the spell checker:

- Click the Spell check tool button or select Tools > Spell Check (F7). You can select commands from the Spell dialog for words not found in the dictionary.

- Click the Mark spelling errors tool button or select Tools > Mark Spelling Errors (SHIFT+F7) to enable spell checking as you type. Right-click in a word and select from the drop-down list of commands.

Validating Code

Before you publish Web pages, it's a good idea to check your code. Studio's integrated validation utility can be used to check and report on HTML syntax errors. It does not correct the errors, but gives you a list of errors and comments. Double-click on an error message to highlight it in the document.

Open the Options > Settings (F8) Validation tab to select the validation levels for the current document.

To run the validator:

- Click the Validate current document tool button or select Tools > Validate Document (SHIFT + F6). The Validation Results pane displays either a "No errors or warnings" message or lists the syntax errors it found.
- Choose Tools > Validate Current Tag (F6) to check the selected tag.

Verifying Links

Due to the transitory nature of Web pages, you'll undoubtedly find that some of the sites you link to will move or be removed during the lifetime of your site. Tracking down these "missing links" can be time-consuming, but Studio makes the job easier by letting you know which links might be bad.

The Link Checker can verify the location of documents on Web sites, local HTML files, and dependencies for graphics and other media files. Links to secure pages (HTTPS) FTP links, and mailto links cannot be verified. By default, all the links will be checked, but you can un-check as many links as you like to skip verifying them. Tags in comments are not calculated.

To verify links in the current document:

1. Click the Verify Links tool button or select Tools > Verify Links. The Links Results pane lists the following:
   - Source — The current document name proceeded by an icon identifying the link as a document or media file.
   - Link — The link as referenced in the document.
   - Full URL — The path or IP address of the link.
   - Status — Initially set to "Untested."

2. Click the Start Link Verification button on the Link toolbar. Each link is checked in order. The Status column displays OK for successful links and returns an appropriate message for failed links.

3. The Stop button is enabled during the link test. Click it if you need to end the test before it completes.

Link verification options:

- Select a link in the list and click the Set Full URL tool button to change the URL or the local directory Studio should use to process the relative link. Entries made in this dialog are stored in the drop-drown list for future use. Setting the root URL does not modify the source document, it is simply used to tell Studio how to test relative links.

- Click the Set Timeout tool button to enter a time value after which the tester moves to the next link in the list.

- Click the Set Proxy tool button and enter a server name and port number if the link is routed through a proxy server.

- Click the Print button to produce a report of failed links. The report displays in your default browser. You can publish the report, email it, or print it from the browser.
- Right-click on a link in the list to select additional options.

Testing Page Download Times

Page download times are a frequently-cited metric of a site's design and effectiveness. If you are developing a public site, you need to be sensitive to achieving a balance between attractive content and tolerable page-loading times. Studio can help you find that balance by supplying real values for document download times across a range of modem speeds.

To test the current document's download time:

1. Select Tools > Document Weight. The file's dependencies are listed along with file size and download times for a range of modem speeds. Tags in comments are not calculated.
2. If the page's download time exceeds the site's requirements, edit the page to decrease the number or size of dependencies, and re-test.

Studio uses the Root URL setting in your FTP configuration to determine the relative path to files.

To set the root URL for an FTP server:

1. Click the Remote Files tab in the Resources view.
2. Right-click on a server name and select Server Properties.
3. Complete the Root URL field.

CHAPTER 23

Customizing the Development Environment

If you have used ColdFusion Studio for a while, you have probably taken advantage of its flexible interface to position toolbars, set tag color-coding, define validation levels, and a host of other options.

You can take this open interface several steps further by creating your own tag editors. Many CFML custom tag developers build tag editors to distribute with their tags. These can be identified in Allaire's Tag Gallery (http://www.allaire.com/developer/taggallery/) by the <VTM> marker next to the tag name. You can also modify existing tag editors to suit your needs.

If you publish Web pages that require a lot of user input or if you develop ColdFusion applications, you can build custom wizards to gather user input.

Contents

The Visual Tool Markup Language (VTML)

VTML is a family of markup languages used to extend the Integrated Development Environment (IDE) of HomeSite and ColdFusion Studio. Using VTML tags you can modify, as well as, add various elements of the IDE.

VTML can be used to customize:

- Tag Inspector
- Tag Tips
- Tag Insight
- Tag Editing Dialogs
- Tag Outline Profiles
- Wizards
- Tag Chooser elements
- Expression Builder

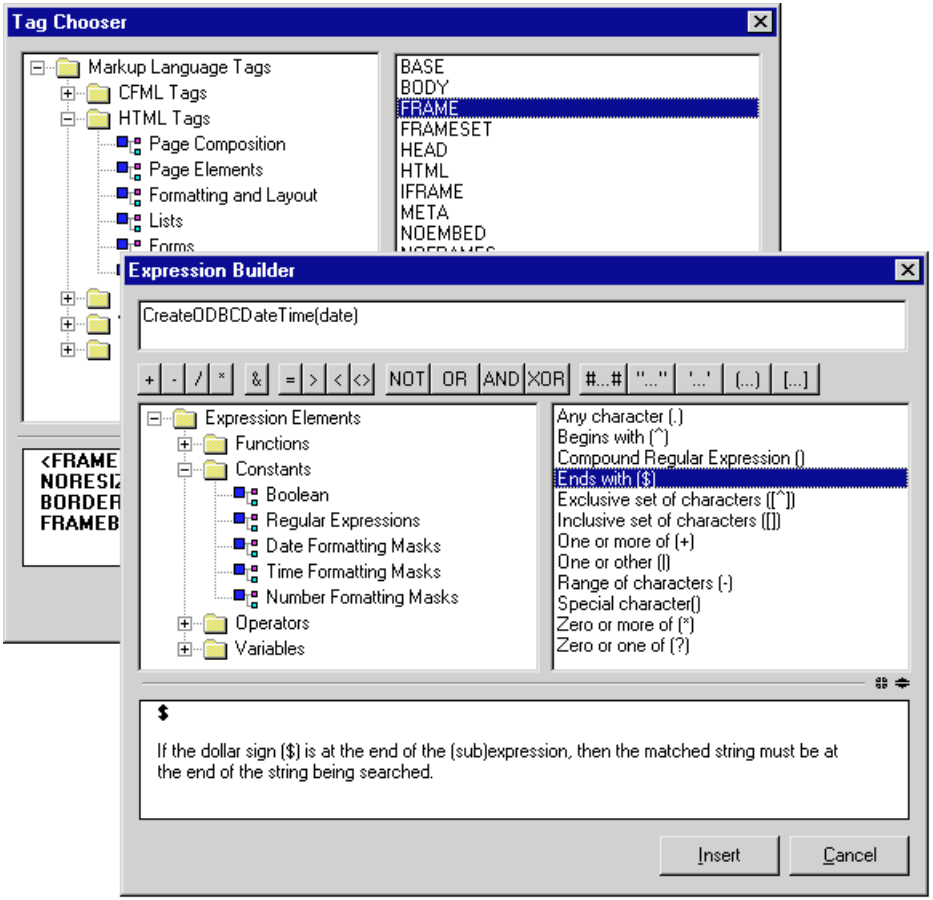
Customizing Tag Chooser and Expression Builder

The Tag Chooser and Expression Builder are important tools of a web developer. The Tag Chooser is one of the primary interfaces to tag selection in ColdFusion Studio. The Expression Builder is used to compose complex expressions in ColdFusion Studio.

Customization objective

The Tag Chooser dialog was designed to adapt flexibly to the new requirements imposed by the constant evolution of HTML and CFML, as well as to accommodate the emergence of new tag-based languages. In a similar way, the Expression Builder provides the user with a hierarchical view of various expression elements that grow with the evolution of ColdFusion, as well as other web technologies.

With the above in mind, both of the dialogs were designed to be fully customizable using VTML. The content, as well as the behavior of the dialogs is controlled by VTM templates: `MarkupTags.VTM` and `ExpressionElements.VTM` in the `\Templates\` directory. Two VTML tags, CAT and E let you customize the content of these dialogs.



Dialog Definition Files

The structure of the `markuptags.vtm` and `expressionelements.vtm` files is very simple. They both contain a set of category and element tags. Category tags can contain any number of elements or other nested category tags.

```
<CAT … main category>

<CAT … sub-category No.1>
            <E … >
            <E … >
            <E … >
    </CAT>
    <CAT … sub-category No.2>
```

```
            <E ... >
            <E ... >
    </CAT>

</CAT>
```

Every category represents a category in the category tree located on the left-hand side of the dialogs. The elements in turn represent the tags or expression elements included under each category. The following section explains how to create and update category and element tags.

Category tag

The CAT tag is used to define a category in the category tree. The following attributes can be used in the category tags.

Category Tag <CAT ... >	
CAPTION	The caption of the category.
DESC	The contents of the HTML help for the category. Notice that the HELPFILE attribute can be used to specify the help as a separate file.
HELPFILE	The relative path to the HTML help for the category. (`HELPFILE = "Docs/MyTag.htm"`)
ICON	"Folder"|"Elements"|RelativeFilePath. Defines the icon used for the category. You can use a pre-defined Folder or Elements category. In addition, a relative file path can be provided to a custom BMP image (ICON ="images/custom.bmp"). By default, the Folder icon is displayed.
EXPANDED	YES\NO Indicates that the category tree-item should be expanded the first time the dialog is displayed. By default this value is set to NO.
SHOWSUBELEMENTS	YES\NO. Indicates that when selected the elements of its sub-categories will also be displayed on the right-hand side. For example, when the "HTML Tags" category is selected the right hand side displays all the tags included in all the HTML tag sub-categories. By default, this value is set to YES.

Element tag

The E tag is used to define elements within a category. These elements are either Tags in the Tag Chooser or Expression Elements in the Expression Builder.

The following attributes define the behavior of an element:

Element Tag <E ... >	
CAPTION	The caption of the element.
VALUE	Tag Chooser — This value represents the tag string pasted when the tag is selected. If a visual editor exists for the tag, an incomplete tag string should be used to invoke the editor. For example, `VALUE="MATED"` would invoke the visual tag editor for MYTAG stored in `mytag.vtm`. Expression Builder – This value represents the syntax of the expression element to be pasted into the expression textbox.
DESC	The contents of the HTML help for the element. Notice that the HELPFILE attribute can be used to specify the help as a separate file.
HELPFILE	The relative path to the HTML help for the element. (`HELPFILE = "Docs/MyTag.htm"`)

Creating Tag Definitions

Various features of the development environment require the knowledge about a specific tag to operate properly. For instance the Tag Inspector feature cannot function properly unless it knows the attributes of the tag being entered. In addition. it needs to know the type of each tag attribute, and in special instances, even the enumerated values for a specific attribute. The Tag Inspector uses a set of tag definitions to learn about the specifics of various tags.

