BUILDING ENTERPRISE PORTALS

U.S. $8.99 (Canada $9.99)

ColdFusion Developer's Journal

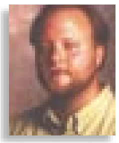
ColdFusionJournal.com

June, 1999 Volume: 1 Issue: 3

RETAILERS PLEASE DISPLAY UNTIL JULY 31, 1999

SYS-CON PUBLICATIONS

The Problem of Complexity
Complex projects call for methodical measures



Able Solutions

www.ablecommerce.com

Macromedia

www.macromedia.com

Interland

www.interland.com

COLDFUSION DEVELOPER'S JOURNAL (ISSN # 1523-9101) is published bimonthly (6 times a year) for $49.99 by SYS-CON Publications, Inc., 39 E. Central Ave., Pearl River, NY 10965-2306. Application to mail at Periodicals Postage rates is pending at Pearl River, NY 10965 *and additional mailing offices.* **POSTMASTER**: Send address changes to: COLDFUSION DEVELOPER'S JOURNAL, SYS-CON Publications, Inc., 39 E. Central Ave., Pearl River, NY 10965-2306.

GUEST EDITORIAL

by **Steve Drucker**

Enter the Scalability Zone

Over the past three years as an Allaire partner, I have been involved in countless marketing presentations where I touted ColdFusion as the best solution for producing Web-based applications. During this time other product vendors, instead of selling against CF based on feature comparisons, have instead focused on peripheral issues. Throughout the CF 2.0 product cycle, I would routinely be questioned about Allaire's staying power in the market, financial performance and capital inflows. Allaire's rapidly expanding market share and IPO silenced those critics. During the CF 3.x product cycle, our competitors shifted to attack CF as an unsuitable product for the enterprise, questioning whether it could "scale" to support an unlimited number of concurrent users.

Despite anecdotal evidence to the contrary (AutoByTel and SmartMoney interactive), some clients remained unconvinced. Allaire recognized this tactic and proceeded to load CF 4 with features to address this concern. Chief among these is the Enterprise edition's BrightTiger ClusterCats integration. Certainly ClusterCats has been an invaluable marketing tool for positioning CF as ready for the enterprise. However, one must ask whether using clustering eliminates the need for designing software that functions efficiently. Throughout my experience as a scalability consultant, I have yet to encounter a Web site that could not be made to scale by tweaking the underlying source code. Optimizing CF applications should be viewed as a five-step process.

Step 1: Review the Debug Stream

Reviewing the debug stream continues to be the fastest method for identifying slowly executing pages. A total page execution time over five seconds may indicate a problem. Review the queries that are executed on each page. Are you retrieving the same data each time? Try caching the query for enhanced performance.

Step 2: CFML Code Review

New CF developers, unfamiliar with the vast number of functions, choose to code their own, less efficient methods in CFML. Granted, reading the CF function reference may not be as exciting as a Grisham novel, but it's necessary. Using the valuelist() function to generate a comma delimited list from the values in a given query column instead of coding a concatenation algorithm using CFLOOP can shave seconds off your execution time. A list of other CFML coding tips can be found on the Allaire Web site at http://allaire.com/Handlers/index.cfm?-ID=1576&Method=Full.

Step 3: Leverage the Database

Poor database design runs rampant in our industry. Review your data model using a database design tool such as ER/Win from Platinum Software. Are you enforcing referential integrity? Is the database represented in at least a third normal form? Consider the queries you'll need to execute throughout your application and ask yourself how easy (or difficult) it will be to retrieve the information from your data structure. Next, move all your queries into SQL Stored Procedures. CF 4.01 now supports returning record sets from ORACLE and passing parameters to MS Access Queries through the <CFSTOREDPROC> tag. You'll get your data returned much faster than using traditional SQL passthrough.

By using Stored Procedure program constructs you may also be able to combine several SQL statements into a single call, eliminating the overhead associated with multiple queries. Still seeing poor response times? Review your database's SQL reference. Query execution time may be reduced by up to 90% by utilizing aggregate functions and platform-specific SQL constructs that quickly perform calculations on vast sets of data.

Step 4: Leverage the Client

Many developers irrationally fear using client-JavaScript in their applications. While support for the language does vary widely among the browsers, a high payoff can result from transferring application logic from the server to the browser. End users see better response times since no server request is involved and the traffic between the browser and server can be greatly reduced. These high-payoff items outweigh the slightly longer development and debugging cycle that JS development requires.

Step 5: Review the Hardware Configuration

Once you've optimized the application programmatically, review the CF server's hardware and software configuration. Make sure that the application and database servers are on different machines and have adequate amounts of RAM (512 MB and 1 GB, respectively) if you expect a lot of traffic. Allocate enough memory for the CF template cache to allow the server to cache all of your application pages. If you've gone through this entire process and are still experiencing server timeouts or require failover capability, then it may be time to take the clustering plunge.

The continual advances in processing performance have allowed developers on other platforms to write bloated, inefficient code. I view ClusterCats, mistakenly seen as a panacea for scalability, as potentially having the same effect on CF application development. Stop the madness before it begins. CFDJ

About the Author

Steve Drucker is president and CEO of Fig Leaf Software, with offices in Washington, DC, and Atlanta, Georgia. He founded the very first ColdFusion users group (DC-CFUG), co-authored the ColdFusion Web Database Construction Kit *(first and second editions) and is a certified Allaire instructor. He can be reached at sdrucker@figleaf.com.*

sdrucker@figleaf.com

CFDJ FEATURE

FROM ZERO TO FINISH IN THREE MONTHS

Launching documents around the globe calls for high-tech control measures

by
Leon Eno

Anyone involved with meeting tight deadlines quickly realizes the power of ColdFusion. When my marketing organization asked me (Leon) to create a system to assist in revision control for new-product launch materials around the world, I wasn't given much time to do it. Getting large numbers of documents from diverse geographic locations to everyone involved in a product launch concerns any corporation whose business is global in nature. One of the main problems in developing such a system is the ramp-up time required to obtain the necessary tools and skills, particularly if this involves learning a new language.

My experience as a Web programmer prompted my management to request that I obtain all documents from the parties involved in the launch process, open and save the documents in Office 95 format and FTP the files to our server. I then had to code an HTML page to point to the individual files. As we all know, this process is an exercise in futility. After notifying all locations of the "hidden" URL, I'd begin accepting documents and post them to the URL. It didn't take much time to realize I was connected at the hip to this "little" maintenance job (which, incidentally, I wanted no part of).

I decided there had to be a better way. Although I have over 20 years of coding experience, I'd never coded in a GUI environment. Nevertheless, I was quite aware of what could be accomplished. Since laziness is the mother of invention (maybe a few of you can relate to this), I sat down and analyzed the situation, then drew up a spec on how this revision control system could be automated. Clearly I had to remove myself from the constant updating and refreshing of information.

With spec in hand I went to a colleague and asked if he could create a prototype in CGI or PERL script within a day or so. I was virtually laughed out of the office. At this time Rob, a young contractor working in my organization, suggested a product called ColdFusion. We discussed the spec – a secure area that only authorized individuals could access; and the need to upload data from an individual's desktop, automatically e-mail associates with a detailed comment and create an area with revision history. Rob researched ColdFusion, then came back to me the next day and said it was all possible. We picked up a copy of ColdFusion Server that evening and Rob had a prototype together in a matter of a few days. I was excited; we'd found a tool to implement my spec and I was saved from this never-ending, tied-to-the-hip task. Life was good. I could now turn this project around in a very short period of time and get on with my other duties.

After a week we were making significant progress and were able to demo the prototype to management on Friday. Management liked what they saw and agreed to fund the development as long as it could be accomplished in one month. A month? Ha! With Rob here and my ability to nudge a little extra effort, there was no problem.

Come Monday morning we were ready to really attack this project. Rob arrived, got his coffee and proceeded to give notice. Aargh! Now what? I certainly wasn't going to have time to implement this, along with all my other duties. After a short search, I found George Pandapas, a ColdFusion expert, and the rest, as they say, is history. In the course of the next couple of weeks, the system was functioning in a basic operational manner and by the end of the third month we had something appropriately named for its acronym: W.I.N., or Workstation Information Network.

W.I.N. – What Is It?

The Workstation Information Network is an automated revision control system designed to organize, store, authenticate and distribute the most recent documents and files pertaining to workstation products. The system provides a simple, effective, automated, worldwide, Web-based system that's available to appropriate workstation personnel. Before W.I.N., revision control was manual and chaotic. With the new system, the workstation team has access to the most recent documentation immediately after the person in charge of content creation has uploaded it. Various documents are neatly organized and centrally located. W.I.N. is dynamic and reflects changes immediately.

W.I.N. was designed to assist in maintaining revision control for projects such as new-product announcements and as a nondisclosure information repository. It allows authorized personnel to access the databases specific to the projects they're assigned to, and to view, update and upload new documents into the database. The system generates automatic e-mail to all participants of a specific project with information pertaining to the nature of the uploaded file and with a comment supplied by the individual who uploaded it. It's Web-based and easy to use. It enables a review of draft and final versions of documents, and is password-protected to safeguard sensitive information.

What Are Its Benefits?

Content providers for the workstation's business are ensured that all participants in a project are aware of the most up-to-date version of their document. It replaces the need to e-mail large files to a host of individuals who are in need of the information. It easily enables distribution of drafts for review to the appropriate areas and all Product Marketing and Developoment (PM&D) members, resulting in worldwide immediate access to updated documents.

Who Is the Audience?

The audience consists of content creators for the workstation business, primarily residing in the PM&D. Other users are geographical-area marketing personnel and sales team members. All users are notified of the availability of draft and final documents. It facilitates the review of documents before they are final, and allows for last-minute updates. Participants on a project can download, localize and re-post information locally on their server. Other PM&D members can benefit from W.I.N. by receiving final documents faster than before.

What's the Learning Curve?

A user is virtually an expert after one login. W.I.N. has been designed to be user-friendly, and is very functional in its simplicity. The main page used to enter the site acts as an introduction. From this page you can access the W.I.N. online instructions or guidelines, and move to the login screen to enter the site. All navigation options are presented via buttons at the top of the screen, under the W.I.N title banner.

The system requires each new user to create a unique password upon first entry to the site. As in all password-protected systems, it is important to keep in mind that the users be instructed as follows: Keep in mind that an administrator will never ask you for your password. Don't share it with anyone.

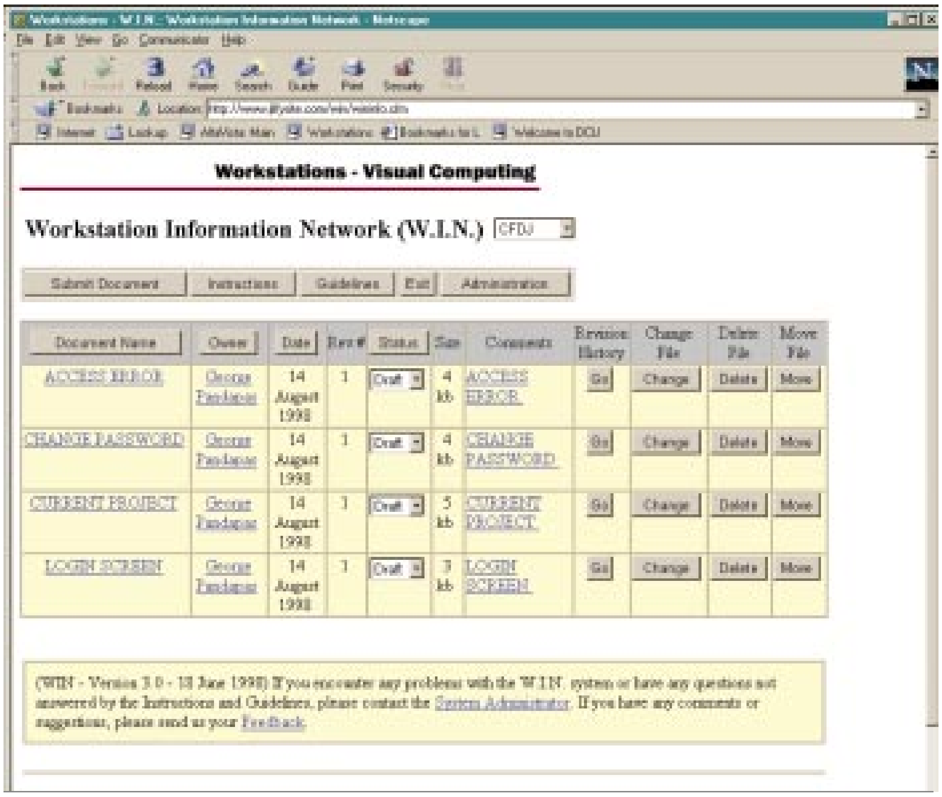
If an incorrect password is entered, the user is notified and directed back to the login screen via a meta refresh statement in HTML code.

Upon successful login, the user is requested to select the project they'd like to work on, if they're registered for more than one (see Figure 1). After your project is determined you'll be transferred to the main, "spreadsheet" page.

The main W.I.N. page lists the documents for a specific project. The document name is linked to the actual document file, and the user can view/download it by clicking on the link. The author's name is also linked. By clicking on this link you can send a comment to the author via e-mail. The status column contains pull-down menus listing whether the most recent document is a draft or final version. The document's author can change the version via a pull-down menu. The status pull-down menu appears for all documents for which the user is the author.

This routine has probably the most complex SQL statement in the system. The first task is to retrieve the latest version of each document in the currently selected project using a subquery involving the "revision" field, as noted below. Next, the document and author tables are joined to obtain the appropriate author information, except in the case of the "ARCHIVE" project, which we use to logically delete or deactivate documents that may be reactivated later. In the latter case we had to dynamically alter the WHERE clause to prevent the "disappearance" of archived documents because no authors are registered in this project. Next, we provide a sort by column feature by dynamically altering the ORDER BY clause, with an additional toggling effect achieved by checking the Client variable list for the column's sort parameter and reversing its sign if found (see Listing 1).

The spreadsheet page displays important information for each document. The first 30 characters of the comments associated with each document are displayed. Each comment is hyperlinked so that if it exceeds 30 characters it can be clicked to reveal the entire comment in a JavaScript-generated window. The revision history button will take you to the Revision History Page. The Date column displays the creation date in European format for its associated document. This is not necessarily the date the document was uploaded, but rather the date the document was originally created. The Revision column displays the document's latest version number, and the Size column displays the size of the document. Other visual features include highlighted rows for documents less than two days old and column headings that repeat every 10 rows so users always know what they're viewing as they scroll vertically.



Workstations - Visual Computing

Workstation Information Network (W.I.N.) CFDJ

Document Name	Owner	Date	Rev #	Status	Size	Comments	Revision History	Change File	Delete File	Move File
ACCESS ERROR	George Pandapas	14 August 1998	1	Draft	4 kb	ACCESS ERROR	Go	Change	Delete	Move
CHANGE PASSWORD	George Pandapas	14 August 1998	1	Draft	4 kb	CHANGE PASSWORD	Go	Change	Delete	Move
CURRENT PROJECT	George Pandapas	14 August 1998	1	Draft	5 kb	CURRENT PROJECT	Go	Change	Delete	Move
LOGIN SCREEN	George Pandapas	14 August 1998	1	Draft	3 kb	LOGIN SCREEN	Go	Change	Delete	Move

(WIN - Version 3.0 - 18 June 1998) If you encounter any problems with the W.I.N. system or have any questions not answered by the Instructions and Guidelines, please contact the System Administrator. If you have any comments or suggestions, please send us your Feedback.

Figure 1: The W.I.N. spreadsheet

The document upload form (see Figure 2) is used to upload documents to a predetermined location on the server's file system. The "Browse" button is included so users can conveniently locate the file on their local system. Clicking this button displays users' folders and files in a manner similar to the Windows Explorer interface, and permits them to click through the hierarchy of folders until they have located their document file. Once located, they upload it by simply selecting it and clicking the "Open" button on the navigation window. The browse window will then close and the file's path will be displayed in the "File To Upload" text field. The "Document Name" field contains a descriptive title associated with the file, which will appear on the W.I.N spreadsheet page. When uploading a revision of an existing document, the user must use the filename used previously. To assist in this task, there's a list of document and associated filenames on the right. This is a variable-length list with a maximum length of 10 rows (to stay within the dimensions of the HTML table). Listing 2 provides the code to generate this list.

The document upload acknowledgment page offers a summary of the document just uploaded. If the document file was already on the server, it's been overwritten by the new version submitted. The summary page template also sends e-mail to everyone on the distribution list associated with the project. Additionally, this template will handle duplicate filename conditions by continually prompting users to rename their file until it becomes unique. This condition occurs when the user attempts to upload a file with the same name as a preexisting file owned by another user. When you're not the owner of the existing file, W.I.N. won't permit you to overwrite it as a document revision. From this page the user has the option to navigate back to the W.I.N. page, which will now display all the information for the new document just uploaded.

Administration of W.I.N.

The W.I.N. administrator is responsible for ensuring that all users are added to the appropriate projects, and that their registration information is properly maintained. In addition to user administration, the administrator may create, change and delete projects. The administrator also has sole access to the following buttons on the W.I.N. spreadsheet page: Administration for entering the administration area, Change for modifying document information, Delete for deleting documents and Move for transferring documents to other projects, including the reserved "ARCHIVE" project for temporarily "deactivating" certain documents.

The Administrator can add, change or delete a single user, multiple users and projects. He or she can copy users from one project to another, list all users either in one project or in the entire system, and subsequently print these listings. The Administrator can turn everyone's e-mail off or on, and can edit the author table record by record.

To add a user, select a project and click on the Single User Add button (see Figure 3). The selected project will be highlighted in a list of all W.I.N. projects. The Administrator may change this selection and/or select additional projects.

Enter the user's first and last names, hit the Tab or Enter key, or click in another field to autogenerate the username, password and e-mail address values. Edit these values as necessary. Select a User Type and an E-mail Notification Level. Click on the Add button to add the user to the selected project.

For purposes of discussion, a uniquely defined W.I.N. user is someone who has a unique real name ("John Smith") and W.I.N. username ("smith") combination. Thus a W.I.N. project can contain two or more users with the same full name as long as their W.I.N. username is different.