The tag definitions are stored in `\Extensions\TagDefs\` under the installation directory. VTML is used to create tag definitions. For instance, all the information about the APPLET tag is stored in `\Extensions\TagDefs\HTML\Applet.vtm`. The definition files are organized in language directories due to name conflicts between various markup languages. Users can customize existing tag definitions, as well as create new tag definition files.

The following features use the tag definition files:

- Tag Inspector
- Tag Tips
- Tag Insight
- Tag Editing Dialogs

Creating a tag definition file

Every tag editor file contains the following markup structure:

```
<TAG>

    <ATTRIBUTES>

    … Defines tag attribute properties and behavior

    </ATTRIBUTES>

    <ATTRIBCATEGORIES>

    … Defines logical grouping for tag attributes
    </ATTRIBCATEGORIES>

    <EDITORLAYOUT>

    … Defines the layout of a tag editor

    </EDITORLAYOUT>

    <TAGLAYOUT>

    … Defines the tag generation template
    </TAGLAYOUT>

    <TAGDESCRIPTION>

    … HTML-based documentation for the tag

    </TAGDESCRIPTION>

</TAG>
```

You can create the definition file in three ways:

- Write it manually.
- Create it from an existing tag definition file (a good way to learn).
- Use the Tag Definitions Library dialog to add a tag and edit the generated file.

Defining attributes

The ATTRIBUTES block defines attributes inside the main TAG block. The ATTRIBUTES block can only contain ATTRIB tags. The following example demonstrates the definition of four tag attributes: VALUE, TITLE, ALT and ALIGN.

```
<ATTRIBUTES>
<ATTRIB NAME="VALUE">
<ATTRIB NAME="TITLE">
<ATTRIB NAME="ALT">
<ATTRIB NAME="ALIGN">
</ATTRIBUTES>
```

In most cases the features such as Tag Insight require to know more than just the names of the attribute. You can use the ATTRIB tag to define:

- Attribute value types (e.g., color, file path)
- Enumerated values (e.g., LEFT, RIGHT, TOP, BOTTOM for the ALIGN attribute)

Defining attribute value types

The value type for a specific attribute can be specified using the TYPE attribute in the ATTRIB tag.

```
<ATTRIBUTES>
<ATTRIB NAME="VALUE"    TYPE="text" />
<ATTRIB NAME="BGCOLOR"  TYPE="color"/>
<ATTRIB NAME="FONTFACE" TYPE="font" />
</ATTRIBUTES>
```

The possible value types are:

- **TEXT** — free text content
- **ENUMERATED** — a list of enumerated values
- **COLOR** — a color value (name or hex)
- **FONT** — font name or font family
- **FILEPATH** — a full file path
- **DIRECTORY** — a directory path
- **FILENAME** — file name only
- **RELATIVEPATH** — a relative representation of the path
- **FLAG** — an ON\OFFattribute containing no value

The following value types are available in ColdFusion Studio ONLY:

- **QUERYNAME** — a ColdFusion record set name
- **EXPRESSION** — a ColdFusion expression

Defining enumerated values

Enumerated values can be specified for attributes of TYPE="Enumerated." A subtag ATTRIBOPTION is used to specify such values:

```
<ATTRIB NAME="CHARSET" TYPE="ENUMERATED">
    <ATTRIBOPTION VALUE="iso-8859-1" CAPTION="Western" />
    <ATTRIBOPTION VALUE="iso-8859-2" CAPTION="Central European (ISO)" />
    <ATTRIBOPTION VALUE="iso-8859-8" CAPTION="Hebrew (ISO-Visual)" />
</ATTRIB>
```

The CAPTION attribute specifies the form in which the option appears in the drop-down lists while the VALUE attribute specifies the underlying value used by the attribute. The CAPTION attribute is optional.

ATTRIB Tag Attributes	
NAME	Name of the attribute.
TYPE	(Optional) Type of the value expected for the attribute: • TEXT - free text content • ENUMERATED - a list of enumerated values • COLOR - a color value (name or hex) • FONT - font name or font family • FILEPATH - a full file path • DIRECTORY - a directory path • FILENAME - file name only • RELATIVEPATH - a relative representation of the path • FLAG - an ON\OFF attribute containing no value Special Types Available in ColdFusion Studio ONLY: • QUERYNAME - a ColdFusion record set name • EXPRESSION - a ColdFusion expression
CACHEFAMILY	(Optional) The name of the cache family associated with the attribute.
CONTROL	(Optional) Use to populate the attribute value to a specific tag editor control when editing existing tags.

Use the following attributes to specify enumerated values:

ATTRIBOPTION Tag Attributes	
VALUE	The value of the enumeration option.
CAPTION	(Optional) The visual representation of the option if different from VALUE.

Defining attribute categories

Use the ATTRIBCATEGORIES section to define attribute categories. The categories are used to organize the attributes when viewed in the Tag Inspector. The ATTRIBCATEGORIES block can only contain ATTRIBGROUP tags. The following example demonstrates the definition of four categories.

```
<ATTRIBCATEGORIES>
    <ATTRIBGROUP NAME="Appearance"
        ELEMENTS="BACKGROUND,BGPROPERTIES,LEFTMARGIN,TOPMARGIN"/>
    <ATTRIBGROUP NAME="Colors"
        ELEMENTS="BGCOLOR,VLINK,ALINK,LINK,TEXT"/>
    <ATTRIBGROUP NAME="Misc"
        ELEMENTS="GIZMO"/>
</ATTRIBCATEGORIES>
```

Use the following attributes to specify the category attributes:

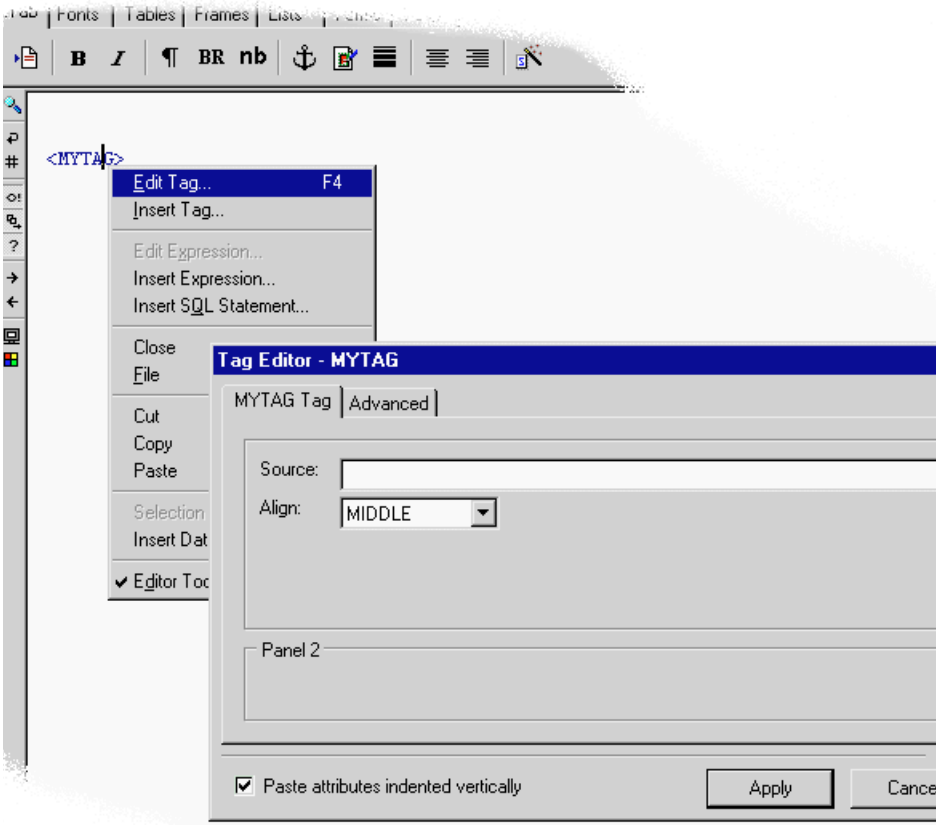
ATTRIBGROUP Tag Attributes	
NAME	The name of the category.
ELEMENTS	The list of attributes included in the category.

Building Tag Editors

The definition of a Tag Editor is composed of three tasks:

- Layout of dialog controls
- Definition of how to populate controls with tag attributes
- Definition of the tag generation template

Have a look at the editor file `mytag.vtm` to see how this works. The following section will explain how to compose the editor file.



Defining controls

This section contains only two kinds of tags: CONTROL and CONTAINER tags. CONTROL and CONTAINER tags represent graphical controls. The tags are identical in definition, with the exception that only CONTAINER tags have the capability to contain other CONTROL tags.

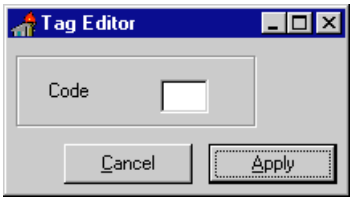
A Panel control is a good example of a control that can be a CONTAINER containing other CONTROLS, such as Labels or TextBoxes.

Look at the following example:

```
<TAG>
    <EDITORLAYOUT HEIGHT=50 WIDTH=200>
        <CONTAINER  NAME="Panel1" TYPE="Panel" WIDTH=150 HEIGHT=50>
            <CONTROL NAME="lblCode" TYPE="label" CAPTION="Code" DOWN=20
                RIGHT=20 WIDTH=70/>
```

```
            <CONTROL NAME="txtCode" TYPE="TextBox" ANCHOR="lblCode"
            <CORNER="NE" WIDTH="30"/>
        </CONTAINER>
    </EDITORLAYOUT>
</TAG>
```

You can name the above template MYTAG.VTM and test it by attempting to edit an empty MYTAG tag. The tag editor dialog would look like this:



The example displays a single Panel CONTAINER containing Label and TextBox CONTROLs. Only a few controls can be containers:

- **TabDialog** — A tab dialog control capable of containing TabPage container controls.
- **TabPage** — Only used inside a TabDialog container control.
- **Panel** — A general purpose panel container control. Can contain any regular or container controls.

You may have already noticed that the control represented by a CONTROL or a CONTAINER tag is defined by the TYPE attribute. We can also see that WIDTH and HEIGHT attributes determine the size.

The ANCHOR and CORNER attributes determine the point relative to which the control is positioned. The ANCHOR can be specified as the name of any control that was already laid down. The corner specifies the corner of the anchor control to be used for positioning. The possible choices are NE, NW, SE, and SW. The DOWN and RIGHT attributes then specify the pixel offset from the anchor control.

The following attributes are available for the Control and Container tags:

CONTROL\CONTAINER attributes	
TYPE	Type of the control/container. Available control types are: • Label • TextBox • CheckBox • RadioGroup • DropDown • TextArea • FontPicker • ColorPicker • Image • FileBrowser • SQLTextArea • ActiveX Available container types are: • Panel • TabDialog • TabPage
NAME	Name of the control.
ANCHOR	(Optional) The name of the control relative to which control is positioned. If omitted, control is positioned relative to the upper left corner of its container.
CORNER	(Optional) Corner of the ANCHOR control relative to which offset position is calculated (NW, NE, SW, SE). The default is NW, the upper-left corner.
DOWN	Offset in pixels down from the anchor point (corner).
RIGHT	Offset in pixels right from the anchor point (corner).

CONTROL\CONTAINER attributes (Continued)	
WIDTH	Width of the control. Can be specified in three ways: • In pixels (e.g., WIDTH=200). • As the name of another control (e.g., WIDTH="SomeControl"). In this case the width of the specified control is used. The control must be already positioned. • Set to maximum (WIDTH="MAXIMUM"). Stretches control to the right boundary of its container.
HEIGHT	Height of the control. Can be specified in three ways: • In pixels (e.g., HEIGHT =200). • As the name of another control (e.g., HEIGHT ="SomeControl"). In this case the height of the specified control is used. The control must be already positioned. • Set to maximum (HEIGHT ="MAXIMUM"). Stretches control to the bottom boundary of its container.
LFWIDTH	Width of the control used when user's system is set to use large fonts.
LFHEIGHT	Height of the control used when user's system is set to use large fonts.
MAXWIDTHPADDING	Used with WIDTH=MAXIMUM. Specifies the padding in pixels from the container's right boundary.
MAXHEIGHTPADDING	Used with HEIGHT=MAXIMUM. Specifies the padding in pixels from the container's bottom boundary.

Populating dialogs with tag data

Once the layout of controls is completed one needs to define the way in which the editor controls are populated when you are editing an existing tag. This is done in the ATTRIBUTES block of the main tag editor template.

The ATTRIBUTES block can only contain ATTRIB tags. The ATTRIB tag defines the way tag attribute values get filled into the dialog controls.

```
<ATTRIBUTES>
    <ATTRIB NAME="VALUE" CONTROL="txtName"/>
    <ATTRIB NAME="TITLE" CONTROL="txtTitle"/>
    <ATTRIB NAME="TITLE" CONTROL="txtTitle2"/>
    <ATTRIB NAME="ALT" CONTROL="txtAltText"/>
    <ATTRIB NAME="ALIGN" CONTROL="dropAlign"/>
</ATTRIBUTES>
```

The name attribute of the ATTRIB tag specifies the name of the attribute, while CONTROL specifies which control the value of that attribute should be assigned to. Notice that you can have multiple ATTRIB tags with the same NAME. This is common for more complex tag editor dialogs where a single attribute value may have to be filled into multiple controls.

Special $$TAGBODY attribute name

There is one special tag attribute name ($$TAGBODY) which is used when a control needs to be populated by the body of a tag. An example of such a tag editor is the editor for the HTML tag TEXTAREA. The body of the TEXTAREA tag is filled into the txtTextAreaContent control using the following ATTRIB tag.

```
<ATTRIB NAME="$$TAGBODY" CONTROL="txtTextAreaContent"/>
```

Generating the tag

The final stage in the process of building a tag editor is the definition of how a tag gets generated from the date entered into the editor controls. The tag generation logic is stored in the TAGLAYOUT block. This block contains a short template used to generate the final tag string. The markup language used the TAGLAYOUT template that was originally designed for wizards and was therefore called Wizard markup language (WIZML). Because of this, all the tags in this section begin with the WIZ prefix.

Consult the section for more info on WIZML. A good way to get started is to have a look at the TAGLAYOUT sections of existing HTML tag editors located in the `\Extensions\TagDefs\HTML` directory in the main installation directory.

Variables passed to the layout template

WIZML is described in a separate section though a few things should be noticed. WIZML supports variables as well as functions. The value of each control of the tag editor is passed to the template using a variable with the same name. Therefore a ColorPicker named colorBGColor will pass its value in colorBGColor variable. The TAGLAYOUT template can then use this data to generate the tag string.

```
<TAGLAYOUT>
    <MYTAG COLOR="$${colorBGColor}">
</TAGLAYOUT>
```

The above example shows a simple layout template for a hypothetical tag with a single attribute COLOR. Notice that in WIZML variables are embedded using the $${}

delimiters. If the user chose *White* in the *colorBGColor* ColorPicker, the above template would generate the following tag:

```
<MYTAG COLOR="White">
```

Special variables

In addition to the CONTROL variables, a few other parameters get sent to the tag layout template:

- **OPTIONLowerCaseTags** — returns 'true' or 'false'. Specifies whether the tag should be generated using lowercase.
- **OPTIONLinearLayout** — Returns 'true' or 'false'. Specifies whether the tag should be generated with its attributes in a single line or indented vertically.
- **TAGDATAUnknownAttributes** — A string containing all attributes which were contained in the edited tag string but are not recognized by the editor.

Using OPTIONLowerCaseTags

This parameter can be used to create a layout template, which generates a tag in lower or upper case based on user preferences. Here is a version of the MAYTAG layout template responding to case preferences:

```
<TAGLAYOUT>

    <WIZIF OPTIONLowerCaseTags EQ 'true'>
        <mytag color="$${colorBGColor}">
    <WIZELSE>
        <MYTAG COLOR="$${colorBGColor}">
    </WIZIF>

</TAGLAYOUT>
```

Using OPTIONLinearLayout

Some people like their tag attributes in a single line while others like them indented. Here is a version of the MAYTAG layout template responding to such preferences:

```
<TAGLAYOUT>

    <WIZIF OPTIONLinearLayout EQ 'true'>
        <WIZSET Spacer =  ' ' >
    <WIZELSE>
        <WIZSET Spacer =  Chr(13) & Chr(10) & '       ' >
    </WIZIF>

    <MYTAG COLOR="$${clrBGColor}"$${Spacer}FACE="$${fontFace}
        "$${Spacer}SIZE="$${txtSize}">

</TAGLAYOUT>
```

The template would generate a tag based on the following layout preference:

```
LINEAR:

<MYTAG COLOR="White" FACE="Arial" SIZE="10">

NONLINEAR:

<MYTAG COLOR="White"
    FACE="Arial"
    SIZE="10">
```

Using TAGDATAUnknownAttributes

The TagDataUnknownAttributes tag contains the list of attributes that are contained in the original tag string but are not supported by the editor. For example, you may write an editor for the HTML tag INPUT. You may provide editing capabilities for all basic attributes, however the editor will not cover JavaScript event attributes (for example, onCLick= …, etc.) In order not to lose these "unknown" attributes during the editing process, the editor engine creates the TAGDATAUnknownAttributes variable containing a list of unknown attributes together with their original values. You can use this variable to 'stamp' all the unsupported attributes at the end of the tag you are generating.

```
<TAGLAYOUT>
    <MYTAG COLOR="$${colorBGColor}"
    <WIZIF TAGDATAUnknownAttributes  NEQ
    ""> $${TAGDATAUnknownAttributes}</WIZIF>>
</TAGLAYOUT>
```

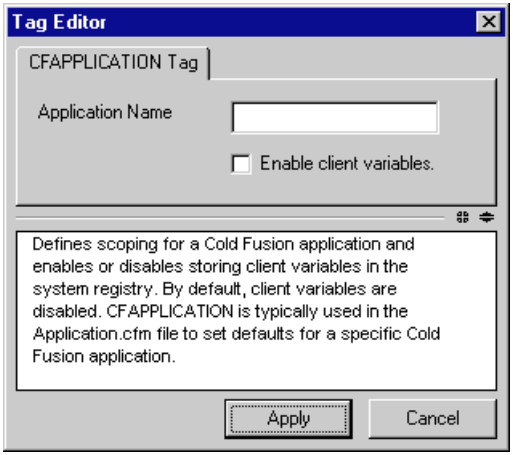
If we edited a tag <MYTAG COLOR="Blue" onClick="CallThis">, the above template would preserve the onClick attribute despite the fact that it is not supported in the editor. The editor is also intelligent enough to list the unknown attributes in a linear or indented format based on a user's layout preferences, as described in the previous section.

Adding Tag Help

You can associate an HTML-based help document with a tag by simply embedding the help text inside the TAGDESCRIPTION block.

Here's an example of a tag description block and the result in a Tag Editor:

```
<TAGDESCRIPTION HEIGHT=100>

    <B>CFAPPLICATION</B>
        <P>Defines scoping for a ColdFusion application and
        enables or disables storing client variables in the system
        registry. By default, client variables are disabled.
        CFAPPLICATION is typically used in the Application.cfm
        file to set defaults for a specific ColdFusion application.

</TAGDESCRIPTION>
```



Providing help from an external file

As the help content grows, it may become cumbersome to specify the entire body of the help inside the TagDescription block. In addition, large bodies of help embedded in the editor file will cause the editor dialog to open more slowly as more markup has to be parsed to compose the editor. Under these circumstances, it advisable to provide large help contents in a separate HTML file. Such files can then be referenced using a relative path from the tag editor template.

For example:

```
<TAGDESCRIPTION HELPFILE="Docs/TagHelpFile.htm"/>
```

VTML Container/Control Reference

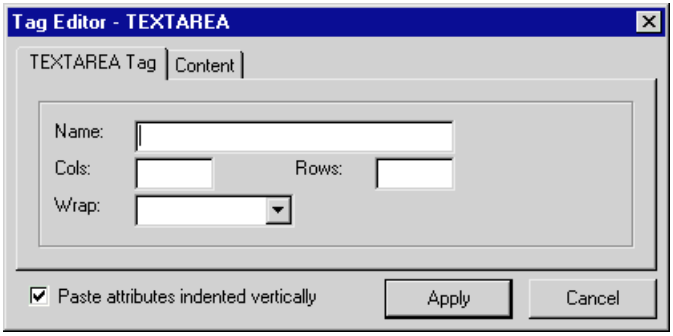
This section contains the complete syntax, with examples, for VTML containers and controls.

TabDialog container

<CONTAINER TYPE="TabDialog" ...
TabDialog control is a special container control, because it can only contain <CONTAINER TYPE="TabPage"> tags. This is natural because one needs to specify the tab pages before embedding more controls on the tab dialog itself.