If you select more than one project and the username you enter is already in use, you'll be asked to try another username. However, if you select a single project and someone with the same name and username already exists in that project, you'll be asked to cancel the Add request if this is the same person, or have W.I.N. generate a unique username if this is a different person.

If you're adding a user with the same last name but a different first name as an existing W.I.N. user, you may get a username conflict since the initial, autogenerated username is the user's last name. If this happens, you'll be asked if you want W.I.N. to generate a unique username for the new user, or if you want to cancel the Add request.

If the user you're adding already belongs to one or more other W.I.N. projects, their existing password and e-mail address, plus their user type and e-mail notification values, will overwrite the values entered in these fields. So don't bother editing these values when you're adding an existing W.I.N. user to a new project because they'll be overwritten by W.I.N. anyway. If you need to change one or more of these values, you may do so in a subsequent Change request.

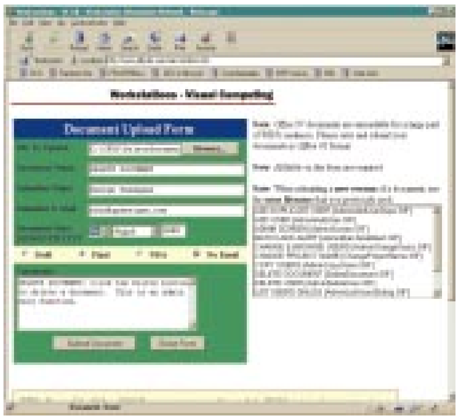


Figure 2: Document upload form

Yesler Software

www.yesler.com

If the user you're adding is a first-time W.I.N. user, the Add command will trigger an e-mail welcome message that explains the W.I.N. system and provides the new user with a W.I.N. username and password, which are initially the same. The user that already belongs to one or more W.I.N. projects won't receive a welcome message.

The default user type, User, can view all document versions in their assigned project, and can upload and change the status of their own documents. The Read Only user type can view all document versions but can't upload documents to W.I.N. The Field Rep user type can view final document versions only, but can't upload documents to W.I.N. or see any NonDisclosure documents identified as NDA. The Admin user type can move and delete documents; change the status of any document; and add, change and delete users and projects to W.I.N., along with other administrative functions.

If the e-mail notification level is set to "Send mail for revisions," the user is notified whenever a new or revised document is uploaded to the project. If the level is set to "Send mail for revisions and status changes," the user is also notified whenever the status of a document changes (from "Draft" to "Final" or from "Final" back to "Draft"). If the level is set to "Never send mail," the user won't receive any W.I.N.-generated e-mail except for the initial welcome message if they're a new W.I.N. user.

The Single User change command looks similar to the Single User Add.

To change a W.I.N. user's registration information:

- Select a user name.
- Click on the Single User Change button.
- Edit one or more user values.
- Click on the Change button to change the user's registration information for the selected project, or click on the Cancel button to cancel the change request.

If the user already belongs to one or more W.I.N. projects, their revised name, username, password and e-mail address will be copied to their other project registrations. However, any changes made to their user type and e-mail notification values will be restricted to the selected project only. Note that the user's document records will also reflect any changes made here; i.e., the user's name and e-mail address will be updated in every document record they own.

A user's project name may be changed. If it's changed to the name of an existing project to which they don't belong, they'll be added to that project and deleted from the current project. If it's changed to the name of an existing project to which they belong, they'll be deleted from the current project. If the new project name doesn't exist, the project will be created with the user as its first member, and the user will be deleted from the current project. If the user is the only member of the current project, the effect of this change will be to delete the project itself. In all cases, whatever documents the user owns under the current project will be moved to the new project.

Select a user name and click on the Single User Delete button. Then click on the Continue button in the confirmation dialog box. If the user belongs to one or more other W.I.N. projects, you'll be asked if you want to delete them from all projects. The default is to delete them only from the project you selected.

If the user has authored one or more W.I.N. documents, you'll be asked if you want to delete the documents from W.I.N. or transfer ownership to another W.I.N. user. If you elect the latter option, you'll be asked to select the new owner from a list of all W.I.N. users. If a transferred document is in a project that the new owner doesn't belong to, W.I.N. will automatically register them for this project so they'll be able to update the document as required. The new owner's registration information will be copied from one of their existing project registrations.

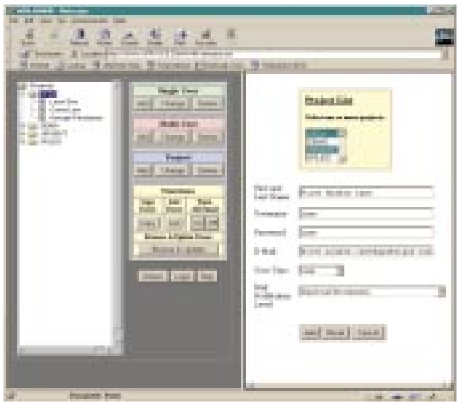


Figure 3: Single User Add

To add more than one user at a time to a W.I.N. project:

- Select a project name.
- Click on the multiuser Add button.
- Deselect the "Send Welcome" e-mail option if you don't want to send e-mail welcoming new users to W.I.N. The default is to send e-mail, primarily because this is how new users obtain their W.I.N. username and initial password.
- Type or paste one or more user names, at one per line, into the text area.
- Click on the Add Users button.

From the user's full name W.I.N. automatically generates the username, password and e-mail address. If the username already exists within the selected project, W.I.N. modifies it until it becomes unique within all of W.I.N. If this happens, W.I.N. displays a list of all users whose usernames were modified, giving you the option of adding (the default) or skipping each individual user. Click the Submit button to either add or skip the users in this list.

If one or more users were skipped, they'll be listed in a subsequent dialog box. This dialog is informational only – no action is required other than clicking the Return button. However, you might want to print it out so you'll have a record of who wasn't added to the selected project. Up to two sections may be displayed: one for users you skipped (by clicking the "skip" checkbox in the previous dialog), the other for those who were skipped by W.I.N. because they appeared to be duplicate users (their first and last names, as well as their username, matched those for users who are already members of the selected project). In the latter case, if a duplicate user is in fact a different person, you may add them to the project with the Single User Add command described above.

To add a new project to W.I.N.:

- Select the root project name, Projects.
- Click on the Project Add button.
- Enter the new project name in the Project field.
- Enter a user's first and last name. The first user in a new project will usually be the Administrator (you).
- Hit the Tab or Enter key, or click in another field to autogenerate the username, password and e-mail address values.
- Edit these values as necessary.
- Select a User Type.
- Select an E-mail Notification Level.
- Click on the Add button to add the new project to W.I.N.
- Click on the Cancel button to cancel the add request.

To change the name of a W.I.N. project:

- Select the project name.
- Click on the Project Change button.
- Enter the new project name.
- Click on the Change Project button.

To copy users into a project:

- Select the project name.
- Click on the Copy Users button.
- Select one or more user names.
- Click on the Copy Users button to copy these users (i.e., import their registration information) into the selected project. W.I.N. won't copy a user who already exists in the selected project.

To list W.I.N. users:

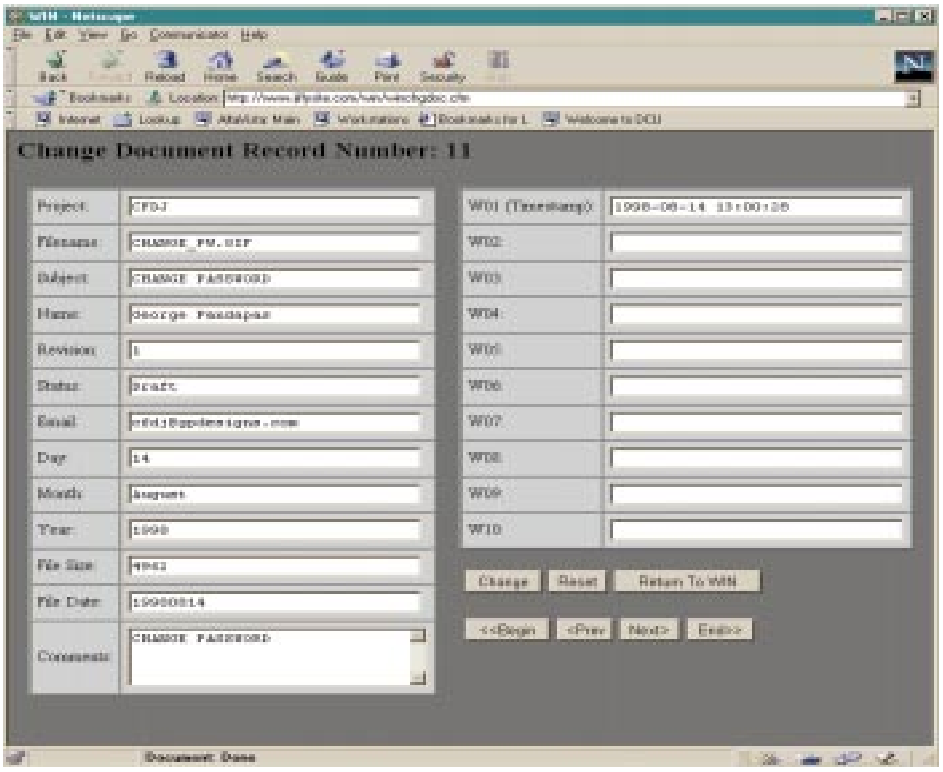
- Select the root project name, Projects, to list all W.I.N. users, or select a single project name to list only its users.
- Click on the List Users List button.

- Click on the Not E-mailed radio button if you want a listing of only those users who don't require e-mail notification of document revisions or status changes.
- Click on the List Users button.
- Optionally print the user listing.
- Click on the Clear button to erase the listing.

To turn all e-mail notification either on or off:
- Select the root project name, Projects. Click on the Turn All E-mail On (or Off) button. This will turn e-mail notification either on or off for all W.I.N. users. It's recommended that you turn all e-mail off before performing test uploads or status changes within various W.I.N. projects so those users won't receive the e-mail generated by your tests.
- Use the List Users command to list users who already don't receive e-mail notifications, so you'll know whom to turn off when you turn all e-mail back on.

To facilitate database maintenance, the system is designed with a browse and update facility for the Author and Document tables. You can access the Author table by clicking the "Browse & Update" button on the administration screen. This displays the entire record, including fields reserved for future use. You can go to the first, previous, next or last record in the Author table by clicking the appropriate navigation button.

Invoking the routine for browsing and updating the Author table required a little programming finesse that combined ColdFusion and JavaScript capabilities in some interesting ways. First, we found it necessary to use JavaScript to escape the frame we were in so we'd have a full window in which to display our record data and related controls. Generally, this is accomplished by specifying a TARGET attribute with a value of "_top". However, ColdFusion's <CFLOCATION> tag doesn't support this attribute, so using this tag would confine our template output to the right-hand frame of the admin frameset. The HTML <FORM> tag supports the TARGET attribute, but we didn't want to interrupt the program flow with a form simply to obtain a larger display area. This left us with the JavaScript window location property that not only can load a URL but can also specify a target via the "top" window property (see Listing 3). Another nice aspect of this code is how we used ColdFusion to retrieve a database value (the record number for the selected user and project) and pass it to our template by pasting it into our JavaScript code using ColdFusion's <CFOUTPUT> tag. A client variable would have been another way of passing this value, but this technique for dynamically generating client-side code (JavaScript) with ColdFusion can be helpful in many other ways as well.


WIN - Netscape
File Edit View Go Communicator Help
Back Forward Reload Home Search Guide Print Security Stop
Bookmarks Location: http://www.[illegible].com/win/winchgdoc.cfm
Internet Lookup AltaVista Main Workstations Bookmarks for L Welcome to DCU
Document: Done

Change Document Record Number: 11

Project:	CFDJ	W01 (Timestamp):	1998-08-14 13:00:28
Filename:	CHANGE_PW.GIF	W02:	
Subject:	CHANGE PASSWORD	W03:	
Name:	George Pandapas	W04:	
Revision:	1	W05:	
Status:	Draft	W06:	
Email:	[illegible]	W07:	
Day:	14	W08:	
Month:	August	W09:	
Year:	1998	W10:	
File Size:	4962		
File Date:	19980814		
Comments:	CHANGE PASSWORD		

Change Reset Return To WIN

<<Begin <Prev Next> End>>

Figure 4: Browse & Update Document table

The remaining administration functions are accessed from the spreadsheet. They are the Change, Delete and Move functions assigned to the Document table.

In the Document table (see Figure 4), as was the case for the Author table, you'll notice that this is the entire record, including 10 fields reserved for future use. You can go to the first, previous, next or last record in the table by clicking the appropriate navigation button.

The ColdFusion code for implementing these navigational controls has some interesting features that may be helpful in your own coding work. First, as with our column sort buttons earlier, the navigation buttons are submit buttons in a form that calls itself as part of the containing template, and with a hidden field value set to the current record number. Upon reentering the template, the code detects whether this is a recursive call (the hidden field parameter, "oldrec", will exist if it is) and produces a sorted list of record numbers if true. By detecting which navigation button was clicked, we can determine which record to display. For the "previous" and "next" buttons this involves decrementing or incrementing the ordinal of the previous record number to find the new record number. The final key step is to specify this value for the STARTROW attribute in the <CFOUTPUT> tag, and to be sure to limit output to only one record by setting the MAXROWS attribute to one (see Listing 4).

The administrator is the only person allowed to delete a document. This normally occurs as the result of an e-mail request from a user who has erroneously uploaded a document, or it may be that a document is simply outdated and needs to be deleted. By clicking on the Delete button for a specific document listed on the spreadsheet page, you may physically delete the record from the database

As mentioned earlier, W.I.N. documents can be moved from project to project. New users will generally use a generic project to begin their work, and as projects are defined they will identify which documents actually belong in different projects. Transferring these documents to their designated projects is the primary reason for having the Move command. It's also used by the administrator for storing requested deletions in a holding area ("ARCHIVE") in the event that the document and the history need to be reviewed in the future.

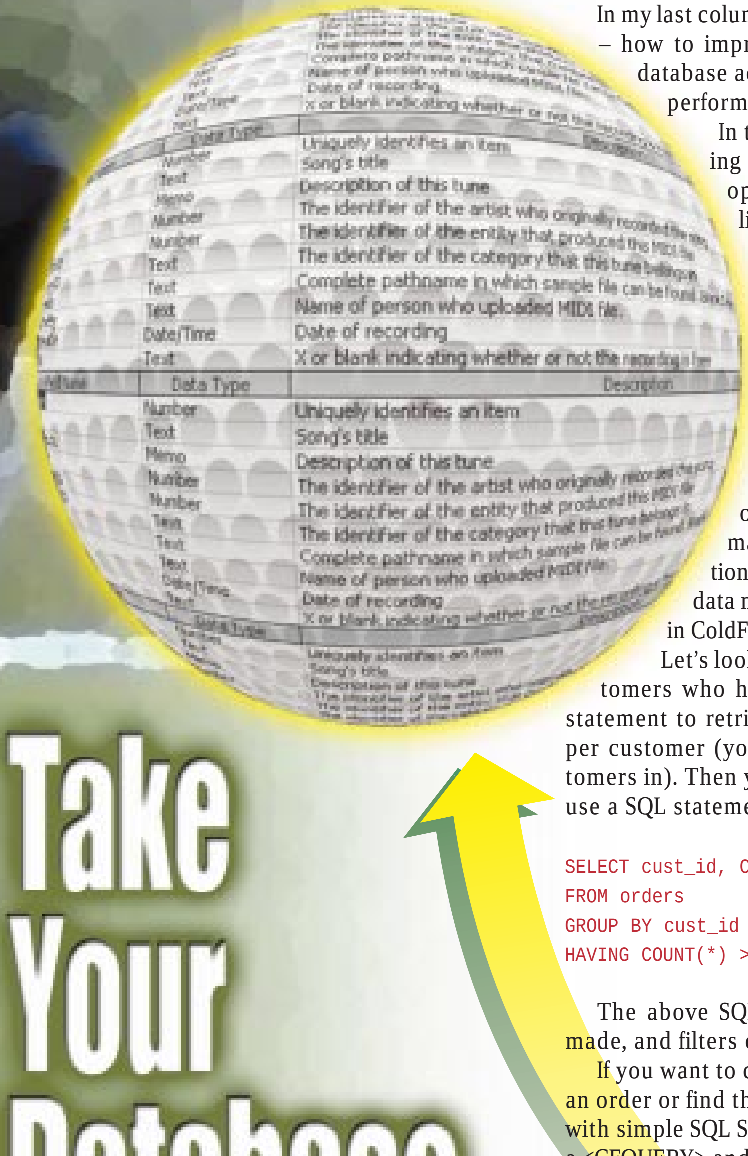
CODE LISTING

The complete code listing for this article can be located at **www.ColdFusionJournal.com**

About the Author

Leon Eno, Webmaster for CVSI, Inc., has over 20 years of proactive experience in information technology. His special focus is on developing and supporting sales/marketing and financial systems. Leon also freelances in the areas of Web graphics, prepress processing and Web design, and programming. He can be reached at enol@bigfoot.com.

enol@bigfoot.com



Take Your Database Out of Retirement

by Ben Forta

In my last column (***CFDJ*** Vol. 1, Issue 2) I discussed database query caching – how to improve application performance by eliminating unnecessary database access. While caching query results will improve application performance, optimizing your application doesn't stop there.

In this column I'd like to continue with the subject of improving performance by improving the efficiency of database operations, but from another perspective. Specifically, I'd like to take a look at how databases should (and shouldn't) be used, and at database features that can be used to enhance application performance.

It's worth noting that some of the features and technologies discussed here (particularly items 3–5) are available only in client/server databases – which is all the more reason you should be using one.

1. Don't Do the Database's Job

This should be obvious, but I see this rule violated over and over. Database software is designed to do one thing only – manipulate data. All databases support a set of basic operations to manipulate data with. And almost always, database-level data manipulation code will outperform manipulation code written in ColdFusion (or any other client language for that matter).

Let's look at a simple example – you need to retrieve a list of all customers who have placed more than one order. You could write a SQL statement to retrieve all orders, and then loop the results to count orders per customer (you'd also need some sort of array or list to store the customers in). Then you'd filter out the ones with only one order. Or you could use a SQL statement that looks something like this:

```
SELECT cust_id, COUNT(*) AS num_orders
FROM orders
GROUP BY cust_id
HAVING COUNT(*) >= 2
```

The above SQL returns a list of customers and the number of orders made, and filters out only the customers with two or more orders.