TabDialog example:

```
<CONTAINER NAME="MainTabDialog" TYPE="TabDialog" WIDTH=MAXIMUM
    HEIGHT=MAXIMUM>

    <CONTAINER  NAME="TabPage1" TYPE="TabPage" CAPTION="TEXTAREA Tag">

        ... embedded controls

    </CONTAINER>

    <CONTAINER  NAME="TabPage2" TYPE="TabPage" CAPTION="Content">

        ... embedded controls

    </CONTAINER>

</CONTAINER>
```

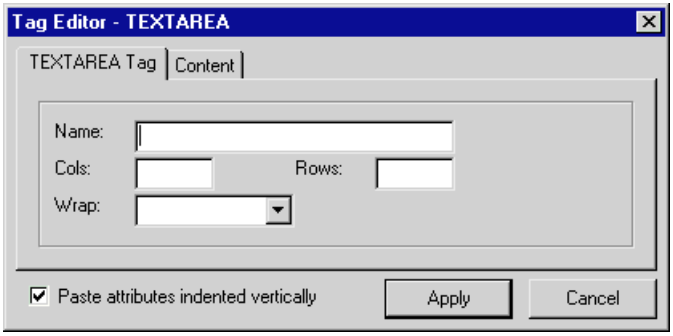


TabPage container

<CONTAINER TYPE="TabPage" ...	
TabPage control is also special because it can only be contained inside a TabDialog CONTAINER control.	
CAPTION	Caption displayed on the top of the tab.

TabPage example:

```
<CONTAINER NAME="MainTabDialog" TYPE="TabDialog" WIDTH=MAXIMUM
    HEIGHT=MAXIMUM>

    <CONTAINER  NAME="TabPage1" TYPE="TabPage" CAPTION="TEXTAREA Tag">

        ... embedded controls

    </CONTAINER>

    <CONTAINER  NAME="TabPage2" TYPE="TabPage" CAPTION="Content">

        ... embedded controls

    </CONTAINER>

</CONTAINER>
```



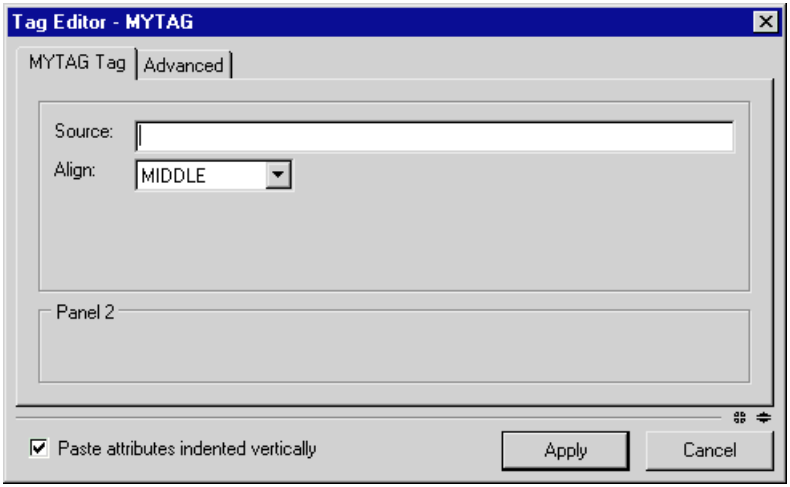
Panel container

<CONTAINER TYPE="Panel" ...	
Panel is the most common container control. Panel can contain any control or container except TabPage, which is restricted to TabDialog.	
CAPTION	Caption displayed in the upper left corner of the panel boundary.

Panel example:

```
<EDITORLAYOUT HEIGHT=225>
```

```
        <CONTAINER  NAME="MainTabDialog" TYPE="TabDialog" WIDTH=MAXIMUM
            HEIGHT=MAXIMUM>

            <CONTAINER  NAME="TabPage1" TYPE="TabPage" CAPTION="MYTAG Tag">

            <CONTAINER  NAME="Panel1" TYPE="Panel" DOWN=5 RIGHT=10
                WIDTH="MAXIMUM" HEIGHT=125>

                <CONTROL NAME="lblSource" TYPE="Label" CAPTION="Source:"
                    DOWN=17 RIGHT=10 WIDTH=50/>

                <CONTROL NAME="txtSource" TYPE="TextBox" ANCHOR="lblSource"
                    CORNER="NE" WIDTH="MAXIMUM"/>

                <CONTROL NAME="lblAlign" TYPE="Label" CAPTION="Align:"
                    ANCHOR="lblSource" CORNER="SW" DOWN=11 WIDTH=50/>

                <CONTROL NAME="dropAlign">TYPE="DropDown" ANCHOR="lblAlign"
                    CORNER="NE" WIDTH=100>
                    <ITEM VALUE="TOP" CAPTION="TOP" />
                    <ITEM VALUE="MIDDLE" CAPTION="MIDDLE" SELECTED/>
                    <ITEM VALUE="BOTTOM" CAPTION="BOTTOM" />
                </CONTROL>

            </CONTAINER>

            <CONTAINER  NAME="Panel2" TYPE="Panel" CAPTION=" Panel 2 "
                ANCHOR="Panel1" CORNER="SW" DOWN=5 WIDTH="MAXIMUM"
                    HEIGHT=MAXIMUM
            </CONTAINER>

            </CONTAINER>

            <CONTAINER  NAME="Advanced" TYPE="TabPage" CAPTION="Advanced">

            </CONTAINER>

        </CONTAINER>

    </EDITORLAYOUT>
```

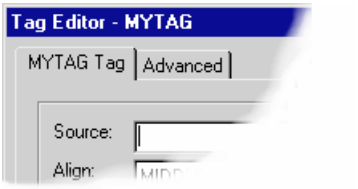


Label control

<CONTROL TYPE="Label" ...	
Panel is the most common container control. Panel can contain any control or container except TabPage, which is restricted to TabDialog.	
CAPTION	The text displayed by the label.
AUTOSIZE	YES\NO. Automatically sizes the control to the text it contains. This option is overridden if WIDTH or HEIGHT are explicitly specified.
TRANSPARENT	YES\NO. Makes label transparent.
ALIGN	LEFT\CENTER\RIGHT. Specifies the horizontal alignment of text in the label.
VALIGN	TOP\CENTER\BOTTOM. Specifies the vertical alignment of text in the label.

Label example:

```
<CONTROL NAME="lblSource" TYPE="Label" CAPTION="Source:" DOWN=17 RIGHT=10
WIDTH=50/>
```



TextBox control

<CONTROL TYPE="TextBox" ...	
A simple textbox control.	
VALUE	The text displayed by the textbox.
AUTOSIZE	YES\NO. Automatically sizes the control to the text it contains. This option is overridden if WIDTH or HEIGHT are explicitly specified.
EDITABLE	YES\NO. Enables or disables editing.
AUTOSELECT	YES\NO. Decides whether contents get highlighted when the cursor enters the textbox.
MAXLENGTH	Limits the amount of text in the textbox to a specific number of characters.
PASSWORDCHAR	A character to be used to mask entered text. You can create a simple password box using PASSWORDCHAR="*".
CHARCASE	NORMAL\UPPER\LOWER. Specifies whether entered text is automatically uppercased of lowercased. The default is NORMAL, preserving entered case.
VALIGN	TOP\CENTER\BOTTOM. Vertical alignment of text in the label.

TextBox example

```
<CONTROL NAME="lblSource" TYPE="Label" CAPTION="Source:" DOWN=17 RIGHT=10
    WIDTH=50/>
<CONTROL NAME="txtSource" TYPE="TextBox" VALUE="Some Value"
    ANCHOR="lblSource" CORNER="NE" WIDTH="MAXIMUM"/>
```

DropDown control

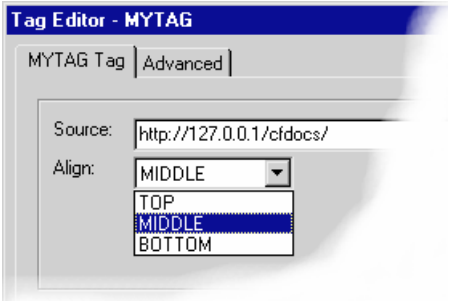
<CONTROL TYPE="DropDown" ...	

A drop-down listbox. This tag requires <ITEM> sub-tags, which specify the list of items in the drop-down list. The item tag has CAPTION and VALUE attributes. CAPTION specifies the visible item text while VALUE specifies the underlying value for the option. SELECTED attribute specifies which item is initially selected. Then free-text is entered into an EDITABLE DropDown, the actual text is considered to be the value of the control.
Example:
`<CONTROL NAME="dropTagOptions" TYPE="DropDown" WIDTH="200">`
`<ITEM CAPTION="option1" VALUE="Value1"/>`
`<ITEM CAPTION="option2" VALUE="Value2" SELECTED/>`
`<ITEM CAPTION="option3" VALUE="Value3"/>`
`</CONTROL>`	

EDITABLE	The EDITABLE attribute decides whether it behaves like an editable combo-box or a fixed listbox.

DropDown example

```
<CONTROL NAME="lblAlign"
    TYPE="Label" CAPTION="Align:"
    ANCHOR="lblSource" CORNER="SW" DOWN=11 WIDTH=50/>

<CONTROL NAME="dropAlign"
    TYPE="DropDown" ANCHOR="lblAlign" CORNER="NE" WIDTH=100>
    <ITEM VALUE="TOP"    CAPTION="TOP" />
    <ITEM VALUE="MIDDLE" CAPTION="MIDDLE" SELECTED/>
    <ITEM VALUE="BOTTOM" CAPTION="BOTTOM" />
</CONTROL>
```

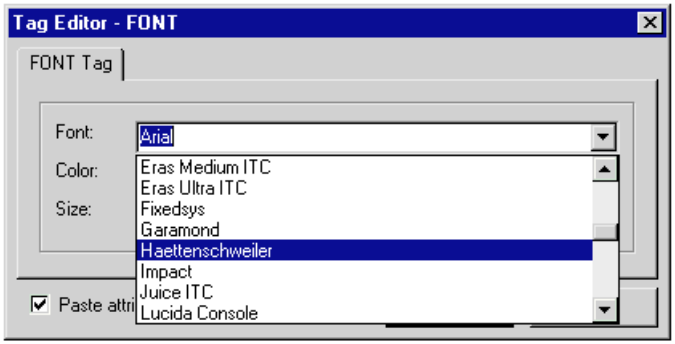


FontPicker control

<CONTROL TYPE="FontPicker" ...	
A simple drop-down list font picker.	
EDITABLE	The EDITABLE attribute decides whether a font name can be entered manually into the listbox.

FontPicker example

```
<CONTROL NAME="lblFace" TYPE="Label" CAPTION="Font:" DOWN=17 RIGHT=10
    WIDTH=50/>
<CONTROL NAME="fontFace" TYPE="FontPicker" ANCHOR="lblFace" CORNER="NE"
    WIDTH="MAXIMUM"/>
```

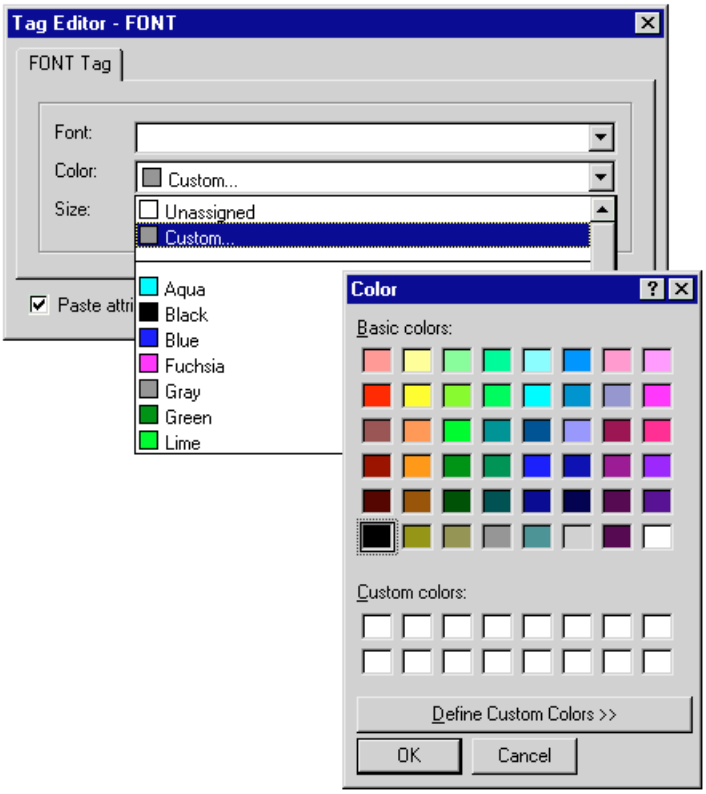


ColorPicker control

<CONTROL TYPE="ColorPicker" …
A simple drop-down color picker. Enables selection of a predefined color or special color code. Control returns value as a color name or hexadecimal value based on user's preferences.

ColorPicker example

```
<CONTROL NAME="lblColor" TYPE="Label" CAPTION="Color:" ANCHOR="lblFace"
    CORNER="SW" DOWN=11  WIDTH=50/>

<CONTROL NAME="colorColor" TYPE="ColorPicker" ANCHOR="lblColor"
    CORNER="NE" WIDTH="MAXIMUM"/>
<CONTROL NAME="lblColor" TYPE="Label" CAPTION="Color:" ANCHOR="lblFace"
    CORNER="SW" DOWN=11  WIDTH=50/>

<CONTROL NAME="colorColor" TYPE="ColorPicker" ANCHOR="lblColor"
    CORNER="NE" WIDTH="MAXIMUM"/>
```

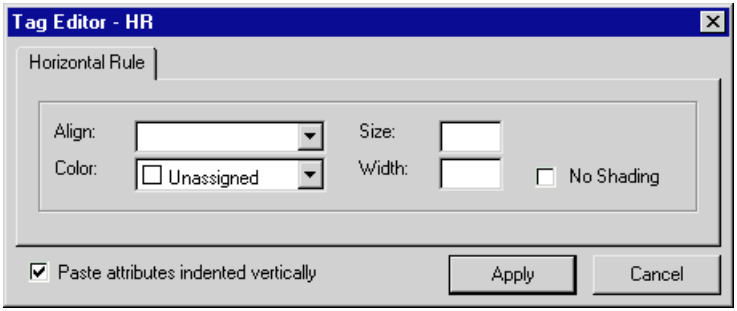


CheckBox control

<CONTROL TYPE="CheckBox" ...	
A simple checkbox control. Returns 'true' or 'false'.	
CAPTION	Caption displayed next to the checkbox.
CHECKED	YES\NO. Specifies initial status.

Checkbox example

```
<CONTROL NAME="checkNoShading"TYPE="CheckBox" CAPTION=" No Shading"
    ANCHOR="numWidth" CORNER="NE" DOWN=4 RIGHT=20 WIDTH=MAXIMUM/>
```



RadioGroup control

<CONTROL TYPE="RadioGroup" ...

A set of radio buttons. This tag requires <ITEM> sub-tags that specify the list of radio buttons. The item tag has CAPTION and VALUE attributes. CAPTION specifies the caption of the radio button, VALUE specifies the underlying value for the option. The SELECTED attribute specifies which radio options should be pre-selected.

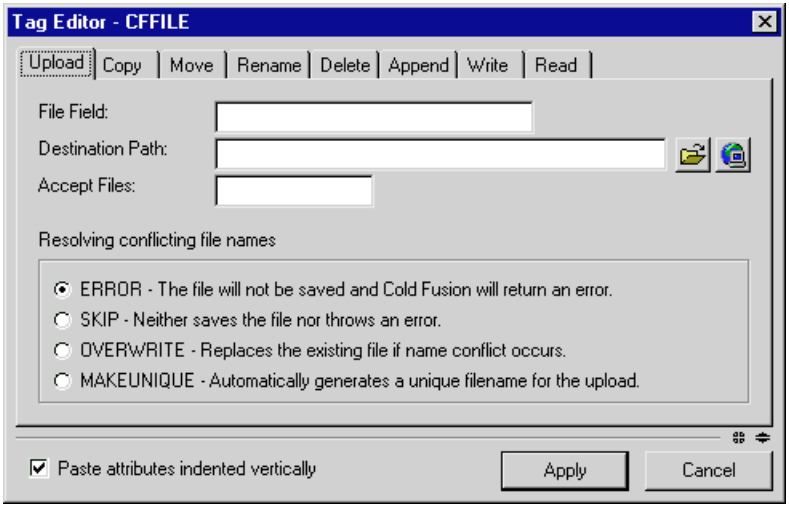
Example:

```
<CONTROL NAME="radioTagOptions" TYPE="RadioGroup" WIDTH="200">
<ITEM CAPTION="option1" VALUE="Value1"/>
<ITEM CAPTION="option2" VALUE="Value2" SELECTED/>
<ITEM CAPTION="option3" VALUE="Value3"/>
</CONTROL>
```

RadioGroup example

```
<CONTROL NAME="radioNameConflict"
    TYPE="RadioGroup" CAPTION="Radio One"
    ANCHOR="lblAccept" CORNER="SW" DOWN=35
    HEIGHT=MAXIMUM WIDTH=MAXIMUM

    <ITEM VALUE="ERROR" CAPTION="ERROR - The file
    will not be saved and ColdFusion will return
    an error." SELECTED="TRUE"/>

    <ITEM VALUE="SKIP" CAPTION="SKIP - Neither
    saves the file nor throws an error."/>

    <ITEM VALUE="OVERWRITE"  CAPTION="OVERWRITE -
    Replaces the existing file if name conflict occurs." />
```

```
    <ITEM VALUE="MAKEUNIQUE" CAPTION="MAKEUNIQUE - Automatically
       generates a unique filename for the upload." />

</CONTROL>
```

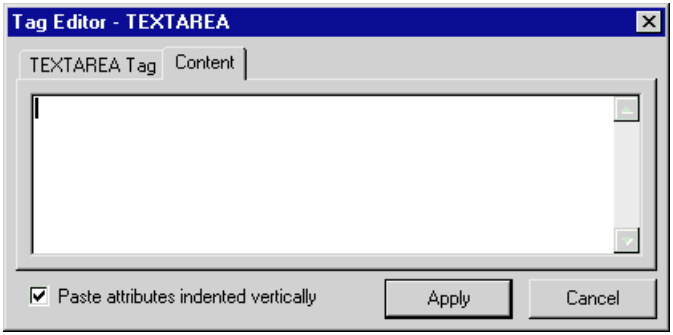


TextArea control

<CONTROL TYPE="TextArea" ...	
A simple multi-line text entry control.	
SCROLLBAR	NONE\HORIZONTAL\VERTICAL\BOTH. Specifies which scrollbars should be displayed.
WRAP	YES\NO. Enables wrapping of text.

TextArea example

```
<CONTROL NAME="txtContent" TYPE="TextArea"
    DOWN=5 RIGHT=5
    WIDTH=MAXIMUM HEIGHT=MAXIMUM
    MAXWIDTHPADDING=5 MAXHEIGHTPADDING=5/>
```



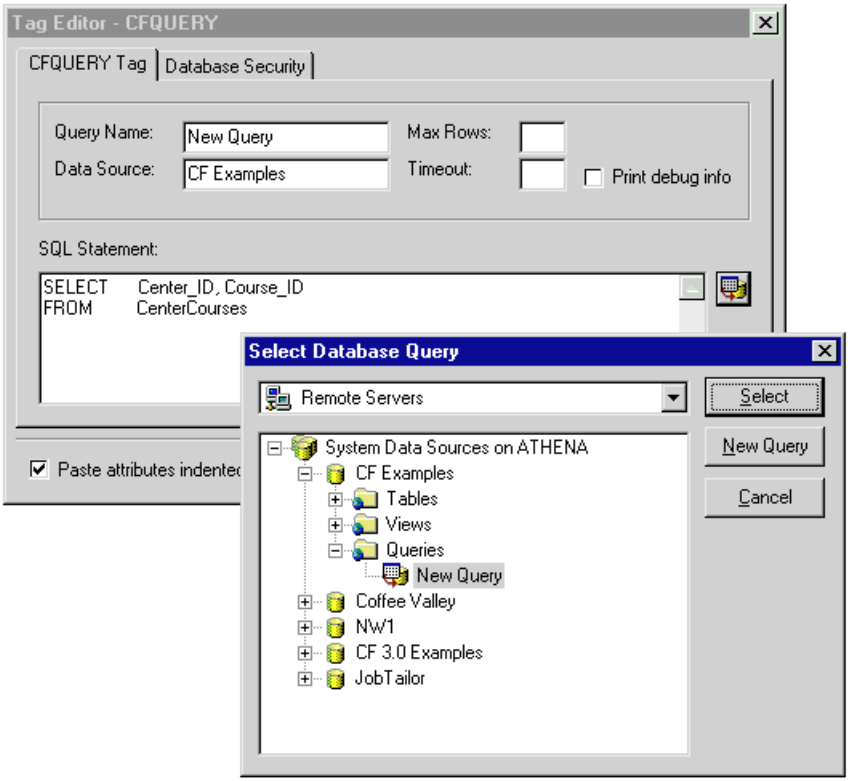
SQLTextArea control

<CONTROL TYPE="SQLTextArea" … (ColdFusion Studio ONLY)	
A multi-line text entry control that allows the user to execute an SQL statement. The control contains a button that the user can use to invoke the query builder.	
SCROLLBAR	NONE/HORIZONTAL/VERTICAL/BOTH. Specifies which scrollbars should be displayed.
DSNAMECONTROL	The name of the control that should be populated with the Data Source name when a query is selected.
QUERYNAMECONTROL	The name of the control that should be populated with the Query Name when a query is selected.
WRAP	YES\NO. Enables wrapping of text.