If you want to count records that match a specific criterion, total items in an order or find the highest value in a set, you can perform these operations with simple SQL SELECT statements. Sure, you can retrieve the records with a <CFQUERY> and then loop through the results to perform the calculations yourself. But why would you want to? All you're doing is making ColdFusion work harder than it should, while forcing your database server to sit idle.

Any time you find yourself looping through result sets to perform calculations, passing the results of one query to a second query, or maintaining long lists of variables that contain running totals or other calculations – take a step back, and determine if you're doing the database's job. That's something you never want to do.

It's worth noting that more often than not, ColdFusion developers violate this rule because they're having a hard time with the SQL needed to perform the specific operations (be it the aggregate functions, joins or the correct use of subqueries). And while re-creating the wheel in ColdFusion works well as an interim solution, it's going to create performance and scalability problems down the road. My advice is to grab a good SQL book and master the language (try *Sams Teach Yourself SQL in 24 Hours*, or wait for my own soon-to-be-released *Sams Teach Yourself SQL in 10 Minutes*).

2. Enforce Data Constraints

An extension of the above is employing database-level constraints. Constraints are simply rules that govern the validity of data at the database level. Common uses for constraints include:

- Making sure required fields have values when records are inserted or updated

BackSoft

www.backsoft.com

- Ensuring that a customer ID used in an order record actually points to a valid customer record
- Forcing minimum or maximum values on specific fields
- Defining default values for specific fields

Data constraints are used to enforce data integrity, and employing constraints of some kind is an important part of application design. Many ColdFusion developers write code to enforce integrity – they validate form fields against databases (using a <CFQUERY> tag) before inserting them into tables; they write complex <CFIF> statements to check for valid values, and then convert data on the fly using lists of <CFSET> statements.

Once again, this is a task best left to the database. Every decent database allows you to define constraints. From simple rules that establish default values or restrict values to those in another table (foreign keys) to more complex rules that define sets of valid values under specific conditions.

Here's a practical example, and one I see asked repeatedly on the Allaire Developer's Forum. You're inserting a record into a table, and you want to save the current date and time along with the record in a timestamp field. Sure, you could play with the ColdFusion Now() and CreateODBCDateTime() functions, but why would you want to? All you need to do is instruct the database to use the current date and time if no explicit value is provided, and then provide no value!

So why is this validation better suited for the database level rather than at the application level? There are two primary reasons:

1. ***Performance*** – The database can process these rules far faster than your application will be able to.
2. ***Data integrity*** – More often than not, your code will not be the only application to access the data. Embedding rules at the database level ensures that all applications use the same set of rules rather than requiring you to re-create the rules in each application and run the risk of missing them in one, or making a mistake in them.

In the past ColdFusion developers were reluctant to rely on database-level validation because if the validation rule failed (meaning invalid data was provided), the <CFQUERY> tag would generate an error and page processing would abort. This was a valid objection prior to the release of ColdFusion 4.0, but not anymore. ColdFusion 4.0 introduces "try-catch" error handling (see the <CFTRY> and <CFCATCH> tags in the ColdFusion documentation), which allows you to programmatically control what happens once an error occurs. This makes it possible to respond to database-generated validation errors as you see fit.

<CFQUERY>	<CFSTOREDPROC>
• Easy to use	• More complex usage
• Best suited for simple stored procedures	• Best suited for more advanced stored procedures
• Can only return a single result set	• Can return multiple result sets
• Supports input parameters only	• Support for input and output parameters
• No support for return codes	• Support for return codes

Table 1: <CFQUERY> vs. <CFSTOREDPROC>

3. Use Stored Procedures

Stored procedures are blocks of SQL code that are stored on the database server – similar to custom functions (or ColdFusion Custom Tags). Using stored procedures, you can group sets of related SQL statements together as a single unit. Instead of calling each SQL statement individually and passing values retrieved in one statement down to the next, you can simply call a stored procedure that contains all the needed statements.

The fine points of stored-procedure use probably require their own column, but for now it's worth noting the primary advantages of using stored procedures:

- ***Performance*** – Stored procedures execute faster than passed SQL.
- ***Enhanced capabilities*** – There are SQL language elements that can't be used in sequential SQL calls (e.g., creating variables, working with temporary tables).
- ***Security*** – Requiring that tables be accessed only via stored procedures (and never directly) can provide added security where needed.

ColdFusion fully supports the use of stored procedures. In ColdFusion 4.0 you now have two different methods with which to invoke stored procedures: as simple SQL passed to a <CFQUERY> tag, and via the new <CFSTOREDPROC> tag family. The detailed differences between the two are beyond the scope of this column, but Table 1 lists the basic advantages of each.

There is no hard and fast rule about where to use stored procedures and where not to, but here's one place to start: if you ever find yourself calling a <CFQUERY> and then passing the results of that query to another <CFQUERY> (perhaps using the ValueList() or QuotedValueList() functions), that code should probably be a stored procedure.

4. Be Trigger-Happy

All client/server databases support triggers. A trigger is simply a block of SQL code that is executed when specific events occur. Triggers are associated with specific operations on specific tables. For example, you might create a trigger that is called whenever a SQL INSERT or UPDATE operation is performed on a specific table.

An ideal use of triggers is data cleanup. For instance, if you want state abbreviations always inserted in uppercase you can write a simple trigger for INSERT and UPDATE operations that will replace the passed state abbreviation with an uppercase version.

Of course, not only will this be faster than doing data conversion at the ColdFusion level, but it also will allow all client applications (including ColdFusion) to share the same code – and it all happens transparently.

5. Schedule Tasks

Most databases allow you to schedule the execution of SQL statements. ColdFusion also allows you to schedule tasks by using <CFSCHEDULE> or the ColdFusion Administrator. The ColdFusion scheduling engine is very useful for scheduling execution of ColdFusion code, but it should never be used to schedule the execution of SQL statements themselves unless you need to do processing on the query results.

For example, if you need to execute daily SQL cleanup code (to perform database cleanup), or if you need to move data from a live table to an archive table on a regular basis, or if you need to run nightly calculations on one table and write the results to another – all of these are best handled by the database's own scheduling engine.

The rule here is to use ColdFusion scheduling to schedule ColdFusion processing and use database scheduling to schedule database processing. It makes absolutely no sense to tie up ColdFusion when it's the database doing the actual work.

Conclusion

The bottom line here is that database developers put a lot of time and effort into optimizing their products. Rather than reinvent the wheel, you should strive to take advantage of their work whenever possible. Most ColdFusion applications are running against powerful database servers that are not working as hard as they should. Push your database server – make it work as hard as it was designed to. Doing so will dramatically improve application performance – and it'll likely save you time too. CFDJ

About the Author

Ben Forta is Allaire Corporation's product evangelist for the ColdFusion product line. He is the author of the best-selling ColdFusion 4.0 Web Application Construction Kit *and the new* Advanced ColdFusion 4.0 Application Development *(both published by Macmillan).*

ben@forta.com

Data Return

www.datareturn.com

Streamlining Data Entry Applications

Standardize your maintenance techniques to make your life easier

by **Leon Chalnick**

You may not believe this, but you'll probably spend more time maintaining your ColdFusion application once it goes into production than you spent developing it in the first place. Among IT professionals it's well known that the effort required to support applications over time is significantly larger than the effort required to build them. Applications are like children: the effort and dollars required to bring them into the world are sizable, but once they're here you'll support them for years. Thus it behooves you to take measures to minimize the effort required to maintain your applications. But how do you do this? One of the most effective solutions is to develop standard techniques and approaches for handling similar problems.

This article presents a compelling, reusable framework for data entry applications. In relational database applications, data entry applications usually involve mechanisms for reviewing, creating, editing and deleting records in tables, so that's what our framework will support. You will, of course, find situations in which this framework needs to be altered somewhat because your requirements are unusual. In many situations, however, this framework is entirely workable.

Data Entry Framework

First, let's look at the framework (see Figure 1). The basic idea is simple: five templates are used to create an application. You create new versions of four of the templates for each data entry app, but you use the same techniques from one application to the next. One template, the Delete prompt (DeletePrompt.cfm in our examples), is reused, as is, in each of your data entry applications. I prefer to name my templates like this:

- AppNameList.cfm
- AppNameForm.cfm
- AppNamePost.cfm
- AppNameDel.cfm

AppName is replaced by the name of the data entry application, e.g., MaintainOrdersList.cfm, MaintainOrdersForm.cfm, MaintainOrdersPost.cfm, etc. The templates always call each other and are related to each other the same way.

As shown in Figure 1, each of the five templates plays the same role in every data entry application. The list template, for example, is always responsible for presenting a tabular listing of records of a particular type. It is always responsible for calling your form template and your delete template. The form template, for example, allows users to add or edit individual records and will always call your post template. As each template always provides the same function, by following these naming conventions, your application will become easier to maintain because you'll be able to tell instantly what each template is responsible for.

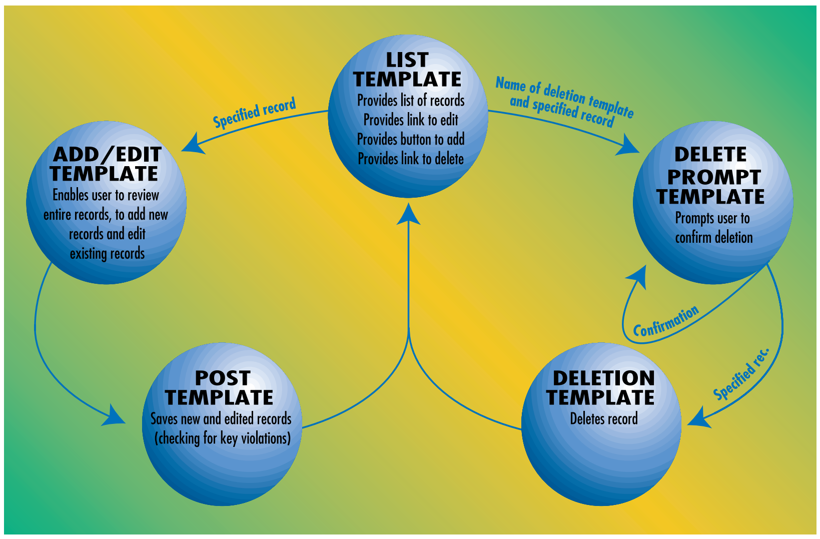


Figure 1: The framework and how the templates are interrelated

STOP! Use URLEncodedFormat()

It is imperative that you use URLEncodedFormat() when you're building the values for URLs dynamically (as in Listing 1). You usually can't be sure that the text value contains only text and numbers. If it contains punctuation or spaces, then it needs to be URL-encoded in order to be passed through the URL correctly. This is particularly applicable when the dynamic value is text that comes from a database. You just don't know what's stored in the database so you should get in the habit of using URLEncodedFormat() to pass values.

Last July I spoke at the ColdFusion User's Conference in Fort Collins, Colorado. One attendee suggested that the logical extension of my approach is to put everything in one template. I can't recommend this idea, based on tenets of structured programming. The notion is that each module (or in the case of ColdFusion, each template) should be focused on providing one function. When a module performs too many functions (e.g., listing, adding, editing, deleting, saving), its internal logic becomes convoluted and difficult to follow. From the perspective of making applications easy to maintain, there is no benefit to using one template to do everything.

One Form for Adding and Editing

New CF users usually develop one form for creating new records and a separate form for editing records. This makes your application harder to maintain. Why? Because whenever there's a change to some aspect of your data entry field, you'll need to change it in two places – the add and edit templates. The more places you have to fix something, the more likely it is you'll make a mistake. In my framework I need only one template for adding and editing. The framework includes a nifty trick that enables you to use one CF template for both adding and editing records.

Generic Deletion Prompt

Virtually every data entry application I've ever written in which users could delete records required that they be prompted to confirm their request to delete a record. That's the purpose of the reusable template DeletePrompt.cfm. This template is entirely generic and can be reused from one application to the next. Its inner workings are detailed below.

List Template

A common starting place for desktop data entry apps is a tabular display that the user can browse through. The listing lets the user review the database records for the type of entity involved in the data entry application and is used as a starting point in the framework. Keep in mind that there's room for more diversity and creativity in Web-based applications than in traditional desktop apps. You don't have to start your applications with a "list template" but it's convenient. The framework is flexible enough that this template can be altered significantly or eliminated altogether. If you eliminate it, you'll need some other way of providing the list template's links with other templates in the framework.

The list template (see Figure 2) provides not only a tabular listing for the user's perusal but also:

- A button for adding new records
- Links to edit records
- Links to delete records

My example consists of a Web-based catalog of music files that have been saved in the MIDI format. The catalog of tunes is stored in the Catalog table, which is the focus of our example data entry application. Figure 3 provides a listing of the Catalog table structure.

Several fields are foreign keys to other tables. The ArtistID field points to records in the Artists table. Each entry in Artists describes a recording artist. The ProducerID field points to records in the Producers table (which describes producers of MIDI titles). The CategoryID field points to records in the Categories table. This table lists musical genres or categories (e.g., classical, hip-hop, jazz). The Categories table groups the entries in the Catalog table in a logical order. Each category is assigned a "sort position" (in the Categories table) indicating the order in which the Categories are to be displayed in the list page.

The code for the list template can be found in Listing 1. Note that the Categories table is joined to the Catalog table because I want to display the complete category name (only the CategoryID is stored in the Catalog table) and because I need the SortPosition field for grouping.

In this framework the list template always contains a form with a button for adding new records, as in this code snippet:

```
<FORM ACTION="MaintainMIDIForm.cfm"
METHOD="post">
<P><INPUT TYPE="submit" NAME="Add"
VALUE="New Listing">
</FORM>
```

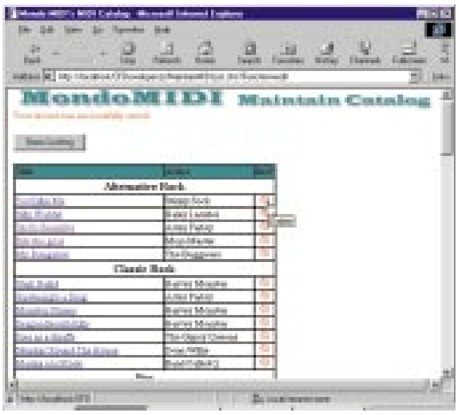


Figure 2: The list template

The list template always lists the existing records, usually in tabular format (see the HTML table construction with CFOUTPUTs in Listing 1). The template always provides a link (i.e., an HREF anchor) that takes the user to a form for editing existing records. The list template also provides a link to delete records. If you need to prevent records from being edited or deleted, leave out the links. In other cases you may want the user to be able to view the entire record through your editing form, but not be able to edit it (i.e., you want to provide a read-only view of the data). I'll demonstrate how to do this later in the article.

The HREF for invoking the editing form is pretty straightforward (from Listing 1):

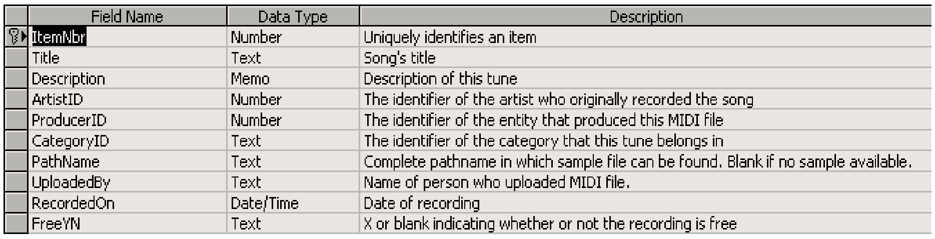
```
<A
HREF="MaintainMIDIForm.cfm?Edit=#URLEn-
codedFormat(ItemNbr)#">#Title#</A>
```

You pass the value of the key field to the editing template (in my example this is named MaintainMIDIForm.cfm) via a URL parameter named Edit.

The HREF, which I build for deletion, is a bit more complex.

```
<A HREF="DeletePrompt.cfm?Val=#URLEncod-
edFormat(Title)#&RT=Catalog&DURL=#URLEn-
codedFormat('MaintainMIDIDel.cfm?I=#ItemN
br#')#"><IMG SRC="../g/x.gif" WIDTH=12
HEIGHT=12 ALT="Delete" BORDER=0></A>
```

As you may recall, a generic template, DeletePrompt.cfm, is used to prompt the user to confirm his/her intent to delete the record. You pass DeletePrompt.cfm the URL for the template you've produced to do the actual deleting of the record in the DURL parameter. In the previous code snippet DURL is set to 'MaintainMIDIDel.cfm?I=#ItemNbr#'. My deletion template is named MaintainMIDIDel.cfm. I'm passing it one parameter named "I" through its URL. The "I" parameter is set to the value of the ItemNbr field in the Catalog. In other words, I'm identifying the item to be deleted. I'm also passing two other optional parameters to the DeletePrompt.cfm template, Val and RT, explained below. Let's take a look at how the process of deleting works.



Field Name	Data Type	Description
ItemNbr	Number	Uniquely identifies an item
Title	Text	Song's title
Description	Memo	Description of this tune
ArtistID	Number	The identifier of the artist who originally recorded the song
ProducerID	Number	The identifier of the entity that produced this MIDI file
CategoryID	Text	The identifier of the category that this tune belongs in
PathName	Text	Complete pathname in which sample file can be found. Blank if no sample available.
UploadedBy	Text	Name of person who uploaded MIDI file.
RecordedOn	Date/Time	Date of recording
FreeYN	Text	X or blank indicating whether or not the recording is free

Figure 3: Catalog table structure

Deletion Template

The list template invokes the generic DeletePrompt.cfm template, which asks the user to verify that he or she wants to delete the specified record. (See Figure 4 and Listing 2.) If the user says "No," I return to the list template. If the user confirms the deletion, I want to run the template that actually does the deleting. Note that this form calls the DeletePrompt.cfm template recursively to process the user's selection.

You write the template that does the deleting. This can't be done generically, because the process of deleting records can vary quite a bit from one data entry application to the next. As pointed out above, you pass the URL of your deletion template to DeletePrompt.cfm in the DURL parameter (see Listing 1).