SQLTextArea example

```
<CONTAINER  NAME="TabPage1" TYPE="TabPage" CAPTION="CFQUERY Tag">

    <CONTAINER  NAME="Panel1" TYPE="Panel" DOWN=5 RIGHT=10
        WIDTH="MAXIMUM" HEIGHT=80>

        <CONTROL NAME="lblQueryName"
            TYPE="Label" CAPTION="Query Name:"
            DOWN=17 RIGHT=10 WIDTH=80/>
        <CONTROL NAME="lblDataSource"
            TYPE="Label" CAPTION="Data Source:"
            ANCHOR="lblQueryName" CORNER="SW"
        DOWN=10 RIGHT=0 WIDTH=80/>
        <CONTROL NAME="txtQueryName" TYPE="TextBox" ANCHOR="lblQueryName"
```

```
            CORNER="NE"  WIDTH=130/>
        <CONTROL NAME="txtDataSource" TYPE="TextBox"
            ANCHOR="lblDataSource" CORNER="NE" WIDTH=130/>
        <CONTROL NAME="lblMaxRows" TYPE="Label" CAPTION="Max Rows:"
            ANCHOR="txtQueryName"  CORNER="NE" DOWN=0 RIGHT=10 WIDTH=70/>
        <CONTROL NAME="lblTimeout" TYPE="Label" CAPTION="Timeout:"
            ANCHOR="txtDataSource" CORNER="NE" DOWN=0 RIGHT=10 WIDTH=70/>
        <CONTROL NAME="numMaxRows" TYPE="TextBox" ANCHOR="lblMaxRows"
            CORNER="NE" WIDTH=30/>
        <CONTROL NAME="numTimeout" TYPE="TextBox" ANCHOR="lblTimeout"
            CORNER="NE" WIDTH=30/>
        <CONTROL NAME="checkDebug" TYPE="CheckBox" CAPTION="Print debug
            info" ANCHOR="numTimeout" CORNER="NE" RIGHT=10 DOWN=4
            WIDTH=MAXIMUM/>

    </CONTAINER>

        <CONTROL NAME="lblSQLStatement" TYPE="Label" CAPTION="SQL
            Statement:" ANCHOR="Panel1" CORNER="SW"DOWN=10 RIGHT=0
            WIDTH=110/>
        <CONTROL NAME="txtSQLStatement" TYPE="SQLTextArea"
            ANCHOR="lblSQLStatement" CORNER="SW" DOWN="8" WIDTH=MAXIMUM
            HEIGHT=MAXIMUM DSNAMECONTROL="txtDataSource"
            QUERYNAMECONTROL="txtQueryName"/>

</CONTAINER>
```



FileBrowser control

<CONTROL TYPE="FileBrowser" ...	
A textbox used to enter a file path. The control contains two toolbar buttons providing accessibility to local and remote file\directory browsing.	
CAPTION	The window caption that should displayed in the file\directory browsing dialogs. For example: CAPTION="Select File You Want Uploaded".
FILTER	The file filter that should be used by the file\directory browsing dialogs. For example: FILTER="*.gif;*.jpeg;*.jpg".

<CONTROL TYPE="FileBrowser" … (Continued)	
DIRONLY	YES\NO. Specifies that a directory is being selected. This option will convert the file dialogs accessible from the browse buttons to become directory-browsing dialogs. By default, file-browsing is assumed.
FILENAMEONLY	YES\NO. Specifies that only a file name should be entered into the textbox when a file is specified in a local or remote file dialog. By default the entire path would be pasted. The attribute is irrelevant when the DIRONLY attribute is used.
RELATIVE	YES\NO. Instructs the control to calculate the relative path when a file or a directory path is selected. The relative path is calculated relative to the file currently opened. When a new file is being edited, it first has to be saved before a relative path can be calculated. By default, the absolute path is assumed.

FileBrowser example

```
<CONTROL NAME="lblSource" TYPE="Label"
    CAPTION="Source:" DOWN=17 RIGHT=10
    WIDTH=60/>

<CONTROL NAME="txtSource" TYPE="FileBrowser" ANCHOR="lblSource"
    CORNER="NE" WIDTH="MAXIMUM" RELATIVE
    FILTER="*.htm;*.html;*.cfml;*.cfm;*.asp" />
```

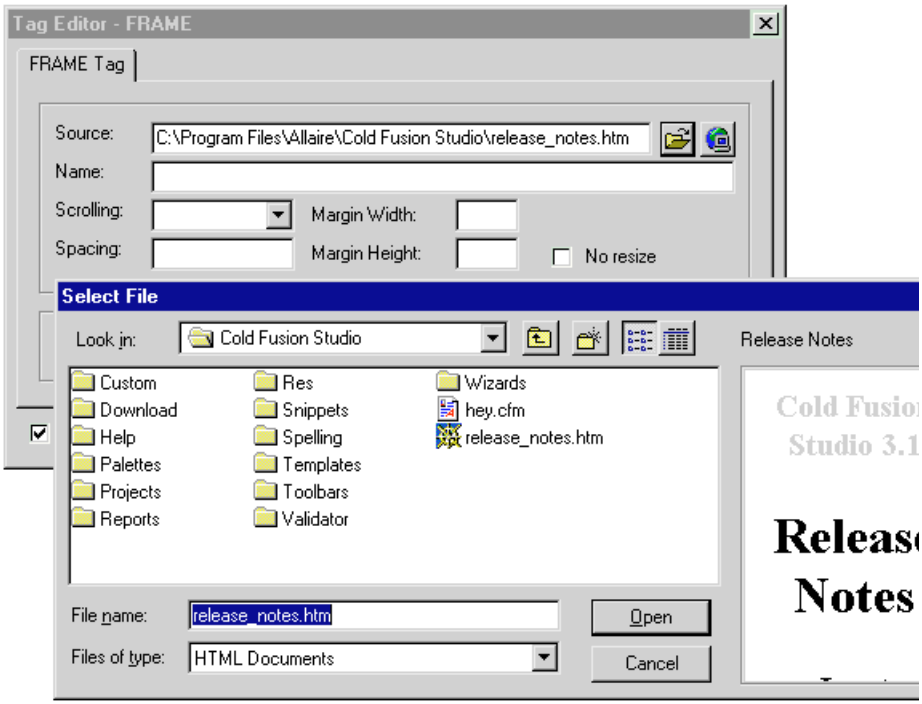


Image control

<CONTROL TYPE="Image" ...	
An image control capable of containing BMP images.	
FILEPATH	Specifies the relative path of the image file from the editor template.
AUTOSIZE	YES/NO. Automatically sizes the control to the image it contains. This option is overridden if WIDTH or HEIGHT are explicitly specified.
TRANSPARENT	YES/NO. Controls transparency.
CENTER	YES/NO. Centers the image.

Image example

```
<CONTROL NAME="imgApplet" TYPE="Image" FILEPATH="Images/Applet.bmp"
    DOWN=10 RIGHT=10 AUTOSIZE="Yes"/>
```

ActiveX control

<CONTROL TYPE="ActiveX" ...	
A container for an embedded ActiveX control. For an ActiveX control to function properly within a tag editor the ActiveX control must implement four methods: • void SetValue(LPCTSTR sValue) - called to set the control value. • BSTR GetValue() - called to retrieve the control value. • void SetFocus() - called to set focus to the control. • void IntitializeFromMarkup(LPCTSTR strMarkup) – called to initialize the ActiveX control with the CONTROL tag in the Custom Markup application.	
PROGID	The ProgID of the ActiveX control.

ActiveX example

```
<CONTROL NAME="activexGizmoPicker" TYPE="ActiveX"
    PROGID="company.Gizmo"/>
```

Building Custom Wizards

This section describes how you can give users an easy way to enter information that can then be published on the Web or used to drive ColdFusion applications. Wizards are an integral part of many software products today because they invite users to perform complex tasks in an orderly, comprehensible interface. Another benefit is that a well-designed wizard controls its input and ensures a high probability of user success.

Allaire makes extensive use of wizards in HomeSite and Studio and now extends that capability to developers. If you have worked with VTML to create or edit tag dialogs, you are familiar with building interface containers and controls and with defining page layout. You can now add the Wizard Markup Language (WIZML) to your skill set and add wizards to your applications.

To create a wizard:

1. Write a wizard definition file (.vtm) to specify the pages, parameters, output, and logical flow.
2. Implement one or more output template files (.wml) for the wizard.
3. Create wizard graphic (.bmp) files.

Each of these steps is described in detail in the following sections.

Saving wizard files

The recommended way to organize wizards and supporting files is to save the .vtm and .wml files in the `\Wizards\Custom` folder and to save the image files in the `\Wizards\Images` folder of your Studio directory.

Creating Wizard Definition Pages

The first step in building a custom wizard is to write a VTML file to define the interface and output parameters. This section describes the VTML tags used in this part of the process and presents an example definition file.

VTML for Wizards tag summary

The following is the hierarchy of the tag set for wizard definition files:

WIZARD — The enclosing tag for the entire file; it defines the new wizard with a name, caption and default image.

PARAM — When used as a sub-tag of the WIZARD tag, it defines a parameter that the wizard uses to generate its output. These parameters are then available for use with wizard output templates.

TEMPLATE — Defines an output template and identifies the file used for text output.

PAGE — Defines a wizard page that determines which page class to load.

PARAM — When used as a sub-tag of the PAGE tag, it sets the behavior of a page class. This is useful for standard pages which are intended to be re-used across multiple wizards.

INPUT — Defines an input control on a wizard page. If the NAME attribute of the INPUT control matches both the name of a control on the wizard page as well as the name of a PARAM defined within the WIZARD tag then the control is automatically 'bound' to the underlying parameter without requiring any explicit code.

NEXTPAGE — The NEXTPAGE tag allows for complex routing between pages based on the value of conditional expressions.

PAGELAYOUT — Same as VTML syntax for containers and controls.

VTML for Wizards tag reference

The following tables provide the complete VTML syntax for writing wizard definition files.

WIZARD

Attribute	Description
NAME	Optional. Used to resolve the names of pages that belong to a specific wizard. May be registered using the syntax `WizardName.PageName`.
CAPTION	Optional. Caption to display in the wizard's title bar.
IMAGE	Optional. Default bitmap to use for pages within the wizard.

PARAM (for WIZARD tag)

Attribute	Description
NAME	Name of the parameter.
VALUE	Initial value of the parameter.
REQUIRED	Optional. The wizard manager will not enable the Finish button until all required parameters are entered.

TEMPLATE

Attribute	Description
NAME	File name of the wizard (.wml) output template.
OUTPUTFILE	Name of the file to which output to based on the results of processing the template.
OUTPUTPATH	Optional. Output directory for the file. Defaults to the value of the variable `LOCATION`. Note that you must provide a wizard page where the user can specify this value.
DESCRIPTION	Optional. Description of the wizard page's function for use in the output summary page.

PAGE

Attribute	Description
TYPE	Required for dynamic wizards. "Dynamic" - VTML layout "PageName" - from Page Library (see the Wizard Definition Page Library section below)
NAME	Name of the page.
CAPTION	Caption to display in the top portion of the page.
IMAGE	Optional. Override of the default wizard bitmap.
CONDITION	Optional. Conditional expression which determines if the page is displayed.
NEXTPAGE	Optional. Name of page to go to after the current one. The default page is to the next page defined in the configuration file.

PARAM (for PAGE tag)

Attribute	Description
NAME	Name of the parameter.
VALUE	Value of the parameter.

INPUT

Attribute	Description
NAME	Name of the form control to which the `INPUT` is bound.
PARAM	Optional. Name of parameter to which the `INPUT` is bound. Defaults is the NAME attribute).
DEFAULT	Optional. Default value for the input.
REQUIRED	Optional. Is the input required?

Attribute	Description (Continued)
VALIDATIONMSG	Optional. A message to display to the user if the input is required and a value is not entered.
LISTCONTENTS	Optional. If this is TCustomListBox or TCustomComboBox based input, then a comma-separated list will be used to populate the list with values.

NEXTPAGE

Attribute	Description
NAME	Name of a page to go to next.
CONDITION	Conditional expression that determines whether to go to the page. If multiple NEXTPAGE tags are specified, then the first one to match a CONDITION will be the next page.

Dynamic expressions in tags

Any tag attribute may combine static, constant text with embedded dynamic expressions that reference parameters or input controls. To embed an expression within a text string, the following syntax is utilized:

```
$${ expression }
```

So, for example, to set the REQUIRED attribute of a parameter based on whether another value was set, you would use the following syntax:

```
<PARAM NAME="RowsPerPage" VALUE="10"
    REQUIRED="$${ ParameterExists('Customize') }">
```

Or, to customize the OUTPUTFILE attribute of the TEMPLATE tag using a name attribute entered by the user, you would use the following syntax:

```
OUTPUTFILE="$${Name}Admin.cfm">
```

The expression syntax supported within the wizard configuration file is the same as the one supported in wizard output templates (see the reference section for more details).

Bound controls

One of the most powerful capabilities of wizard pages are bound controls. Bound controls allow you to place controls onto the wizard page and have their values automatically bound to wizard parameters. To do this, simply add an INPUT sub-tag to the PAGE tag for each control you wish to bind, making sure that the NAME attribute of the INPUT tag matches the Name property of the control. All controls specified in the layout can be bound.

Wizard definition page example

This sample wizard creates an HTML template.

```
<WIZARD NAME="DefaultTemplate" CAPTION="Default HTML Template">

<!--- wizard parameters --->
<PARAM NAME="sDocType" VALUE="HTML 4.0" REQUIRED="true">
<PARAM NAME="sTitle" VALUE="">
<PARAM NAME="bMetaDescr" VALUE="false">
<PARAM NAME="sMetaDescr" VALUE="">
<PARAM NAME="bMetaKeywords" VALUE="false">
<PARAM NAME="sMetaKeywords" VALUE="">

<!--- WIZARD PAGE --->

<!--- attributes page --->
<PAGE NAME="DocAttribs" TYPE="DYNAMIC"
    CAPTION="HTML Document Attributes"
    IMAGE="..\\images\\main.bmp">

    <PAGELAYOUT>
        <CONTROL NAME="lblDocType" TYPE="label"
            DOWN="10" RIGHT="10"
            WIDTH="90"
            CAPTION="Document Type:"
        />

        <CONTROL NAME="ddDocType" TYPE="DropDown"
            EDITABLE="no"
            ANCHOR="lblDocType" corner="NE" WIDTH="MAXIMUM" down="-5">
                    <ITEM CAPTION="HTML 2.0" VALUE="HTML 2.0"/>
                    <ITEM CAPTION="HTML 3.2" VALUE="HTML 3.2"/>
                    <ITEM CAPTION="HTML 4.0" VALUE="HTML 4.0"/>
        </CONTROL>

        <CONTROL NAME="lblTitle" TYPE="label"
            ANCHOR="lblDocType" CORNER="SW" down="20"
            WIDTH="90"
            CAPTION="Title:"
        />

        <CONTROL NAME="tbTitle" TYPE="TextBox"
            ANCHOR="lblTitle" CORNER="NE" WIDTH="MAXIMUM" down="-5"
        />

    <CONTAINER NAME="pnlMetaDescription" TYPE="Panel"
            CAPTION="Meta Description"
            ANCHOR="lblTitle" CORNER="SW" DOWN="20"
            WIDTH="MAXIMUM" MAXWIDTHPADDING="10" HEIGHT="80"
                LFHEIGHT="90">

            <CONTROL NAME="chkMetaDescr" TYPE="CheckBox"
                CAPTION="Add meta description:"
```

```
                DOWN="20" RIGHT="15"
                WIDTH="MAXIMUM"/>

            <CONTROL NAME="tbMetaDescr" TYPE="TextBox"
                ANCHOR="chkMetaDescr" CORNER="SW" DOWN="8"
                WIDTH="MAXIMUM"/>

        </CONTAINER>

    </PAGELAYOUT>

    <INPUT NAME="ddDocType" PARAM="sDocType">
    <INPUT NAME="tbTitle" PARAM="sTitle" REQUIRED="yes" VALIDATIONMSG=
        "Please enter a document title" or some equivalent message>
    <INPUT NAME="chkMetaDescr" PARAM="bMetaDescr">
    <INPUT NAME="tbMetaDescr" PARAM="sMetaDescr">
</PAGE>

<!--- attributes page --->
<PAGE NAME="MetaKeywords" TYPE="DYNAMIC"
    CAPTION="Meta Keywords"
    IMAGE="..\\images\\main.bmp">

    <PAGELAYOUT>

        <CONTROL NAME="chkMetaKeywords" TYPE="CheckBox"
            CAPTION="Add meta keywords:"
            DOWN="15" RIGHT="10"
            WIDTH="MAXIMUM"/>

        <CONTROL NAME="taMetaKeywords" TYPE="TextArea"
            ANCHOR="chkMetaKeywords" CORNER="SW" down="10"
            HEIGHT="MAXIMUM" width="MAXIMUM"/>

    </PAGELAYOUT>

    <INPUT NAME="chkMetaKeywords" PARAM="bMetaKeywords">
    <INPUT NAME="taMetaKeywords" PARAM="sMetaKeywords">

</PAGE>

<!--- OUTPUT TEMPLATE --->

<TEMPLATE
    NAME="Custom.wml"
    OUTPUTFILE="MyFile.cfm"
    DESCRIPTION="New HTML file">

</WIZARD>
```

Creating Wizard Output Templates

The Wizard Markup Language (WIZML) enables the customization of files produced by the wizards. WIZML is used inside the templates to dynamically create files based on the data provided by the wizard. For example, if a wizard generates a tag called <GIZMO> with a single attribute FILEPATH, the template could look as simple as this:

```
<GIZMO FILEPATH="$${txtFilePath}">
```

This example will create a file with a single <GIZMO> tag. Notice the syntax `$${variablename}` that is used to populate the value of FilePath with the actual value entered in the wizard.

Using WIZML

WIZML output templates use a high-level markup syntax that works very much like CFML. Supported tags include WIZSET, /WIZELSE/WIZELSEIF, WIZLOOP, and WIZINCLUDE. In addition, a simple expression syntax that is a subset of the ColdFusion expression syntax and function library is supported within output templates.

Parameters

Output templates are driven by the values of parameters, much like ColdFusion templates are driven by the values of Form and URL parameters. Parameters can be output directly or can be used to customize the type of output generated. The values of these wizard parameters can originate from several locations:

- From a value set by a PARAM tag provided by the wizard
- From an embedded tag editor control
- Through execution of the WIZSET tag within the output template

To output the value of a parameter within a template, use a double dollar sign escape sequence. For example, to output the value of a variable named Color you would use the syntax $${Color}. While this is the recommended syntax, you can use a simpler form when for a parameter value within the attribute of a WIZ tag. For example, `<WIZIF Color= “black”>` is valid.

Expressions and functions

In addition to outputting and manipulating simple parameter values, an expression syntax that includes support for a subset of the ColdFusion functions is also provided. To output the value of an expression you add a set of curly braces to the $$ and include the expression within the braces, for example:

```
$${ 'This is the ' & Color }

$$( 'The result of 7 divided by 22 is ' & 7/22 }

$$( Left( 'FooBar', 3 ) }
```

As you can see from the example above, these expressions are very similar to ColdFusion expressions. Strings are delimited using the single quote character. The customary set of arithmetic and concatenation operators are supported (+,-,*,/,&). The comparison operators LT, LTE, GT EQ, and NEQ are supported, and logical comparisons using AND, NOT, and OR are supported.

The two main categories of functions currently supported are string and runtime.

String functions

- Chr
- Compare
- CompareNoCase
- Find
- FindNoCase
- LTrim
- RTrim
- Trim
- LCase
- UCase
- Len
- Left
- Right
- Mid
- RepeatString

Runtime functions

- ParameterExists
- SetVariable
- Evaluate

The syntax and behavior of all of these functions is identical to the equivalent functions in ColdFusion, except for ParameterExists, which takes a string argument rather than a direct variable reference.

WIZ Tags

The behavior of wizard output templates is controlled by the use of WIZ tags in the template. These tags are like ColdFusion tags except that they are prefixed with the characters WIZ instead of CF.

Supported tags

- **WIZSET** — Sets a wizard parameter.
- **WIZINCLUDE** — Includes another wizard output template.
- **WIZLOOP** — Iterates over a set of output.
- **WIZBREAK/WIZCONTINUE** — Assists in loop flow control.
- **WIZIF/WIZELSEIF/WIZELSE** — Sets conditional flow control.

Special considerations

Strings used within an output template tag attribute use the C-language convention (\) for escaping special characters. For example, the following attribute uses a newline character to split the value into two lines:

```
CAPTION="This is line one\nThis is line two"
```

Other special characters of note include carriage-return (\r), tab (\t), and slash (\\). For example, the following attribute references a template in a directory two levels above the current directory:

```
TEMPLATE="..\\..\\Header.wml"
```

WIZML reference

Here is the complete Wizard Markup Language syntax:

WIZSET

WIZSET works the same way as the ColdFusion CFSET tag.

For example:

```
<WIZSET Color = 'Red'>
<WIZSET Pi = 7/22>
<WIZSET ShortName = Left( LongName, 5 )>
```

WIZINCLUDE

Attribute	Description
TEMPLATE	Required. The relative path of a template that is to be included in the currently executing template.