DeletePrompt.cfm includes two optional URL prameters that you'll find useful. The RT URL parameter allows you to specify the type of record being deleted. If you specify an RT, it will be included in the prompt presented to the user. I'm deleting Catalog records, so RT has been set to "Catalog" in the URL for DeletePrompt.cfm. The second optional parameter, Val, lets you provide a value that'll be presented at the end of the prompt to identify the specific record being deleted. I've set Val to the song title from the catalog. Again, using this parameter helps to further customize the deletion prompt so the user is clear about what is about to be deleted.

If the user doesn't want to delete the record, I return to the list template without invoking the deletion template. This can be done through JavaScript (as in the code provided) or through a regular form submission. If you don't want to use JavaScript, you can make it work through a form submission by changing the input object from type "button" to type "submit" and removing the onClick="history.back();" JavaScript code from the button. Doing so causes the form to be submitted for processing on the server when the user clicks the No button. If the No button exists when the DeletePrompt.cfm is called recursively, the user is redirected (through CFLOCATION) to the URL stored in the CT field, which is built dynamically when the form is first displayed. It contains the URL of the template that invoked the DeletePrompt.cfm in the first place, i.e., the list template. Note that I'm simply setting CT to the value of the CGI variable HTTP_REFERER:

```
<INPUT TYPE="Hidden" NAME="CT"
VALUE="#CGI.HTTP_REFERER#">
```

What happens if the user *does* want to delete? Again, the DeletePrompt.cfm form invokes itself to process the form submission. This time, however, the Yes input object will exist (the user clicked the Yes button, right?). In this case processing is simply redirected to the URL that was passed to DeletePrompt.cfm in the DURL parameter.

```
<CFELSEIF IsDefined("Form.Yes")>
   <CFLOCATION URL="#Form.DURL#">
```

This URL value was stored in the hidden field that we also named DURL.

```
<INPUT TYPE="Hidden" NAME="DURL"
VALUE="#URL.DURL#">
```

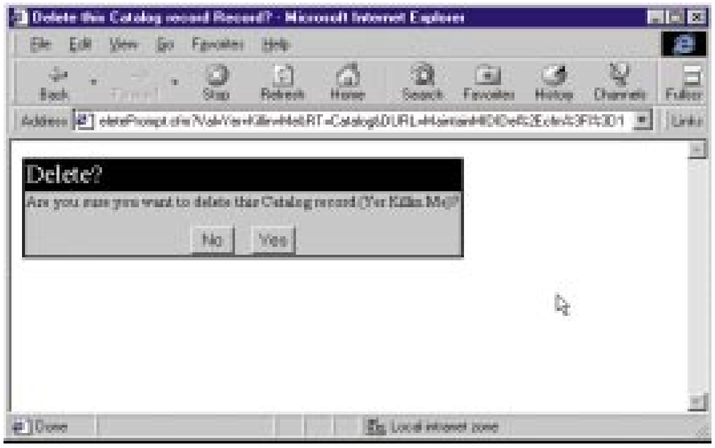
Remember that this is the URL to your custom template for executing the deletion. Here's the deletion template used in this example (the file name is MaintainMIDIDel.cfm):

```
<CFQUERY NAME="DeleteItem"
DATASOURCE="#DataDB#">
   DELETE FROM Catalog WHERE
ItemNbr=#URL.I#
</CFQUERY>
<CFLOCATION
URL="MaintainMIDIlist.cfm?Function=delete
">
```

I defined one parameter, URL.I, to pass the ItemNbr to be deleted. Where did the URL.I come from? From the URL that was created back in the list template. After the record is deleted, I redirect the user to the list template. The deletion templates you create for your applications will execute different code, obviously, but they'll be structured to do the same thing – delete the specified record(s) and direct the user back to the list template. (The Function=delete query string on the URL will be explained shortly.) Let's see how we can use one form to add new records and edit or view existing records.

Form Template

The form template presents a formatted view of an individual record. (See Listing 3 and Figure 5.) The data entry form you use to add records doesn't need to be very different from the form you use to edit records. In some applications you may need to prevent the user from editing key field values (once they've saved the record). There may be some other minor differences too, but



Delete this Catalog record Record? - Microsoft Internet Explorer

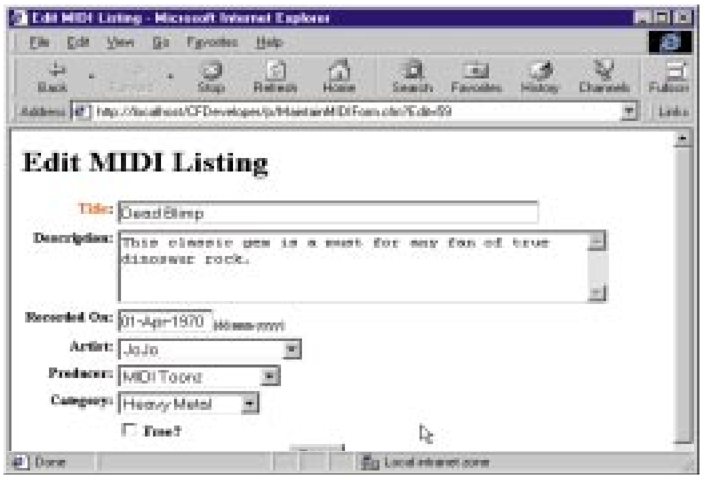
Address: eletePrompt.cfm?Val=Yer+Killin+Me&RT=Catalog&DURL=MaintainMIDIDel%2Ecfm%3FI%3D1

Delete?

Are you sure you want to delete this Catalog record (Yer Killin Me)?

No Yes

Figure 4: The DeletePrompt.cfm template asks the user to confirm an intent to delete a record.



Edit MIDI Listing - Microsoft Internet Explorer

Address: http://localhost/CFDevelopers/js/MaintainMIDIForm.cfm?Edit=59

Edit MIDI Listing

Title: Dead Blimp

Description: This classic gem is a must for any fan of true dinosaur rock.

Recorded On: 01-Apr-1970 (dd-mmm-yyyy)

Artist: JoJo

Producer: MIDI Toons

Category: Heavy Metal

☐ Free?

Figure 5: Form template

Live Software

www.cfanywhere.com

usually the forms function similarly. Using two forms is more than a waste of effort; the real problem occurs at some point in the future when you're asked to change the way processing works. Perhaps a field should be removed or added. With two forms you'll have to go to two places to make the same adjustment and this increases the chances of making an error or omission. Using a third form for presenting a read-only view of the individual record makes things even worse.

Let's take a look at how the list template invokes the form template when a new record is to be added. A simple form at the top of the list template contains one submit button named Add. While the list templates you create for your applications won't look the same as this one, the approach for invoking the form template is the same.

```
<FORM ACTION="MaintainMIDIForm.cfm"
METHOD="post">
<P><INPUT TYPE="submit" NAME="Add"
VALUE="New Listing">
</FORM>
```

If you're editing an existing record, the form template is called through a URL that includes a parameter named Edit, which is set to the key value of the record to be edited. A little chunk of code near the top of the form template is used to determine the context in which the template is being called:

```
<CFIF IsDefined("Form.Add")>
   <CFSET Function="add"> ·
<CFELSEIF IsDefined("URL.Edit")>
   <CFSET Function="edit"> ·
```

So it's the presence of one of these two parameters that enables the template to determine the context in which it was invoked. It then instantiates the Function variable with either "add" or "edit". This little chunk of generic code should be used in your form templates.

Your data entry forms are composed of HTML form elements. When you're editing a record, you need these elements to reflect the values in the existing record. This is typically done by querying the desired record from the database and creating a variable (with the value from the database) for each field that the users are allowed to edit. These variables are then used to populate the values of the form's INPUT, TEXTAREA and SELECT elements.

But what about a new record? There are no values that you need to populate the fields with (unless, perhaps, you need some fields populated with default values). Should you build another set of logic to handle the HTML form elements when working on new records? There's no need to – the framework includes a reusable technique for this.

Do a "SELECT * FROM..." query to retrieve a record from the table you're editing containing all the fields in the table. (It doesn't have to retrieve all the fields, but it should return the fields needed to add or edit records.) When you're working on an existing record, you've passed the key of the record to be retrieved to the form template via the URL.Edit parameter. When you're working on a new record, create a "WHERE" clause that returns an empty data set. In both cases, after you run the query you have a CF query object you can use to derive the field names of all the fields in the record. Here's the query from my example:

```
<CFQUERY NAME="GetRec"
DATASOURCE="#DataDB#">
   SELECT * FROM Catalog WHERE ItemNbr =
<CFIF Function IS "add">-1
   <CFELSEIF Function IS
"edit">#URL.Edit#</CFIF>
</CFQUERY>
```

If I'm adding a new record, we query for the record where ItemNbr is –1. I do this because I know there are no records with –1 as the item number. Thus I'm using one query to deal with both new and existing records.

In my example the query I'm running is named GetRec. The following code snippet creates a variable named after each field. The value will be set to the value in the underlying record. If the record is null – as it is with a new record – the variable will have a null value.

```
<CFLOOP LIST="#GetRec.ColumnList#"
INDEX="ii">
   <CFSET x = #SetVariable( ii, Evalu-
ate("GetRec." & Evaluate("ii")))#>
</CFLOOP>
```

After this code executes, I'm assured of having one variable for each field in the table. I can now use these variables to populate the INPUT, TEXTAREA and SELECT objects in our form regardless of whether I'm working on new or existing records (see Listing 4).

Note the trick employed in the form template with the checkbox input named FreeYN. When a form with an unchecked checkbox is submitted, HTTP handles it in an unexpected manner. The value and the name of the checkbox input aren't passed to the processing template. As far as you can tell in the processing template, the checkbox input doesn't exist.

```
<INPUT TYPE="hidden" NAME="FreeYN"
VALUE=""><CFOUTPUT>
<INPUT TYPE="checkbox" NAME="FreeYN"
VALUE="X"
#Iif(FreeYN EQ "X",
DE("CHECKED"),DE(""))#> Free?
</CFOUTPUT>
```

By including a hidden field in our form, using the same names as the checkbox input (i.e., FreeYN), I'm assured that the variable, form.FreeYN, will exist in the processing template. If the user didn't check the box, then form.FreeYn will have the null value it gets from the hidden input object. If the user does the box, then form.FreeYN will have the value x (which is what I need to save in the associated database field). This approach means I don't have to include any logic in the processing template to check for form.FreeYN's existence.

A number of select objects displayed as dropdown lists are used in this form template to provide a list of valid entries for the associated fields ArtistID, ProducerID and CategoryID. I've used CFSELECT rather than a plain old HTML SELECT because it simplifies the job significantly. The value stored in the field is usually a code or an ID, but I want the user to be able to select from lists containing artist, producer and category names. This task is simplified by using the CFSELECT object's VALUE and DISPLAY attributes. These three dropdown lists need to be handled differently depending on whether the user is editing or adding.

If the user is editing, these fields (ArtistID, ProducerID and CategoryID) already contain values. While the entire list needs to be presented, the current value needs to be selected. This task is made considerably easier by the CFSELECT's SELECTED attribute. If the user is working on a new record, however, these fields will be null; therefore, there is no value to select by default. This is the reason for the chunk of logic for each CFSELECT – to determine whether the user is adding or editing.

```
<CFIF Function IS "add">
   <CFSELECT ·></CFSELECT>
<CFELSEIF Function IS "edit">
   <CFSELECT ·></CFSELECT>
</CFIF>
```

Handling Read-Only Access

You may have applications in which you are required to provide a read-only level of access to certain people. In my experience this is generally handled by assigning people to specific groups or by assigning different "rights" or privileges to a person. How do I accommodate this requirement in my framework? Do I provide a separate read-only template for certain people? You can probably guess my answer: of course not!

To accommodate users with read-only privileges, I add a little logic to the code chunk that determines the function, and to the field display. Suppose you have a session variable named ReadOnly that stores a Yes or No indicating whether or not the current user has read-only rights. The code chunk could be modified as follows:

Allaire

www.allaire.com

```
<CFIF Session.ReadOnly IS “Yes”>
    <CFSET Function="view">
<CFELSEIF IsDefined("Form.Add")>
   <CFSET Function="add">
<CFELSEIF IsDefined("URL.Edit")>
   <CFSET Function="edit">
```

Farther down in the form you'd include logic to display the field:

```
<CFIF Function EQ “view”>
   <CFOUTPUT>#Replace(Description,
chr(10),
    “<BR>”, “all”)#</CFOUTPUT>
<CFELSE>
   <TEXTAREA NAME="Description" COLS=50

ROWS=4><CFOUTPUT>#Description#</CFOUTPUT>
     </TEXTAREA>
</CFIF>
```

When you need to display the contents of TEXTAREAs in a display-only format, you can use this trick to force hard carriage returns that the user may have entered in the field: replace all of the line feed characters (ANSI character 10) with a BREAK tag (<BR>). This is done in the previous code snippet with the line:

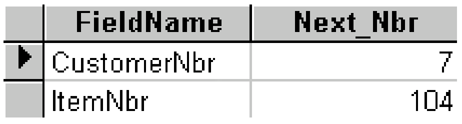
```
#Replace(Description, chr(10), “<BR>”,
“all”)#
```

There's another, simpler way to handle the folks who have read-only access: rather than having a separate set of logic for producing read-only text rather than HTML form objects, don't include a Save button in the form. Alternatively, you can include a button that returns the user to the calling template. The point is, don't execute the code that saves the data. I generally don't use this approach because if you give users the ability to change the data – or at least don't stop them – they'll expect to be able to save whatever changes they've made.

Dealing with Keys

One of the subtle complexities in developing Web-based database applications is the Back button. What happens, for example, when a user adds a new record to the database by submitting a form and then hits the Back button in his browser, goes back to the form and tries to submit the form again? Well, what shouldn't happen is the record should not be added again. If you don't do anything to prevent this situation, however, you can bet it will happen.

Here's a relatively simple approach that can be modified to deal with different circumstances. The general idea is that you create a unique number in advance, before the form is initially presented to the user. You include this number in a hidden form



	FieldName	Next_Nbr
▶	CustomerNbr	7
	ItemNbr	104

Figure 6: The NextNbr table is used to generate unique numbers for CustomerNbr and ItemNbr.

field. When the user submits the form to add a new record, you first check to see if the number has already been used; if it hasn't, you create a new record using the number. If the number has been used, you can display an error message telling the user not to resubmit records, or something to that effect.

Here's how I use the technique in my example. The Catalog table is keyed on the ItemNbr field, which contains a unique number in each record. I have a custom tag named NEXT_NBR that generates sequential numbers for specified fields. It works in conjunction with a table called NEXT_NBR. As you can see in Figure 6, NextNbr stores the next available number for use in two fields, CustomerNbr and ItemNbr. You can use other fields in your data entry applications.

When you call CF_NEXT_NBR, you indicate the datasource for the NextNbr table and the name of the field for which you want the next available number (the code for NEXT_NBR.CFM is included with the rest of the source code for this article). You can optionally specify the name of the variable you want created in your template to store the next available number. In my example I tell it to create a variable named ItemNbr.

```
<CF_NEXT_NBR DATASOURCE="#DataDB#" FIELD-
NAME="ItemNbr" VAR="ItemNbr">
```

I store the value of the next number in a hidden INPUT named ItemNbr. When the user submits the form and the Function is “add,” the post template checks to see if there's already a Catalog entry with this value. If there is, we know the user is trying to resubmit the same information and an error message is displayed (see Listing 4).

This functionality is needed only when you're adding new records to a table. But in the posting template how does it know whether I'm posting newly added records or modified existing records? Simple: in my data entry form I include a hidden INPUT that stores the value of the Function variable (“add” or “edit”).

The next number process can be made more efficient by using Application-level variables to store the next available number. The idea is that you'd get the next number when the application is initialized but save the next number to the NextNbr table only intermittently, perhaps every hundred numbers or so. This significantly cuts down on the I/O used by the approach described above. Well, I've started to get into issues relating to the posting template, so let's move there.

Post Template

The final template in my framework is used to save modified and newly created records into the database. It's invoked from the FORM ACTION in the form template (MaintainMIDIForm.cfm in my example). After the record has been dealt with, the user is redirected to the list template. Let's examine the post template (MaintainMIDIPost.cfm) in more detail.

It starts by determining whether it needs to add a new record or update an old one. There's a separate set of logic for each alternative. If I'm adding a new record, the query described in the previous section is executed. This query determines whether the new record has already been posted to the database.

```
<CFQUERY DATASOURCE="#DataDB#"
NAME="CheckForDuplicates">
      SELECT ItemNbr FROM Catalog WHERE
ItemNbr = #Form.ItemNbr#
</CFQUERY>
```

Next, I check to see if the query returned any rows. If it did, I display an error message and abort further processing. I'm using a custom tag called CF_SHOWERR. You can display your error message however you like.

```
<CFIF CheckForDuplicates.RecordCount GT
0>
   <CF_SHOWERR
      ErrorTitle="Problem Saving New
Record"
      ErrorMsg="The database already
contains a
       catalog listing with this num-
ber|Try
       starting over"
      GotoURL="MaintainMIDIList.cfm"
   >
   <CFABORT>
</CFIF>
```

Next, my posting template calls another custom tag, CF_REPLACEPIPESINFIELDS. This tag is extremely handy if you're working with MS Access databases and your input form includes INPUTs or TEXTAREAs that enable the user to enter free form text. For some reason the ODBC driver for Access won't let you insert the pipe character (“|”) directly into a table. This tag searches through the form variables you specify in its FieldNames attribute and replaces any pipes it finds with “|Chr$(124)|”. For some reason Access allows this. Go figure. (Mateo Ferrari turned me on to this trick and I took it and turned it into a custom tag. Thanks, Mateo!)

The post template then calls either an insert query or an update query to post the record. A trap to watch out for has to do with posting null values. MS Access text fields and memo fields have an "Allow Zero Length" property that takes a Yes/No value. You usually set this to Yes. If you leave it set to No and try to execute an insert query that contains a null value for one of your text or memo fields, the query will fail and the user will get an ugly error message. While it's safer to set this property value to Yes, that isn't necessarily enough.

If you're working with SQL server databases or if there's a chance that someday you may, write your queries so you never write null strings (i.e., "") into SQL server fields. Why not? Because they'll be inserted as the ANSI 32 character, i.e., a space (" "). To prevent this, always include code in your insert and update queries to check for empty strings and in their place use the SQL reserved word NULL. It's imperative that you don't insert this as text string. In other words, do not surround it with double or single quotes. Instead, use the word NULL as I've done in the example (see Listing 4).

The last task performed by the post template is to redirect the user back to the listing template. This is done with the last line of code.

```
<CFLOCATION
URL="MaintainMIDIList.cfm?Function=#URLEn
codedFormat(URL.Function)#">
```

Why am I passing the query string Function=#URLEncodedFormat(URL.Function)# back to the list template? Perhaps you remember that I did something similar in the deletion template. It's really very simple. The list template has some logic in it to provide the user with feedback on what just happened.

```
<CFIF IsDefined("URL.Function")>
   <P><FONT COLOR="Red">
   <CFIF URL.Function EQ "edit">
      Your record was successfully
saved.
   <CFELSEIF URL.Function EQ "add">
      Your record was successfully added
to the
      database.
   <CFELSEIF URL.Function EQ "delete">
      Your record was successfully
deleted from
      the database.
   </CFIF>
   </FONT>
</CFIF>
```

Note the message, in red, in Figure 2. Well, that pretty much wraps it up.

Conclusion

The real benefit of these techniques comes when you develop – and have to support – a number of data entry applications. If you adopt a framework and use it consistently from one app to the next, your work will become easier to maintain. If you follow my suggestions regarding naming your templates, they'll be easier to find. If you apply the same techniques within each type of template, your apps will be easier to maintain because you'll have a good idea of the techniques used in the template even if you've long since forgotten the original function of the app. CFDJ

CODE LISTING

The complete code listing for this article can be located at **www.ColdFusionJournal.com**

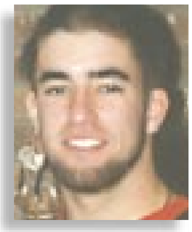
About the Author

Leon Chalnick is an Allaire Partner, a CF consultant and an independent author. He's been writing about ColdFusion since early 1996, when he wrote a column for the Informant Communication Group's "Down Cold." He's also written four of the advanced chapters in Ben Forta's CF books.

lchalnick@prprpr.com

Fusion FX Inc.

www.fusionfx.com



The BlackBox Technique

Simple...and free!

by **Dan Chick**

Recently, we've seen a dramatic increase in object-oriented programming languages. The ability to develop a solid piece of code and then reuse it has become the foundation for modern application development.

Web applications suffer from a lack of state and continuity. ColdFusion helps developers address both these issues. However, there's no provision for functions or subroutines when building a ColdFusion page. The use of custom tags helps address the issue, but creating functionality modules or libraries is something that has challenged ColdFusion developers since its inception.

Setting the Stage

Various techniques have been developed to help developers address some of these limitations, the most notable being Fusebox (www.fusebox.org). Fusebox calls for a strict set of coding techniques that enable developers to modularize their code into a specially formed directory. It has been an effort to establish some type of standard for ColdFusion application development.

To use someone else's Fusebox application, you simply included the directory under your application root and pointed your links to the index.cfm page. Despite the perks, Fusebox has its limitations and quirks.

For starters, all links in a Fusebox application point back to the initial calling document, which allows you to nest applications. This process, however, results in a very tricky effort to manipulate the "fuseaction" that must be passed with every request. This creates unattractive URLs and makes it nearly impossible to inform people about a subsection of a Web site. It also means you have to fake it to have nice URLs for complex sites. Finally, customizing the look and feel of these Fusebox applications, especially when running multiple instances of the same application within a single Web site, is difficult and limiting.

On February 23, 1999, I got together with a couple of other members of the local Twin Cities CFUG to discuss how we could create a technique that would behave the way we, as programmers, would like. At the same time we wanted the spec to be broad enough so you wouldn't have to be a hardcore programmer to use it.

Enter BlackBox...

The BlackBox style of developing ColdFusion applications is based on the premise that developers like to have a lot of control over their pages; they want to be able to access the brains of an application without worrying about the predefined beauty. It's designed to nest applications easily and create attractive URLs. It's designed so integration between multiple applications within the same site is possible and easy.

BlackBox is also based on the premise that developers will want to include the same functionality in several sections of a given Web site, but use it in very different ways. It employs this functionality to create the illusion of "functions." It's designed to let you code in the style of your choice, while letting you share your developments with others. It's a simple technique to learn and employ. Best of all, it's free.

The Basics

It doesn't take a lot to develop in BlackBox. The essence of a universally available function library is encapsulated in a file called blackbox.cfm. The most recent version of this file is available at the BlackBox Web site (www.black-box.org).

Instead of calling the pages in your library directly, you'll call them through this new CF_BLACKBOX tag. The tag automatically recurses the directory tree and finds the file or function you specified. All BlackBox library functions are contained in a subdirectory in your application's Web root (not necessarily the server Web root). This directory is called "blackbox," and each class of functions you add has its own directory within it.

If your Web root is c:\inetpub\wwwroot\, you'd have another directory called c:\inetpub\wwwroot\blackbox. You might have functions for "users" and "fileactions" that would be located in c:\inetpub\wwwroot\blackbox\users\ and c:\inetpub\wwwroot\blackbox\fileactions\.

To call an element of one of these libraries, simply use the blackbox.cfm custom tag:

```
<CF_BLACKBOX TOOL="users" ACTION="login-
form">
```

The TOOL attribute here refers to the name of the BlackBox directory that contains your files. The ACTION attribute refers to the name of a file in the directory that you want to call. By default, a .CFM extension will be appended to the ACTION. If a period is included in the ACTION, no extension will be appended. This allows you to differentiate between JavaScript files, CFML files and other "tools" you might have in your BlackBox toolbox.

Any additional attributes that you include in the tag are passed along to the called file. This enables you to mix and match functionalities from different function libraries within the same CFML page. The only compliance this technique requires is the BlackBox custom tag in every BlackBox-enabled directory.

Inheriting Application Settings

Since each displayed file can be called using the standard HTTP protocol, application.cfm is processed before every file. This is standard ColdFusion. Just like other custom tags, applications within the BlackBox directory won't have application.cfm called automatically, so each of your BlackBox templates might include a line like:

```
<CFINCLUDE TEMPLATE="bb_settings.cfm">
```

This bb_settings.cfm file is analogous to the app_globals.cfm file in Fusebox. The file name isn't part of the spec, so you have the freedom to build your BlackBox apps however you choose. Many of the BlackBox apps I build follow the same naming convention that Fusebox uses.

Within this settings file you can specify variables that your BlackBox files might need. You can also include a line like:

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

Since your templates will always be two directories down from your application root this is a safe technique. You can use it to pass DSN connection information to all of your BlackBox apps and preserve the same environment. One setting you might choose to make in your top-level application.cfm is:

```
<CFSET BlackBoxPath = "/blackbox/">
```

This represents the Web path to the root of your BlackBox directory. It will enable you to link directly to BlackBox files should the need arise. It also allows BlackBox files to make self-referential calls by linking to #BlackboxPath#appdirectory/thisfile.cfm. Unfortunately, since the BlackBox file is being called through a custom tag in a parent document, we can't use a relative link to achieve the same end.

Using CFINCLUDE Instead

If you like, you can CFINCLUDE the blackbox.cfm file instead of calling it as a custom tag. This technique has both positives and negatives.

On the upside, you'll be able to set variables in your BlackBox documents and have them available to the parent document. Alternatively, you can access variables and queries from the parent document directly instead of using "caller." variables. Files that are included also have a slight performance increase over files run as custom tags.

The downside is that you'll have to set ATTRIBUTES.TOOL and ATTRIBUTES.ACTION explicitly before you perform the CFINCLUDE. Also, since BlackBox files will have direct access to the variables in the calling document, there's a possibility that someone else's variable names will conflict with yours while using someone else's BlackBox code.

The best time to use CFINCLUDE instead of a custom tag is when you're building your own BlackBox directories and can control any possible naming conflicts.

How It Handles

In its typical form, a BlackBox tag is handled in the same way as custom tags. The action file you pass into BlackBox is then included using CFINCLUDE, so the same variable scope that applies to custom tags applies to BlackBox applications.

You can use BlackBox techniques to display and update information, or simply to set variables in your active document.

When Would I Use It?

If you're building a particular functionality that you know will apply only to a single situation, there isn't much to be gained by writing it in the BlackBox format. On the other hand if you're writing a user module, a guestbook application or some other functionality for use in several places in one or several different sites, then BlackBox is a great fit.

The first BlackBox module I built was a user management module. I built in functions to check if a user was logged in, to let users update their personal information and to test which users had access to what areas of the site. This last functionality ties into a separate BlackBox module I wrote to manage certain modular aspects of my site.

I built another functionality that presents a user login form. This tag accepted two parameters that allow a developer to specify URL destinations for both successful and unsuccessful logins. This allowed me to check if a user was logged in and to display the login form if he or she wasn't. Once the users logged in, no matter where they were in the site, I could put them on the very page they were trying to access without making them navigate the site to get there.

I was involved in the construction of a new portal site for boat-towed sports (www.wakecentral.com). I built the whole site using BlackBox techniques and created functionality modules for users, dynamic content, guestbooks, classified ads, banner rotations and more. When I started to build my portal for alternative Christian music (home.jesusfreak.com), the only thing I needed to do was create a new look and feel. The hard work had already been done and was completely reusable!

Since functionality rather than presentation is the key here, building intelligent Web sites with BlackBox and ColdFusion is like putting together the right Lego pieces and slapping on a fresh coat of paint.

Conclusion

I'm excited about using BlackBox. The power it affords in developing and sharing CF applications is tremendous. Applications can be developed in record time by developing reusable modules. CFDJ

About the Author

Dan Chick is the lead developer for FastCart Corp., a Minneapolis-based e-commerce firm. He recently headed the development of FusionCart, a ColdFusion-based shopping system. Dan runs the Twin Cities CFUG. You can contact him at chickd@fastcart.com.

chickd@fastcart.com

Macromed

www.macr

dia Spread

omedia.com

CFDJ FEATURE

The Problem of Complexity

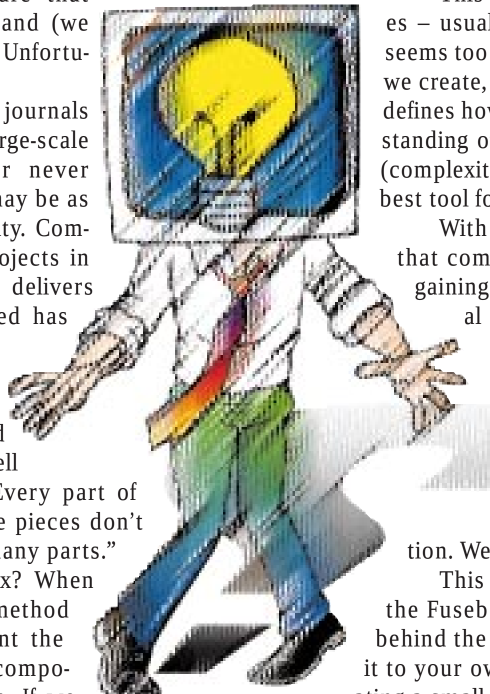
PART 1

Complex projects call for methodical measures

by **Hal Helms, Steve Nelson** & **Gabe Roffman**

As a reader of ***CFDJ***, you're probably an experienced programmer. But what exactly is it that you do? What is programming? Bruce Eckel, in his excellent book *Thinking in Java*, offers this answer: "At one level, all of programming is about managing complexity."

For many of us this answer resonates deeply. Arguments about which language is "best" miss the point – that programmers write programs, not benchmarks. And in the real world, the greatest challenge isn't to shave a few machine cycles off accomplishing a discrete task, but to build software that empowers users to do their work and (we hope) delights them in the process. Unfortunately, in this mission we have a foe.



Recent studies reported in trade journals indicate that the percentage of large-scale software projects that are either never deployed or designated as failures may be as high as 70%. The culprit? Complexity. Complexity is what drowns software projects in chronic time and cost overruns, or delivers software so late that the initial need has changed.

The more complex the project, the more failure-prone it is. As Arno Penzias, Nobel Prize winner and recently retired chief scientist of Bell Laboratories, recently remarked, "Every part of Windows 95 works perfectly, but the pieces don't work together because it's got too many parts."

When is something too complex? When does it have "too many parts"? One method of measuring complexity is to count the possible interactions between the component pieces that make up a system. If we have three coins – a penny, a nickel and a dime – the complexity of that system can be measured by how many separate combinations of coins we can pair.

With three components, our system has three possible interactions. If we begin adding coins, things get more interesting. Adding a quarter to the system gives us four components, but yields six possible interactions. If we add a half-dollar and a dollar, the number of possible interactions balloons to 15. By doubling the number of components, complexity has risen fivefold.

This simple example shows how easily complexity increases – usually without the system designers' awareness. If this seems too esoteric, consider this: since our tools influence what we create, the choice of those tools – and the methodology that defines how we use them – becomes highly important. An understanding of the probable failure point in software development (complexity) will guide us in selecting not the "best" tool, but the best tool for dealing with that problem.

With ColdFusion, Allaire has given us a tool for managing that complexity. What we lack is a methodology – a plan for gaining the most benefit from this tool. Over the past several months, a group of programmers has taken on this task. Spearheaded by two of the contributing authors to this article, Steve Nelson and Gabe Roffman, a programming methodology best suited to ColdFusion is emerging. With the help of almost two hundred developers, the methodology is becoming a specification. We call that specification *Fusebox*.

This article is the first of two meant to introduce you to the Fusebox specification. In this article we'll examine what's behind the specification and how you can benefit from applying it to your own work. Then we'll ask you to try your hand at creating a small Fusebox application.

Fusebox Methodology

As ColdFusion's success began attracting more sophisticated developers, they naturally brought with them the experience they gained working with other tools, languages and environments. These developers began working together to create a methodology that took the best of their experience.

Fusebox is a methodology for building ColdFusion applications. The purpose of Fusebox is to give ColdFusion developers a methodology that:

- Promotes reuse of code
- Allows small applications to scale to larger ones
- Supports division of labor among content providers, programmers, graphic artists and layout workers
- Makes code easier to read and understand
- Makes changing/maintaining code far easier
- Allows programmers to "embed" an entire ColdFusion application by turning the application into a ColdFusion tag

Object-Oriented ColdFusion?

You can hardly catch your breath without hearing about object orientation. Everything, it seems, has "gone OO" – even venerable COBOL. So it's natural to ask: Is Fusebox object orientation applied to ColdFusion? It's not. Objects are instantiations of classes – a notion foreign to Fusebox (and ColdFusion). Fusebox uses a different model than OO for building applications. Both approaches strive to reduce the complexity associated with software projects.

Look at an electrical fusebox (see Figure 1). You'll see the main electrical line that comes into the box and then splits out among different fuses. This is exactly what you do when writing a Fusebox application. A single application will be split among various "fuses." Each fuse is quite modular in design, which makes it small and allows it to be reused. As we'll see, this modularity is a great boon to development teams, promoting a division of labor among the divergent skills needed to create successful Web applications. Finally, this modularity makes maintaining code much easier than before.

Hub-and-Spoke Architecture

If you were to draw out the architecture of Fusebox, it would resemble a hub-and-spoke system (see Figure 2).

The fusebox is the index.cfm file. Its structure is remarkably simple, making understanding the overall application design quite easy. The job of the fusebox is to "manage the action." The actions, as we'll see, are the sum total of what the application needs to do.

Like a good manager, the Fusebox delegates (fuse)actions to others. These others we call *fuses.* Fuses may be single, consisting of only one file, or compound, where multiple files are joined to make up the fuse. In either case, the fuse-files are <cfinclude>d into the fusebox. This allows them to inherit all the variables of the fusebox itself. Whether single or compound, fuses handle a fuseaction and then return the action to the Fusebox. This constant "returning to the Fusebox" eliminates the problem of the "spaghettification" of code.

The Secret Life of Fuses

Just like its electrical counterpart, Fusebox contains different kinds of fuses. They can be as simple as a single line, or they may contain complex forms or sophisticated algorithms. We find it helpful to prefix fuses so as to convey information about what sort of action the fuse handles.

Fuses that involve the display of information on the screen and gathering user input are given a "dsp_" prefix. Other fuses are used for heavier lifting, such as working with databases. These fuses are called *action fuses* and have an "act_" prefix. If you find yourself using a query repeatedly in your application, you might want to make a fuse out of it and give it a "qry_" prefix. We find that by using these prefixes we can scan a fusebox (index.cfm page) quickly and have a good sense of what the application is about within minutes.

We mentioned earlier that Fusebox is not OO programming applied to ColdFusion. (OOP requires that a language support polymorphism, inheritance and encapsulation – the so-called OO "pie.") However, since both methodologies have the idea of reducing complexity as a key part of the process, it doesn't surprise us to find that creating fuses is quite similar to building objects in OO languages. Like objects, fuses work best when they have limited functionality and remain small in size.

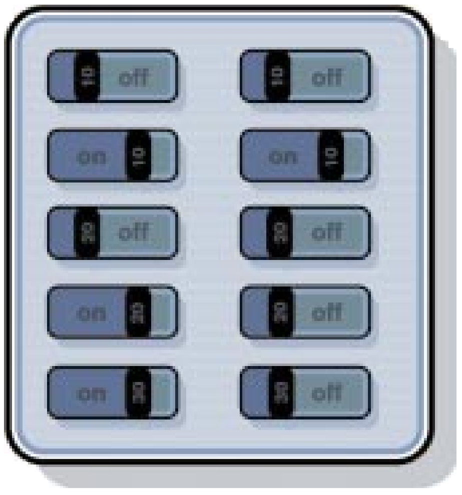


Figure 1: Electrical fusebox

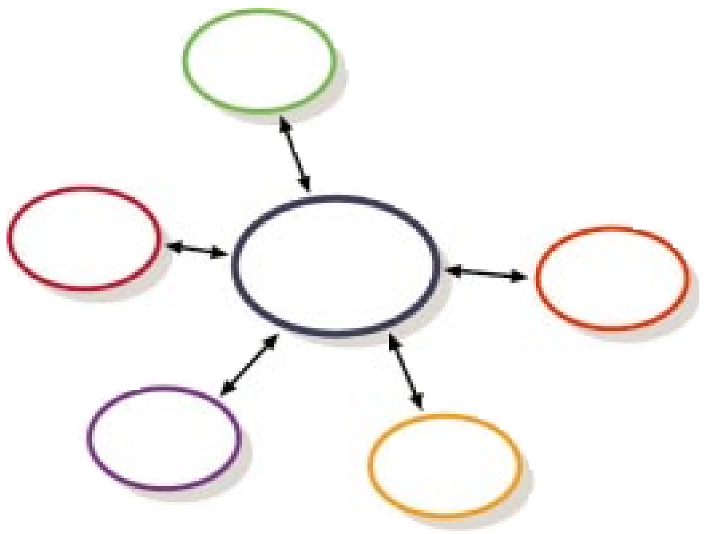


Figure 2: Fusebox architecture: Information flows are well defined.