WIZLOOP, WIZBREAK, and WIZCONTINUE

The WIZLOOP tag supports several types of loops including:

- Index-based (using the INDEX, FROM, TO, and STEP attributes)
- Condition-based (using the CONDITION attribute)
- List-based (using the INDEX and LIST attributes)
- TStringList-based (using the INDEX and STRINGLIST attributes)

The attributes for the WIZLOOP tag are as follows:

WIZLOOP Tag Attributes	
Attribute	**Description**
INDEX	Name of a variable for a loop to set on each iteration (required for index and list-based loops). It serves as a counter.
FROM	Index to start looping from.
TO	Index to loop to.
STEP	Step value for each increment (can be positive or negative).
CONDITION	Conditional expression to control whether the loop should be exited.
LIST	A list of CommaText format. This is the format that a Delphi-based string list uses to store textual representations of itself.
STRINGLIST	The name of a parameter created with the SetObjectParameter function which is of type TStringList.

The WIZBREAK and WIZCONTINUE tags have no attributes, and can be placed anywhere within a WIZLOOP tag to either exit the loop (WIZBREAK) or move on to the next loop iteration (WIZCONTINUE).

WIZIF, WIZELSEIF, and WIZELSE

The WIZIF, WIZELSEIF, and WIZELSE tags work identically to the corresponding CFML tags. Any valid Boolean expression can be used in the WIZIF and WIZELSEIF tags.

```
<WIZIF sDocType eq "HTML 4.0">

    <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
        <wizelse sDocType eq "HTML 3.2">
    <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2//EN">
        <wizelse sDocType eq "HTML 2.0">
    <!DOCTYPE HTML PUBLIC  "-//IETF//DTD HTML 2.0//EN">

</WIZIF>

<HTML>
<HEAD>
    <TITLE>$${sTitle}</TITLE>

<WIZIF bMetaDescr eq "true">
    <META NAME="description" CONTENT="$${sMetaDescr}">
</WIZIF>

<WIZIF bMetaKeywords eq "true">
    <META NAME="keywords" CONTENT="$${sMetaKeywords}">
</WIZIF>

</HEAD>
<BODY>


</BODY>
</HTML>
```

Wizard Definition Page Library

ColdFusion Studio contains a set of seven page definition files that you can use to quickly build data access capabilities into your wizards. The pages are based on the Studio Drill-down Wizard available in the File > New dialog.

SelectNameAndLocation

Description

This page collects application name and location where the destination directory for the output templates. It allows the user to select the directory from the local or remote server. This page is required if any of the following pages are included in the wizard.

Exposes controls

editApplicationName (textbox) — After the page is submitted, the content of this textbox are copied to the SafeApplicationName parameter with all non-allowed characters stripped out. This value can be then used as part of the file names.

editLocation (textbox) — This control should be bound to the Location parameter. The value of this parameter is preset to the directory opened in the left pane.

Example

```
<PAGE name="SelectWizardNameAndLocation" type="SelectNameAndLocation"
caption="Data Drill-Down Application" image="..\\images\\Main.bmp">

    <INPUT name="editApplicationName" param="ApplicationName"
        required="yes"
        validationMsg="You cannot leave the Application Name field
        blank">

    <INPUT name="editLocation" param="Location"required="yes"
        validationMsg="You cannot leave the Location field blank">

</PAGE>
```

SelectDataSource

Description

This page displays a list of available data sources on the selected local or remote server.

Uses controls

cbDataSources (dropdown) — This control lists all available data sources. The control exchanges with the bound parameter the name of the selected data source.

Reads parameters

ListBoxLabel — Caption for the dropdown control.

ListBoxDescription — More detailed description for the dropdown control.

ResetParams — Comma-delimited list of parameters that should be emptied if user reselects the data source.

RemoveParams — Comma-delimited list of parameters that should be removed if user reselects the data source.

Example

```
<PAGE name="DataSource" type="SelectDataSource" caption="Data Source"
    image="..\\images\\SelectData.bmp">

    <PARAM name="ListBoxLabel" value="Select data source:">
    <PARAM name="ListBoxDescription"
        value="Choose the data source from which you would like to display
        data.\n\nIf your database is not registered as ODBC data source,
        open the ODBC administrator in Control Panel and add system data
        source for this database.">
    <PARAM name="ResetParams" value="Joins">
    <PARAM name="RemoveParams" value="Tables,SearchFields,ResultFields,
        DetailFields,UniqueIdentifier">

    <INPUT name="cbDataSources" param="DataSource" required="yes"
        validationMsg="You did not select the data source. Please select
        one before proceeding.">

</PAGE>
```

SelectTables

Description

This page displays a list of tables in the specified data source and lets the user select one or more of them.

Exposes controls

lstTables (dropdown) — The control lists tables in the data source specified in the DataSource parameter. The control exchanges with the bound parameter a comma-delimited list of selected tables.

Reads parameters

DataSource (required) — Name of the data source

ListBoxLabel — Caption for the dropdown control

ListBoxDescription — More detailed description for the dropdown control

MultiSelect — If "YES", user is allowed to select more than one table.

ResetParams — Comma-delimited list of parameters that should be emptied if user reselects the tables.

RemoveParams — Comma-delimited list of parameters that should be removed if user reselects the tables.

Example

```
<PAGE name="Tables" type="SelectTables" caption="Tables"
    image="..\\images\\SelectTable.bmp">

    <PARAM name="DataSource" value="$${DataSource}">
    <PARAM name="ListBoxLabel" value="Select database tables:">
    <PARAM name="ListBoxDescription"
        value="Please specify the tables which will be involved in this
        application. This should include any tables against which you
        would like to search or tables containing data that will be
        displayed on either the Result or Detail pages.\n\nPress Ctrl or
        Shift together with the mouse click in order to select more than
        one table. Do not select unrelated tables.">
    <PARAM name="MultiSelect" value="yes">
    <PARAM name="ResetParams" value="Joins">
    <PARAM name="RemoveParams" value="SearchFields,ResultFields,
        DetailFields,UniqueIdentifier">

    <INPUT name="lstTables" param="Tables" required="yes"
        validationMsg="You did not select any tables. Please select at
        least one before proceeding.">

</PAGE>
```

SelectTable

Description

This page lets the user select one of the tables from the specified data source.

Exposes controls

cbTables (dropdown) — The control lists tables in the data source specified in the DataSource parameter.

The control exchanges with the bound parameter the name of the selected table.

Reads parameters

DataSource (required) — Name of the data source

ListBoxLabel — Caption for the dropdown control

ListBoxDescription — More detailed description for the dropdown control

ResetParams — Comma-delimited list of parameters that should be emptied if user reselects the table.

RemoveParams — Comma-delimited list of parameters that should be removed if user reselects the table.

Example

```
<PAGE name="Table" type="Table" caption="Table"
    image="..\\images\\SelectTable.bmp">

    <PARAM name="DataSource" value="$${DataSource}">
    <PARAM name="ListBoxLabel" value="Select database table:">
    <PARAM name="ListBoxDescription"
        value="Records from this table will be displayed in the record
        viewer.">
    <PARAM name="RemoveParams" value="Table,ViewFields,EditFields,
        UniqueIdentifier">

    <INPUT name="cbTables" param="Table" required="yes"
        validationMsg="You didn't select the table. Please select one
        before proceeding.">

</PAGE>
```

SelectTableJoins

Description

The page lets the user select fields from multiple tables for table joins.

Exposes controls

lstJoins (dropdown) — The control lists field pairs as selected from the dropdown field lists. This list can be pre-filled from the comma delimited list of items, where each item has format "table1.field1=table2.field2".

Reads parameters

DataSource (required) — Name of the data source

Tables (required) — Comma-delimited list of the tables

ListContent (required) — Input parameter

Example

```
<PAGE name="TableJoins" type="SelectTableJoins" caption="Table Joins"
image="..\\images\\SelectJoins.bmp">

    <PARAM name="DataSource" value="$${DataSource}">
    <PARAM name="Tables" value="$${Tables}">
    <PARAM name="ListContent" value="$${Joins}">

    <INPUT name="lstJoins" param="Joins">

</PAGE>
```

SelectFields

Description

This page displays all fields from the specified tables and lets the user select one or more of them.

Exposes controls

lstFields (dropdown) — The control lists fields in the tables specified in the Tables parameter. The control exchanges with the bound parameter a comma-delimited list of selected fields, where each item is in format "table.field=type,size,required". *Type* specifies the type of the table field can have following values (BIT, CHAR, MEMO, INT, FLOAT, DATETIME. *Size* gives information about size of the field in bytes (this might not be true for certain DBMS and types). *Required* is "YES" if the field value cannot be NULL.

Reads parameters

DataSource (required) — Name of the data source

Tables (required) — Comma-delimited list of the tables

ListBoxLabel — Caption for the dropdown control

ListBoxDescription — More detailed description for the dropdown control

MultiSelect — If "YES", user is allowed to select more than one field

Example

```
<PAGE name="SearchFields" type="SelectFields" caption="Fields for Search
    page" image="..\\images\\SearchCriteria.bmp">

    <PARAM name="DataSource" value="$${DataSource}">
    <PARAM name="Tables" value="$${Tables}">
    <PARAM name="ListBoxLabel" value="Select the search fields:">
    <PARAM name="ListBoxDescription"
        value="Choose all fields that should be included as search
        criteria on the Search page. Press Ctrl or Shift together with the
        mouse click in order to select more than one field.">
    <PARAM name="MultiSelect" value="yes">

    <INPUT name="lstFields" param="SearchFields" required="yes"
        validationMsg="You did not select any fields. Please select at
        east one before proceeding.">

</PAGE>
```

SelectField

Description

This page lets the user select one field from the list of fields. The list of fields contains all fields from the specified tables.

Exposes controls

cbFields (required) — The control lists fields in the tables specified in the Tables parameter. The control exchanges with the bound parameter the field information in the format "table.field=type.size.required" (see the SelectFields page)

Reads parameters

DataSource (required) — Name of the data source

Tables (required) — Comma-delimited list of tables

ListBoxLabel — Caption for the dropdown control

ListBoxDescription — More detailed description for the dropdown control

Example

```
<PAGE name="IDField" type="SelectField" caption="Unique Identifier"
    image="..\\images\\UniqueIDDetail.bmp">

    <PARAM name="DataSource" value="$${DataSource}">
    <PARAM name="Tables" value="$${Tables}">
    <PARAM name="ListBoxLabel" value="Select the unique identifier for
        the Detail page:">
    <PARAM name="ListBoxDescription" value="In order to 'drill-down' to
        the detail page the wizard needs to know the unique identifier for
        the detail page. This is the field that determines which record
        should be displayed in detailed form.\n\nFor example, if you are
        building an application to search an employee database you might
        use a field called 'Employee_ID' as the unique identifier.">
    <PARAM name="MultiSelect" value="no">

    <INPUT name="cbFields" param="UniqueIdentifier" required="yes"
        validationMsg="You did not select the unique identifier. Please
        select one before proceeding.">

</PAGE>
```

Special

A

B

C

D

S

W

Y