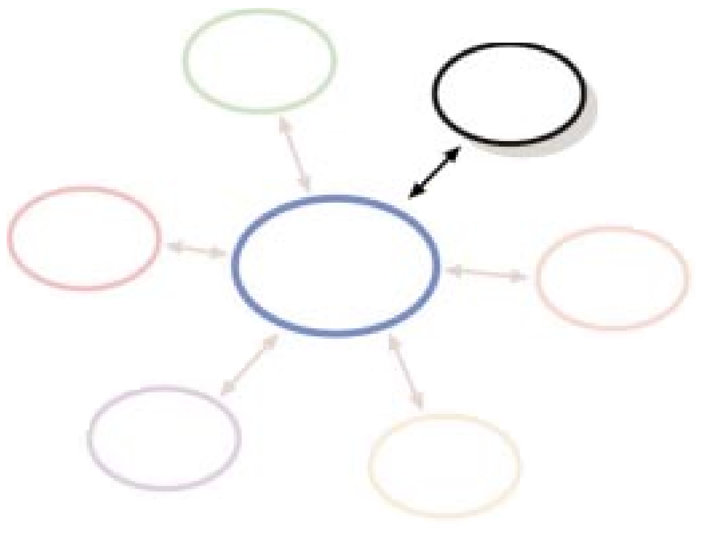


Figure 3: Needed changes involve minimal disruption to the system.

By breaking a complex program into smaller "chunks," we can now better use the skills of any team members working on the project. Graphic artists and layout workers can create display pages while programmers simultaneously create the action pages.

As we'll see, the interface is attached to the actual workings of the system by a few well-defined methods. This provides a loose coupling between interface and code that allows us to build prototypes rapidly. We find that nothing provides as much valuable feedback as getting the actual application in front of users early and often. Fusebox's modularity encourages us not to delay this process, allowing us to place stubs in place of code and graphics when needed.

Every application has one – and only one – fusebox, but may have many fuses. Since fuses can be built to do virtually anything, their structure is less defined than the fusebox. Still, fuses have some elements in common. Here our simplistic fuse, dsp_ShowAll.cfm, handles the "ShowAll" fuseaction by querying a database for members and displaying them as a hyperlinked list.

As was said earlier, a single fuse is designed to have limited functionality. This tends to make fuses more generic and helps developers write code for reuse. Since this code is <cfinclude>d into the fusebox code (index.cfm), you can concern yourselves with only the functionality this fuse needs to provide.

Use of Attributes Variables

Keen-eyed readers may have noticed a call to a custom tag in the fusebox itself. This tag, "FormURL2Attributes," converts form and URL-style variables to attributes variables. It's important to understand why this is done, as the use of attributes variables is essential in allowing Fusebox to encapsulate an entire application so that it can be called as if it were a custom tag, using either the <cf_tagname> or <cfmodule> syntax.

A custom tag takes variables passed to it and turns them into an "attributes"-style variable. Within the custom tag, you must refer to any variables with the "attributes." prefix. If an entire Fusebox application is to act like a custom tag, it too must subscribe to this naming scheme.

If you make this call (see below), ColdFusion designates the variables "authors" and "subject" with the "attributes." prefix.

```
<cf_MyCustomTag
authors = "Hal Helms, Steve Nelson, and
Gabe Roffman"
subject = "Fusebox Methodology">
```

If you want to refer to the variables from within a custom tag, you must use this "attributes." notation. If you don't, ColdFusion will stall, failing to understand what you're referring to. The code within your custom tag must look like this:

```
<cfoutput>
#attributes.authors# are the experts on
#attributes.subject#
</cfoutput>
```

We said that we want to be able to use entire applications as custom tags, but they must also function as standalone applications, capable of receiving and handling variables of all stripes: URLs, forms, cookies, etc. We must find some way of accommodating all kinds of variables within a single code file.

This is just what the custom tag "FormURL2Attributes" does. It transforms form and URL-style variables into attributes-style variables. When the fuse is operating in the context of a standalone application, FormURL2Attributes ensures that it needs to deal with attributes-type variables. Now our code can refer to all variables passed to it as "attributes." variables.

You may wonder why we need to call entire applications as custom tags. We said at the outset that one of the benefits of the Fusebox methodology was to allow developers to piggyback onto the functionality of other applications. If, for example, we already had an application called "Members" that took care of adding, editing and deleting members, we wouldn't want to duplicate these actions in our application. Ideally, we'd like to ask this "secondary" application to handle it for us.

We can do this by treating the Members application as a custom tag. The fusebox tells us that it accepts a fuseaction called "addMember" by calling the "dsp_AddMember" fuse. The fuse "dsp_AddMember.cfm" expects to receive both a "userName" and a "password" variable when it's called. We can do that with this call:

```
<cfmodule
template="../Members/index.cfm"
fuseaction="addMember"
userName="hsimpson"
password="doh">
```

This will work identically to running the Members application and selecting the "New Member" button.

If you're still unsure about this, relax. Most users struggle with it initially. The good news is that just a little practice writing to the Fusebox specifications will make this clear.

Back to the Fusebox

In the example "dsp_ShowAll.cfm" fuse, we let the user drill down on an individual member to see more information. But

Allaire

www.allaire.com

notice the target of the hyperlink: it's the file "index.cfm" – the fusebox itself. Since the fusebox never relinquished control, the call to index.cfm is recursive; the page is calling itself.

This notion is key to Fusebox. We said that the architecture is like a hub and spoke. All action returns to the hub – that is, the fusebox. This is always accomplished by calling the fusebox and providing it with a new fuseaction and any other parameters it needs. Adding functionality to existing Fusebox applications is as simple as adding new fuseactions and the fuses to handle them (see Figure 3).

Now the circle is complete – the fusebox has passed the action off to a member fuse to take some action. Having completed its task, the fuse returns the action to the fusebox and a new cycle begins.

A Break from the Action

If you've followed along this far, stop and congratulate yourself. Learning anything new is difficult: you're asked to enter a place with few, if any, recognizable landmarks, and it often seems impossible that you'll ever grasp the layout of the new land.

Yet for those who persevere, the landscape gradually becomes less foreign and strange concepts become familiar territory. The secret to acquiring this knowledge is similar to learning any new territory: exploration. At the end of this article I'm going to ask you to strike out on your own by trying to create a simple Fusebox application.

For now, let's cover the remaining "rules of the road" you'll need to safely navigate your way through this new world of Fusebox.

No more application.cfm files: Application.cfm files are executed prior to any .cfm file. This makes them useful for holding "global" variables and any code common to all application files. However, application.cfm files present a problem for the Fusebox developer. We saw above how one application could "borrow" functionality from another by treating the entire application as a custom tag. But in the case of custom tags the application.cfm file isn't called prior to code execution. This means that the application won't have access to code that it typically depends on to function correctly. If we want to make sure that our application is Fusebox compliant – and can therefore be called as a custom tag – we must approach the matter differently.

Fusebox applications will have an app_Globals.cfm file that's <cfinclude>d as part of the fusebox's initialization and setup. This gives us a place to put variables that will need to be used by many fuses. This includes common code and the <cfapplication> tag that lets us enable client and session management for an application. The app_Globals file provides for the functionality of application.cfm files while allowing the app to be processed as a custom tag.

Attributes.Fuseaction is required at all times: This means that all URL links, forms and <cfmodule> tags will pass an attribute called fuseaction. Notice that we set a default value for this in our fusebox file so that any call to the fusebox without a fuseaction will not break our code. This rule is simply a repeat of what I already said. If you find your code doesn't do what you want, check to make sure that you have a current fuseaction – and that it's the one you expect.

Here are different ways of calling the fusebox with a "deleteMember" Fuseaction and a parameter of "memberID".

- For a URL link:

```
<a href="index.cfm?fuseaction=deleteMem-
ber&memberID=7">Delete Member</a>
```

- For a form, either of these methods:

Form Method 1

```
<form action="index.cfm" method="post">
<input type="hidden" name="fuseaction"
value="deleteMember">
<input type="hidden" name="memberID"
value=7>
</form>
```

Form Method 2

```
<form action="index?fuseaction=deleteMem-
ber&memberID=7" method="post">
```

- For a file that returns to the fusebox without any user interaction:

```
<cflocation
url="index.cfm?fuseaction=deleteMember&me
mberID=7">
```

- For use with a <cfmodule> tag:

```
<cfmodule
template="../yourApplication/section1OfY-
ourApplication/index.cfm"
fuseaction="deleteMember"
memberID=7>
```

An application is always called through the index.cfm file: This is simply another way of saying that the fusebox itself defines the application. If your Web server doesn't serve index.cfm as the default page, change the server properties or rename the fusebox to support your Web server.

More on Learning

So far we've covered the bare bones of creating a Fusebox application. For a few, this will be sufficient. Most of us, though, need more than rules; we need to model behavior. We begin by imitating others who have mastered some art or skill – often without understanding the reasons behind their actions. Gradually, some of what we have imitated rubs off. We can now take what we've learned and arrange it in new ways to solve new problems.

Anyone who's been fortunate enough to watch children learn to speak must marvel at how inborn and natural this ability is. It's a miracle that occurs so frequently we take it for granted. That is, until it's our turn to learn something new – like Fusebox.

In the Beginning...

Give a young programmer a task and watch how earnestly and with what energy he or she begins producing code. This is appropriate for novices: their main attention is on the mastery of coding basics. Only after this challenge is mastered can we reflect on *what* we're building instead of *how* we're building it. Give the same task to a battle-scarred veteran and the activity is quite different, as the programmer first takes on the task of designing the application to be coded.

But how to approach this task of designing? In this section I'll introduce you to a methodology for determining application requirements and for building a functional design. If you've had much exposure to OO design techniques, you'll notice we borrow some of the excellent methods this technology has produced. Our goal here is to do more than present a new set of rules. Instead, think of this as a guided introduction to Fusebox. Use it to spur your own creativity and apply your own judgment to learning these powerful techniques.

To help you in this, I'll introduce your homework assignment – creating a simple Fusebox application. I'll also introduce your client, Tom. He's asked for your help in creating an application to help manage the "action items" that he and his staff are responsible for.

The first step in our methodology is to talk to the client and find out what he needs.

You: Tom, what problem do you want to solve with this application?

Tom: Well, I spend most of my day in meetings, and at these meetings we make decisions about what we need to do next. These "action items" get assigned to various people to handle. The problem we have now is that they're written down on paper and handed to people. By the next meeting, nobody can remember exactly what action items were assigned, who they were assigned to and what their status is. We've got to find a better way to handle this. That's what I'm hoping you can help us with.

Eprise

www.eprise.com

You: Okay. Imagine that I've just delivered your new "Action Item Manager" application to you. Tell me, what sort of things does it handle?
Tom: Handle?
You: Right. For instance, does it let you create a new action item? Can you get rid of an old one? That sort of thing.
Tom: Well, yes, I need to be able to create a new action item and assign it to myself or to one of my staff. Let's see....I want to assign a priority to it, and have a due date for it.
You: Will you be the only one creating action items – or can others do this too?
Tom: Oh, definitely – let others do it too.
You: Okay, so we'll want to know who created the action item?
Tom: Yes. Good point.
You: Okay, what else?
Tom: Well, it needs to tell me what action items I have, and let me sort them by date, priority or whatever. Then I'd want to be able to mark it completed, but keep a record of it. I'd also like to let others update the item's status so we can see where they were in finishing it, what problems they were having, what resources they needed – that sort of thing.
You: And you want to keep this private so only authorized people can view the action items, right?
Tom: Yes. And I'd like to be able to ask for the status of a particular item – maybe have it e-mail the person with a request. Also, if we assigned it to the wrong person, he or she should be able to alert the person doing the assigning, or when an item is finished, be able to let the person who assigned it know about it.
You: We can do that. Anything else?
Tom: I think that's it. How long will it take you to finish this? I'd like to start using it!
You: Well, we do things a little differently now. Instead of coming back to you in a few months with a finished application, I'm going to involve you and your staff early and often in the prototype phase.
Tom: Really?
You: Absolutely. We find it's really the only way to make sure that your needs get translated into software.
Tom: Great. What's the next step?
You: I'm going to do some design work and produce something we call "use case scenario tests." This will help determine if we've identified all your requirements. I'd like to come back to you in a few days to go over them. And it would be great if you invited anyone who will be using the system to join us.
Tom: Okay. Give me a call when you're ready.

Making Fusebox Cards

As a system designer, your main job is to take the requirements of users and translate them into an actual Fusebox design. Customers tend to think in terms of complex actions that involve several smaller, distinct actions. You need to decompose these complex actions into simple actions, which will then determine what fuseactions – and from there, what code files – your application will need.

For example, Tom has identified a need to decline an action item assigned to a user. As system designer, you understand that this is a complex action requiring three simple actions:

1. Let the user specify which item to refuse.
2. Change the database record to reflect the refusal.
3. Notify the original sender of the item that it's being declined.

There are no rules to determine how far to break down actions. Your experience and preferences as a developer will guide you in this. Will your fuse call a single "do-it-all" file, or will you decide on using several smaller, more generic files that together will define the fuse? The choice is yours. If you want to write code that can be reused, you will probably "slice" actions finely, allowing them to be made generic.

The next step in the design process is to make "Fusebox Cards." These are standard 3" x 5" cards that you'll use in developing a workable, scalable and maintainable design for your system. Each card contains the following information:

1. *Name of a single fuseaction* – you'll have a separate card for each fuseaction.
2. *File(s) that make up the fuse* – those (hopefully) reusable building blocks of code.
3. *Responsibilities of the fuse* – personify the fuse and have it describe what it does. For example, an "addItem" fuse might show the user a list of existing items and then present the user with a form to enter a new item.
4. *Attributes required* – what information does this fuse need to do its job? One advantage of this is that when you're done with your design, you can factor out common attributes and make them global in scope.
5. *Return fuseaction* – if present, this indicates that the fuse will by default call a specified fuseaction. For example, the "action" portion of a form would be the fuse's return fuseaction.
6. *Return attributes* – these are the variables that this fuse will return to the fusebox.

You want these as separate cards so you have maximum flexibility in determining a good design. Also, you'll use these cards in the next phase of Fusebox design. This idea may seem strange at first, but go through the process and see if it doesn't help you make better design decisions.

Use Case Scenarios

By now, our novice programmer has been hard at work. He has an impressive amount of code produced, and perhaps the growing realization that he's forgotten one or two minor things. As he walks by our office, he pops his head in. "Are you done playing with those little cards yet?" Not quite yet.

We told Tom we'd be back to run some "use case scenarios" with him and any other interested users. Use case scenarios are part of a process to ensure that our design is sufficient to handle likely real-world scenarios. With your Fusebox cards in hand, you're ready for your next meeting with Tom and his staff.

You: Thanks for coming, everybody. Tom has asked me to create an application that will help you all track the action items that come up in meetings. Tom and I went over some of the requirements of the application and I've come back – with these.
Tom: Little cards, a piece of paper and a Nerf ball?
You: Exactly. The little cards are what we call *fuses*. This paper represents the "fusebox" and the Nerf ball will indicate which fuse has the action. I'm going to ask each of you to play a part in making sure that we have the application planned to handle things that come up in the real world. Each of you gets a set of cards. Notice that each card has a name on the top?
Shelley: Yes, I've got "qry_showItems", "act_deleteItem"…
Wallace: I've got "dsp_declineItem".
Maria: I've got "act_declineItem".
You: Okay. You're all going to play the part of these fuses. Notice the next line on the card?
Bob: Responsibilities.
Susan: This is written as if the little fuse is alive.
You: It sure is. Personifying fuses really helps to make sure that we have the information we need for each fuse – and that we know what to do with it. I'm going to call out something that a user might want to do with the system – something like "the user wants to add an item." Tom's

list – what we call the fusebox – has three columns. One's called "Use Cases" – that's what we call those "user wants to do" things – and another is called "Fuseaction." Fuseactions are specific tasks the system needs to respond to.

Wallace: Like decline an item?

You: Right. The user thinks, "Tom sent me an action item to redo the marketing slicks, but I can't do that. It's due in two weeks and I've got training in Boston all next week. Somebody else has to do this." But that's too hard for a machine to understand, so we'll simplify it into the fuseaction "declineItem".

Susan: So "declineItem" is in the second column?

You: Right – because it's a fuseaction. The third column tells Tom which individual fuses – that's you, folks – are set to respond to the fuseaction. Sometimes he'll only need one of you and sometimes he'll get several of you together to each do a part.

Bob: Why not just tell us what fuseactions we're responsible for?

You: Good question. The truth is, Bob, as fuses, you have a very limited social life. You wait to get called, you make sure you have the information you need and you do your thing.

Bob: Sounds like my life at work.

You: Well, that's what the fusebox is for. It coordinates all your efforts to make sure that what the user wants gets done. When I call out a use case, Tom will translate that into a fuseaction. Then he'll read across to see which fuses need to be involved. Then Tom throws the Nerf ball to the fuse.

Maria: You said some fuseactions take several fuses?

You: Right. Tom will throw the Nerf ball to the first fuse on his list. You do your thing, and if you're connected to another fuse you throw the ball to them. If you've got a return fuseaction listed, you throw the ball back to Tom and call out that fuseaction.

Susan: The "Return Attributes" are what?

You: This is the information you're going to give Tom. Things like the name of the table to update, or the ID of the selected user.

Maria: But how do we get these use case scenarios that start everything off?

You: We make them up! I tried coming up with some on my own – but the ones you'll come up with will be a lot better because you know how you need the application to work. We start each use case scenario with "User wants to…" and then you fill in the blank to reflect all the possible things a user could want to do. We may decide that we won't have the application handle it – but at least it'll be a conscious decision based on your precise needs.

Tom: That makes sense.

You: Let's get started then.

This kind of simulation is helpful in creating a working design. Problems in design exposed here – prior to coding – are easy and inexpensive to fix. Plus, having users involved in these scenarios can be a wonderful way to test a design for completeness and correctness.

One of the surprising things we find using a use case scenario is that what we often perceive to be valuable to the user is seen as completely different by the user. And inevitably, things we had no idea about are brought up. What users want – and what they don't want – is entirely unpredictable. Often this will be the first time that users have even been consulted on what's important to them. This is the one effort that will give you, the developer, the greatest payback. You get highly targeted feedback from the ultimate judges of your program and buy-in from all.

Starting the Session

Brainstorming

Start the session with a group brainstorming session. At this point you want ideas, so take any that are given. Once we're in edit mode, the group will sort out the unimportant ones. It helps give structure to the exercise to start each scenario with "The user wants to…." Somehow filling in a sentence is easier than creating one.

Brainstorming is something akin to popping popcorn. It begins slowly, followed by a period of intense activity, and then tapers off to very little action. Just like popcorn, you stop when the frequency of activity dramatically falls off. It has its own natural flow that you'll sense.

Role-Playing

Next comes the really interesting part – role-playing. Ideally, each person takes the part of a single fuse. Depending on the complexity of your application and the number of use case testers you have, users may have to play several roles. Someone else needs to act as scribe, writing down the comments of the fusebox and fuses alike.

One user – Tom in our example – takes the job of the fusebox. Start with the first scenario and toss it to Tom. Using a small Nerf ball to define the fuseaction may break the ice. You can watch the fuseaction as it moves through the application. This is the first test of your system. Is there a default action the fusebox will initiate?

Once the person acting as fusebox determines the action to initiate, he calls the appropriate "fuse" to step forward. The fusebox throws the ball to the fuse. The fuse is responsible for (1) determining whether she has the information she needs and (2) executing her behavior. Once done, the fuse throws the ball back to the fusebox and calls out the next fuseaction. This is often a failure point: Does the fuse know what fuseaction to request from the fusebox? If the fuse handles multiple fuseactions, do they all require the same follow-up fuseaction? If not, how will your fuse know which fuseaction to request?

We find it helpful to keep the application structure very simple for the use case scenarios. Once coding begins, you may extend the hub-and-spoke analogy, letting fuses call custom tags to help carry out their assigned tasks. If necessary, a custom tag could even call another custom tag.

There's no set limit on the number of helpers each fuse may employ. Remember, however, that ultimately the action must return to the fusebox along with a fuseaction request. That said, you might well be suspicious if you have custom tags calling other custom tags: it may be an indication that your code is spaghettifying. Remember that one of the primary reasons for using the Fusebox method is to reduce complexity. Remember, too, that the more entities involved in an action, the greater the complexity. Keep it simple.

Now for the Homework...

Tom is still waiting for his Fusebox application! If you're feeling up to it, why not try building a small Action Items application that will handle the kind of use cases we've outlined above? Don't be too hard on yourself if you don't get it at first. In my next article I'll take a look at how Fusebox was used recently to build a very complex electronic commerce application, and I'll talk with the programmers and designers to find out their reactions to using Fusebox. See you then. CFDJ

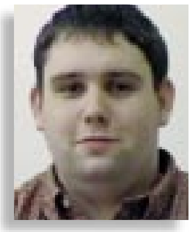
About the Authors

Hal Helms lives in Atlanta, Georgia, where, as chief technologist for User Technology Associates, he heads a team of developers working to simplify Web development. He can be reached at hal.helms@utaweb.com.

Steve Nelson is the lead ColdFusion developer at Monticello Systems, Inc. He hosts the Charlottesville, Virginia, ColdFusion Users Group and can be reached at m@secretagents.com.

Gabe Roffman lives in Charlottesville, Virginia, where he acts as lead developer for Netric Systems. He can be reached at gabe@netric.com.

hal.helms@utaweb.com m@secretagents.com gabe@netric.com



An Extension to ColdFusion

Deal with complexity without CFML

by **Paul Underwood** & **Simon Clarke**

We started out in the industry when Visual Basic had just been unleashed on the developer community. It contained these add-ons called VBXs that allowed programmers to add just about any type of grid, graph, spell checker or button maker to the language. Over time, VBX became the standard for other Microsoft products (as well as for most third-party companies). It took development tools to a new world.

Though VBX controls looked fairly simple at the time, Microsoft was quick to jump on the idea of allowing users to add extra features to its products. Today we have COM objects or ActiveX controls, which at the end of the day are the basics of most Microsoft products.

To prove we're not just on Microsoft's side....In the last couple of years, a small games company (ID Software) produced *Doom* and *Quake*. These games allowed players to create their own levels, weapons and so on, and they are two of the most played games in the world. And they have thousands of commercial and free add-ons.

ColdFusion offers its own extensibility in the form of ColdFusion Extension (CFX) tags, but having seen the changes from VB to Java tools and ASP, nothing really grabbed our attention until ColdFusion.

One of the main reasons any developer tool, or game like *Quake*, succeeds lies in its ability to be extended. ColdFusion has two ways to create custom tags. One uses the CFML language, but the real power is in using the CFX API in ColdFusion directly.

Why use a CFX tag?
To speed up repetitive processing, for one thing

At Internet Vision we've created a number of CFX tags, and we'll soon be releasing some commercial applications with a CFX interface.

Technical Overview

In many respects, CFX tags are similar to custom tags written in CFML. A CFX tag, however, takes the form of an external Dynamic Link Library (DLL) written in Visual C++, Delphi or any other language that creates DLLs.

So why would you want to write a CFX tag? The main reasons are to speed up repetitive processing that would be slow in CFML, and to use features available in C++ or Delphi that aren't available in CFML.

Creating a Tag in VC++

A CFX tag wizard is provided to construct the framework for the DLL and to register the tag with the ColdFusion server. Allaire provides a number of support classes known collectively as the ColdFusion Extension API (CFXAPI). The tag revolves around a class called CCFXRequest. When the ColdFusion server comes across a CFX tag in a page, it executes the ProcessTagRequest function in the DLL and passes a CCFXRequest object as a parameter. This object contains all the information contained in the tag. The information can then be used to create an output HTML and write it back to the CCFXRequest object. Simple.

Take a look at the following example, which takes two tag attributes, A and B, adds them together and then writes back, "The Answer is A+B" (see Listing 1). To use an attribute, you first have to make sure the attribute exists. You can do this with the CCFXRequest::AttributeExists method. You can then use the CCFXRequest::GetAttribute method to get your attributes, generate the HTML code and use the CCFXRequest::Write method to send the HTML back to the server. If for any reason you can't complete processing the tag (in this case, if either attribute A or B is missing), you can raise an exception and provide information on what went wrong. This will cause processing to terminate and an error to be displayed in the client browser.

For brevity, the exception-handling code generated by the wizard has been omitted.

Use of this tag might look something like this :

```
<CFX_OUR_TAG A=4 B=6>
```

and it would output:

```
The Answer is 10
```

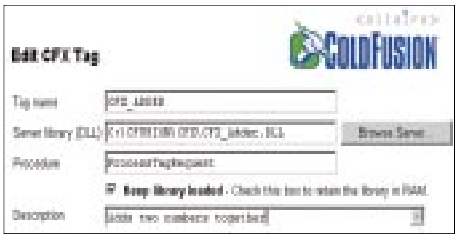
In addition, Allaire has provided two other classes that support the CFX tag. We'll look at each one in turn and examine what they're used for.

- ***CCFXQuery Class***: The CFX tag can contain a query as an attribute. The CCFXRequest::GetQuery method can be used to retrieve this query, which is of type CCFXQuery. The CCFXQuery class gives basic access to a query. Using CCFXQuery, you can retrieve a list of field names, add rows, and retrieve and set individual data items.
- ***CCFXStringSet Class***: This is the basic class for storing and accessing lists of strings. For example, if you take a query and want to know if a field exists, you can use the CCFXQuery::GetColumns method to populate a CCFXStringSet with the column names. You can then employ the CCFXStringSet::GetIndexForString method to find out if an item exists and, if so, where. There is, in fact, a fourth class called CCFXException class. This is used internally, and has no methods of its own.

It's not necessary to use MFC in CFX tags, but it can be useful.

Drawbacks

This is all well and good, but what are the drawbacks of using CFX tags over CFML? The most common problem is in debugging the tag. While support is given by the ColdFusion server to debug a tag, it's still very easy to get the server and/or windows to fall over.


Edit CFX Tag

Tag name: CFX_ADDER

Server library (DLL): C:\CFUSION\CFX\CFX_Adder.DLL

Procedure: ProcessTagRequest

☑ **Keep library loaded** - Check this box to retain the library in RAM.

Description: Adds two numbers together

Figure 1: Registering the tag with the ColdFusion Administrator

Another drawback (in some ways related to this) is that any bugs in the CFX tag have the potential to cause serious problems on the server when in general use, unlike a CFML tag, which may simply cause an error to be generated.

There are also some annoying shortcomings in the CFXAPI. For example, there's no direct way to find out where this tag is executed. Let's say that you want to write a JPEG file back to the page's local directory so it can be included in the page. You'd have to force the user to pass this directory as an attribute. There's also the limitation of being able to pass only one query to a CFX tag, which is particularly annoying.

Conclusion

While CFX tags have their drawbacks, once you're used to the way they function, they're easy to write and use. This said, CFML is preferable if you don't need the added speed of C++ and if what you want to do is possible in CFML.

Once you've finished your tag, you'll want to install it on your Web server, which is extremely easy. All that's required is to copy the DLL into the cfusion/customtags directory, plus any other supporting files to the server, and then register the tag with the ColdFusion Administrator, as shown in Figure 1. CFDJ

About the Authors

Paul Underwood, a business and development manager, is a cofounder of the UK ColdFusion User Group, and has been working in the developer tools industry for about seven years. Paul works for Internet Vision in the UK and can be reached at paul@ivision.co.uk.

Simon Clarke, a CFX consultant, is Internet Vision's in-house CFX developer. He designs CFX controls for in-house projects as well as for the firm's clients. Simon can be reached at simon@ivision.co.uk.

paul@ivision.co.uk simon@ivision.co.uk

LISTING1

```
void ProcessTagRequest( CCFXRequest* pRequest )
{
 …
  CString Answer;
  If (pRequest.AttributeExists("A") &&
   pRequest.AttributeExists("B")) {
   Answer.Format("The Answer Is <B>%i</B>",
    atoi(pRequest->GetAttribute("A")) +
    atoi(pRequest->GetAttribute("B")));
   pRequest->Write( Answer ) ;
  }
  Else {
   pRequest.ThrowException( "Attributes Missing",
    "Check that attribute A and B exist");
  }
 …
}
```

CODE LISTING

The complete code listing for this article can be located at **www.ColdFusionJournal.com**

Sitehosting.net

www.sitehosdting.net

Creating Subforms with CFGRID

Lose the restrictions of HTML for your layout and design needs

by **Mark Cyzyk**

You're familiar with the rudiments of ColdFusion development. You know how to set and read local, session and client variables. You have input, updated and deleted data from a database. You've created drill-down "Master/Detail" templates by passing values in a URL string and you've created dynamic, data-driven SELECT boxes. Your forms look spiffy and everything is working fine, just like it did with Microsoft Access…well, almost.

Your Access application included an input form that contained a "subform" and allowed you to update two tables simultaneously. They were linked by a one-to-many relationship so for every record in your main table there could be a variable number of linked records in the secondary table. But you can't figure out how to accomplish this same layout and functionality within the restrictions of HTML. Luckily, the ColdFusion Markup Language contains a tag, CFGRID, that will enable you to do this.

The Scenario

Suppose, for instance, you want to create a Web-based "Experts List," i.e., a database of experts and lists of their respective topics of expertise. Each individual expert can have a variable number of these, so the database should therefore be designed with two tables – one containing the main record for the expert, the other containing a list of topics of expertise. These two tables should be linked in a one-to-many relationship, using the Expert ID as the key. Figure 1 illustrates the entity-relationship model of this simple database.

Input Form

The ColdFusion input form for this simple application will start out with some straightforward HTML tags. It will include TEXT boxes in which to input the expert's first and last names:

```
<form method="post"
action="inputaction.cfm">
Firstname: <input type="text"
name="firstname" size=25><br>
Lastname: <input type="text" name="last-
name" size=25>
<p>
<input type="submit" value="Input It!">
</form>
```

Now it's time to program the subform for the variable number of topics, which is where CFGRID comes in.

HTML wasn't designed with databases in mind. Hence, there's no tag allowing for a subform structure. CFML was designed with data access in mind, and includes a Java applet – CFGRID – that adds a spreadsheetlike dynamic grid to your forms. This grid structure can be used as a subform within the confines of the regular forms we're all familiar with.

The first thing to do when adding a CFGRID to the form is to change the <form></form> tags. Because CFGRID provides complementary functionality to the regular HTML form tags, it must be surrounded by <cfform></cfform> tags. So the new form tags will look like:

```
<cfform action="inputaction.cfm">
.
</cfform>
```

Note that, by default, CFFORM uses the "post" method, so you don't explicitly need to state that within your tag.

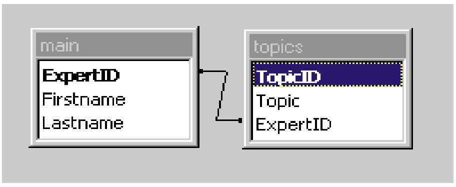


Figure 1: Entity-relationship model

The next tag is CFGRID, which allows a Web client to input a variable number of topics. The final input form, including the CFGRID code, can be seen in Listing 1.

The grid has an opening as well as a closing CFGRID tag. The opening tag sets attributes such as the grid name, width and height. It also sets a SELECTMODE – in this case "Edit" – that determines whether or not the Web client can interact with the data in the grid or just browse. The final attributes of the grid enable the Insert and Update buttons. It's via the Insert button that the Web client will be able to insert multiple records.

Within the CFGRID tags is a CFCOLUMN tag, which determines how many columns the grid will have as well as the format of the data in those columns. It also provides a name for the column – in this case "topic" – which will be used to insert data into the database via the final action template. Figure 2 illustrates how the form now looks.

To input data, fill in the Firstname and Lastname fields, then click on the Insert Topic button within the grid to insert each new topic, as illustrated in Figure 3.

For some reason you won't see the whole list of items you've entered within the grid until the length of your list of items approaches the height of the grid. When this happens, a scroll bar down the right side will pop up and allow you to see your entire list. This seems to be just a quirk of CFGRID. Once the input form is done, it's time to program the action template.

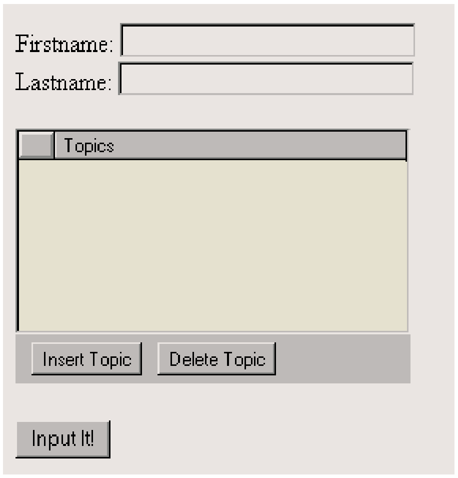


Figure 2: Input form

Input Action Template

Inserting the FORM.Firstname and FORM.Lastname fields into the Main table is pretty straightforward:

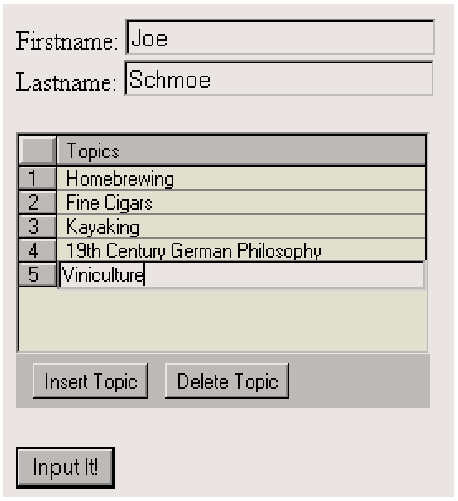
```
<cfquery datasource="experts"name="input-
Experts">
INSERT INTO MAIN(Lastname, Firstname)
VALUES('#FORM.lastname#', '#FORM.first-
name#')
</cfquery>
```

Next, topics from the grid will be inserted into the Topics table via the CFGRIDUPDATE tag:

```
<cfgridupdate
 datasource="experts"
 tablename="Topics"
 grid="insertGrid"
 >
```

This code inserts each topic into the Topics table with a default ExpertID of zero (which is the default value for a numeric field in MS Access).

Now we have a few problems. First, if we try to run this action template, we'll get an error message because we've mapped a one-to-many relationship in our database design and enforced referential integrity between the Main table and the Topics table. When the CFGRIDUPDATE tag attempts to insert our topics into the Topics table, it does so with a default ExpertID of zero, breaking our referential integrity rule. So the first thing we must do is dive back into our database relationships and delete the relationship between the Main table and the Topics table. This doesn't mean that the relationship doesn't exist conceptually and logically – it just means that we're not going to have our database force referential integrity on us for the purposes of this application. (Don't worry, the logic of our application will ensure that referential integrity is preserved, as you'll see later.)



Firstname: Joe

Lastname: Schmoe

	Topics
1	Homebrewing
2	Fine Cigars
3	Kayaking
4	19th Century German Philosophy
5	Viniculture

Figure 3: Inserting new topics

The second problem is that, because the new records in the Topics table are being inserted with ExpertIDs of zero, they don't map to the proper ExpertID in the Main table. First, you have to determine the ExpertID of the record we just inserted into the Main table, then update our newly inserted records in the Topics table so that the values in the two ExpertID fields correspond.

The following code determines the ExpertID for the record we just inserted into the Main table. It assigns that value to the "theID" variable:

```
<cfquery datasource="experts"
name="getID">
SELECT  MAX(ExpertID) AS theID
FROMMAIN
</cfquery>
```

Now that we know the value of the ExpertID field in the Main table, we can update the records in the Topics table that we just inserted so the value in their ExpertID fields correspond. This is done with a simple update statement:

```
<cfquery datasource="experts" name="set-
ExpertID">
```

RSW

www.rswsoftware.com

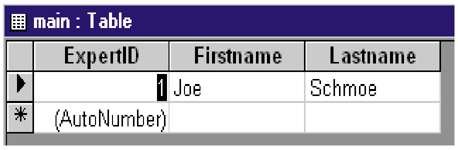
```
UPDATE TOPICS
SET
 expertID = #getID.theID#
WHERE expertID = 0
</cfquery>
```

Now the ExpertIDs of our records in the Topics table match the ExpertIDs of the record in the Main table. And all is well with the world…almost.

Suppose that just before we determined the ExpertID from the Main table, someone else inserted another record. We might mistakenly think that the ExpertID that our records in the Topics table need is this new ExpertID, coming from a completely different record in the Main table. Likewise, suppose we've determined our true ExpertID from the Main table, but just before we go to update the ExpertIDs in the Topics table someone inserts several records. Our update would then include not only our records, but their records as well. In short, we need some way to ensure that all of the above code executes as a monolithic block and the database is "frozen" while we're interacting with it. Luckily, ColdFusion provides us with <cftransaction></cftransaction> tags that provide just this sort of functionality. The final code can be found in Listing 2.

In summary, the action template (1) inserts a new record into the Main table, (2) inserts a variable number of topics into the Topics table, (3) determines the ExpertID of the newly inserted record in the Main table and (4) updates the ExpertID field of the newly inserted topics in the Topics table with this value. Because it does this all within CFTRANSACTION tags, the relationships and referential integrity of the database are preserved.

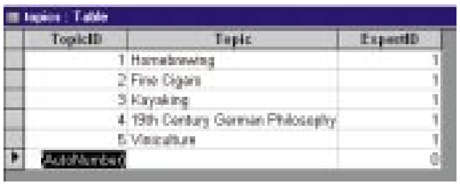
Figure 4 displays the contents of the Main table, while Figure 5 displays the contents of the Topics table after the Input Action template fires.



main : Table

	ExpertID	Firstname	Lastname
▶	1	Joe	Schmoe
*	(AutoNumber)		

Figure 4: Main table



topics : Table

	TopicID	Topic	ExpertID
	1	Homebrewing	1
	2	Fine Cigars	1
	3	Kayaking	1
	4	19th Century German Philosophy	1
	5	Viniculture	1
▶	(AutoNumber)		0

Figure 5: Topics table

Update Form

The Update Form for this application will be similar in structure to the Input Form, but, like most update forms in CF, will be populated by queries from the database, as seen in Listing 3.

The getMainRecords query retrieves records from the Main table that match our

LISTING 1

```
<cfform action="inputaction.cfm">
Firstname: <input type="text" name="firstname" size=25><br>
Lastname: <input type="text" name="lastname" size=25>
<p>
<cfgrid
 name="insertGrid"
 width="250"
 height="160"
 selectmode="EDIT"
 insert="YES"
 insertbutton="Insert Topic"
 delete="YES"
 deletebutton="Delete Topic"
 >
<cfgridcolumn
name="topic"
header="Topics"
headeralign="LEFT"
dataalign="LEFT"
bold="No"
italic="No"
display="Yes"
headerbold="No"
headeritalic="No"
>
</cfgrid>
<p>
<input type="submit" value="Input It!">
</cfform>
```

LISTING 2

```
<cftransaction>

<cfquery datasource="experts" name="inputExperts">
INSERT INTO MAIN(Lastname, Firstname)
VALUES('#FORM.lastname#', '#FORM.firstname#')
</cfquery>

<cfgridupdate
 datasource="experts"
 tablename="Topics"
 grid="insertGrid"
 >

<cfquery datasource="experts" name="getID">
SELECT  MAX(ExpertID) AS theID
FROMMAIN
</cfquery>

<cfquery datasource="experts" name="setExpertID">
UPDATE TOPICS
SET
 expertID = #getID.theID#
WHERE expertID = 0
</cfquery>

</cftransaction>
```

LISTING 3

```
<cfquery datasource="experts" name="getMainRecords">
SELECT   *
FROM MAIN
WHERE Firstname = 'Joe' AND
  Lastname  = 'Schmoe'
</cfquery>

<cfloop query="getMainRecords">
<cfform action="updateaction.cfm">
<cfoutput>
<input type="hidden" name="expertid" value=#expertid#>
Firstname: <input type="text" name="firstname" size=25
value='#firstname#'><br>
Lastname: <input type="text" name="lastname" size=25
value='#lastname#'>
</cfoutput>
<p>
<cfquery datasource="experts" name="getTopics">
SELECT   *
FROM TOPICS
WHERE ExpertID = #getMainRecords.ExpertID#
</cfquery>
<cfgrid
 name="updateGrid"
 query="getTopics"
 width="250"
 height="160"
 selectmode="EDIT"
 insert="YES"
 insertbutton="Insert Topic"
 delete="YES"
 deletebutton="Delete Topic"
```

CODE LISTING

The complete code listing for this article can be located at
www.ColdFusionJournal.com

criteria. In this case, the query is hard-coded and simply selects the record for Joe Schmoe, but your query will probably take its criteria from a set of FORM variables.

A CFLOOP is then opened that loops through the results of the getMainRecords query, building an update form for each record. Within each form there is an HTML input box of type "hidden" that will pass the ExpertID – the key – to the Update Action template.

Inside this main loop is another query, which grabs the topics from the Topics table based on the current value, from the getMainRecords query, of ExpertID. The getTopics query is then used to populate the grid.

Besides being populated by a query, another thing to note about the update grid is that instead of a single Topics column, it includes a hidden TopicID column. Just like a hidden field in an HTML form, a hidden field can be included within a CFGRID that will allow behind-the-scenes passing of a value. In this case we're passing along the TopicID value, a value that the Web client shouldn't have the opportunity to edit, but that is nevertheless crucial to the update function because it is the primary key of the topic record. Without this value the CFGRIDUPDATE in the Update Action template wouldn't know which topic to update.

Once the Update Form is populated, the Web client can insert, update or delete any of the topics in the grid.

Update Action Template

The Update Action template is actually much simpler than the Input Action template because the ExpertID has already been mapped between the Main and Topics tables. Thus, all that needs to be done is to update the Main table and let CFGRID do the updating to the Topics table, as seen in Listing 4. It's as simple as that.

Deleting Records

Deleting records from this application requires issuing SQL DELETE statements:

```
<cftransaction>

<cfquery datasource="experts">
DELETE FROM MAIN
WHERE ExpertID = 1
</cfquery>

<cfquery datasource="experts">
DELETE FROM TOPICS
WHERE ExpertID = 1
</cfquery>

</cftransaction>
```

These two queries delete all records from the Main and Topics tables that meet the specified criteria – in this case, that have ExpertIDs of 1. Your criteria will probably be taken from FORM variables, not hard-coded like the example above. (Also, using a more sophisticated SQL statement, you can delete records from more than one table simultaneously.)

Conclusion

If a simple gridlike structure will satisfy your layout and design needs, the CFGRID Java applet that ships with ColdFusion can be used to create easy-to-use and attractive subforms. When inline frames are supported in both of the main browser platforms, another more sophisticated technique for generating subforms may become prevalent, with ColdFusion and JavaScript code dynamically interacting with a CF action template inside the inline frame. But for now, CFGRID should do the trick for most simple subforms. CFDJ

About the Author

Mark Cyzyk is a Web developer for Johns Hopkins University and Nine Web, LLC, where he uses ColdFusion, JavaScript, Cascading Stylesheets and advanced HTML to create data-driven, Web-based applications.

mcyzyk@nineweb.com

Virtualscape

www.virtualscape.com

CFDJnews

Macromedia Flash and Shockwave Players Available

(San Francisco, CA) – Macromedia, Inc., has included Flash and Shockwave players with Microsoft Internet Explorer 5. As a result, Internet Explorer 5 users can experience distinctive Web sites featuring animated user interfaces, interactive demos, vector-based Web graphics, games, cartoons and more. Now, Macromedia Flash and/or Shockwave players are delivered with every major Web browser, operating system, online service and Web television service.

Macromedia recently reported that Flash has a reach of more than 100 million Web users. Innovative Web sites such as Pepsi Cola, Volkswagen, CNN Financial Network, E*Trade, Absolut Vodka and Reebok use Flash and Shockwave to attract, engage and retain new and current customers. Internet Explorer 5 is the newest version of Microsoft's leading browsing technologies, designed to save users' time by providing a dramatically faster, easier to use, and more flexible Web experience.

For more information visit www.macromedia.com. CFDJ

PSINet Enhances E-commerce Solutions

(Herndon, VA) – PSINet, Inc., a commercial Internet service provider, has enhanced its electronic hosting services. The integrated services combine Web hosting, transaction payment systems and merchant storefront capabilities. Each component is available separately for businesses that do not require the complete solution.

PSINet's standard offering for electronic commerce has added the award-winning INTERSHOP3 e-commerce hosting platform from INTERSHOP Communications Inc.

Visit www.psinet.com. CFDJ

Verio and NetObjects Join Forces

(Englewood, CO) – Verio Inc. and NetObjects, Inc., have joined forces in an effort to build, enhance and host Web sites for small and mid-sized businesses.

Under the agreement, Verio will offer NetObjects Fusion 4.0 customers 30 days of free Web hosting and domain-name registration. Verio will also offer NetObjects Fusion 4.0 and NetObjects Personal Edition products to its Web-hosting customer base at a discounted price. In return, NetObjects will promote Verio's Web-hosting services through its own site, in-box marketing materials and other initiatives.

For more information you can visit www.verio.com and www.netobjects.com. CFDJ

CyberCash and Concentric Network Deliver E-commerce Solutions

(San Jose, CA) – Cybercash, Inc., and Concentric Network Corp. have signed an agreement that will enable Concentric to deliver a completely integrated solution designed to make it quick and easy for small to medium-sized businesses to establish their own e-commerce Web sites and to use Cybercash for accepting secure, real-time payments. Concentric is the first commerce service provider to directly resell CyberCash services for processing Internet credit card transactions.

For more information visit www.cybercash.com. CFDJ

Live Software Unveils Third Beta Release

(Cupertino, CA) – Live Software is proud to announce the third beta release of its <CF_Anywhere> product. <CF_Anywhere> brings powerful multi-platform support to Allaire's ColdFusion markup language (CFML). <CF_Anywhere> allows for the running of standard CFML pages on any platform that JRun supports, including UNIX (HP-UX, Irix, AIX, Solaris x86, Solaris Sparc), Linux, Novell, Macintosh, and Windows 9x/NT. CF_Anywhere 3.1 supports the ColdFusion 3.1 tags with 4.x support to follow.

For more information visit http://cfanywhere.com. CFDJ

Allaire and Compaq Form New Partnership

(Houston, TX) – Compaq Computer Corporation has announced a new solutions development and joint marketing partnership with Allaire Corporation. Compaq also announced that Allaire has selected Compaq as their Microsoft Windows NT and Linux reference platform for critical product development and quality assurance.

For more information visit www.allaire.com or www.compaq.com. CFDJ

Intermedia.NET Helps Clients Build Easy to Maintain Sites with ColdFusion 4.0

(Palo Alto, CA) – Intermedia.NET, a Web-hosting provider supporting ColdFusion 4.0 and custom tag registration, has announced a new site, www.OurMIB.org (Our Men In Black), built with ColdFusion. This site, from the Catholic priests of New Ulm, Minnesota, is built to share information about vocations in a question-and-answer format.

An administration portion of www.OurMIB.org enables the priests, who are not trained in HTML, to manage ongoing administration of the site. They post answers to the submitted questions and send these Q&As out to the people who have joined the mailing list, all without knowing any HTML or programming. ColdFusion server events handle all the back-end functions.

For more information visit www.intermedia.net. CFDJ

Allaire

www.allaire.com

More Punch with <CFQUERY>

The challenge: To allow site visitors to text-search a database and display results in a table

by Jerry Bradenbaugh

Welcome to the first edition of <CFBASICS>. The concept behind this article is to show you some techniques of ColdFusion by focusing on a small group of CF tags or functions, usually just one or two. After each article, you'll have another trick to add to your bag. Not cheap tricks, mind you, but techniques that help you solve common business requirements for Web sites. Sounds cool, doesn't it? Let's dispense with the rhetoric and get to work.

A common challenge facing Web coders is enabling site visitors to perform text searches against a database and display matching results in a table. ColdFusion's <CFQUERY> tag allows you to do this. This article demonstrates how you can use a single <CFQUERY> to easily search a database and even custom-sort the results. Figure 1 shows the sample site I created. The database is written in MSAccess and contains one table (named Customers) with only five fields, less than a hundred records. All the data comes from the sample Northwinds Trade database found in a number of Microsoft software distributions. Users can search by company name, contact name, contact title and country.

Not much to the interface. You're looking at a frameset with two frames. The document in the left frame contains a simple HTML form. Users simply enter search text, select a category to search and then choose Go! Results are listed in the larger frame to the right in Figure 1. Figure 2 has the results of searching for all the records with the string "USA" in the Country field.

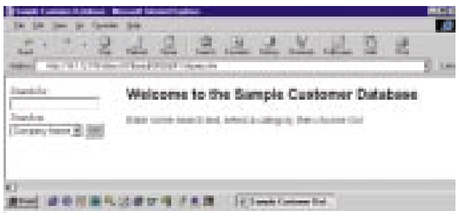
This looks simple enough. Let's see how ColdFusion pulls it off (Listing 1). First, the code for the HTML form: it's all in select.cfm. This file contains pure HTML, but putting it in a .cfm file makes it easy to add CF later.

The important thing to note here is the name of the FORM elements. The TEXT field is named QueryText, and the SELECT list is named Category. Notice, also, that the METHOD attribute is set to "get." That means all the FORM information will be passed in through the URL query string (the part after the ? in a URL, e.g., www.sample.com/index.cfm?Day=Monday). Now suppose the user has entered search text, chosen a category and slammed on Go! The ACTION attribute of select.cfm is set to search.cfm, which calls search.cfm. Let's see how that looks (Listing 2).

That's a lot bigger than select.cfm, but it's still pretty simple. Think about it – each time it's called, this script performs one of two operations:

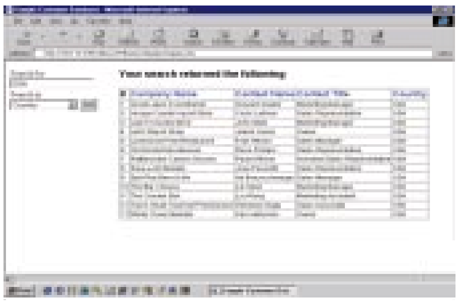
1. Displays a cheap default greeting (see Figure 1)
2. Searches the database and displays matching results

The decision to do one or the other happens after ColdFusion examines the values of several variables. The form element names, Category and QueryText, are created as form variables when they arrive at the script. That means that #Category# and #QueryText# are declared. Remember, though, not every call to search.cfm has a query string. When you first load the application, you're calling search.cfm, not search.cfm?SomeQueryStringText. To make sure these variables have values every time the script is called, we'll create a CFPARAM for each. In fact, that's the first few lines of the script. See for yourself:



Welcome to the Sample Customer Database

Figure 1: Sample database



Your search returned the following:

Figure 2: Search results

```
<CFPARAM NAME="Category" DEFAULT="">
<CFPARAM NAME="QueryText" DEFAULT="">
<CFPARAM NAME="Sort" DEFAULT="Company-
Name">
```

Now #Category# equals an empty string by default, or whatever category the user selected. Variable #QueryText# equals an empty string or whatever text the user entered. If both variables are empty strings, the script was *not* called by select.cfm. If they contain data other than empty strings, the call must have come from the FORM submission. That's how the top-level <CFIF> statement works:

```
<CFIF #Category# NEQ "" AND #QueryText#
NEQ "">
 <!--- Search the database --->
<CFELSE>
 <!—Let’s have the cheap greeting --->
</CFIF>
```

What about #Sort#? It too, has been assigned a default value with <CFPARAM>. #Sort# is used to order the matching results. More on that in a moment. First, let's perform a search with the much anticipated <CFQUERY> tag:

```
<CFQUERY NAME="SearchCustomers" DATA-
SOURCE="CustomerDB">
 SELECT * FROM Customers
 WHERE #Category# LIKE '%#QueryText#%'
 ORDER BY #Sort#
</CFQUERY>
```

Listing 3 contains the ColdFusion syntax reference, straight from the CFML Language Reference Guide. Needless to say, there are more attributes than we'll need, but let's go over a few of them:

- ***Name***: This is required if you're going to query a database. The name must begin with a letter and then can be followed by other letters, numbers and underscores. Sorry, no white spaces. Ours is set to SearchCustomers.
- ***Datasource***: Also required. It tells ColdFusion what ODBC database it will be using. If you're a little hazy about ODBC datasources in ColdFusion, see *Administering ColdFusion Server* in your documentation.

- ***Username and password***: These are both optional and aren't included in our <CFQUERY> tag because this database has no username/password requirements. I mention them here because you might have to restrict access to your data somehow. You can set a username and password in the ODBC datasource of the ColdFusion Administrator, but these attributes allow you to substitute them for anything you include in these tags.

On to the SQL

If you're not familiar with SQL (Structured Query Language), check out www.geocities.com/ResearchTriangle/Node/9672/sqltut.html for a great crash course. If you've done any SQL coding, you'll find the statement used here is pretty simple – here's proof:

```
SELECT * FROM Customers
WHERE #Category# LIKE '%#QueryText#%'
ORDER BY #Sort#
```

This SQL statement means return (SELECT) all the data from each record (*) in the Customer's table (FROM Customers) where the category the user chose (WHERE #Category#) contains within it the text that the user entered (LIKE '#QueryText#'). Notice that #QueryText# is surrounded by single quotes. This indicates that variable #QueryText# will be compared to the field as a character data type as opposed to a numeric or a date. Notice the % signs as well. Those are the SQL wild cards. If you've ever searched for files in Windows, you may have used *.txt to find all filenames with a .txt extension. The % is the CF equivalent of *.

The last "thing" the SQL statement does is to order the results in ascending order. As you can see from Listing 2, the default for #Sort# is CompanyName, which is one of the fields in the database. We'll do something really cool with this shortly. For now, assume that the results are sorted by company name.

From the example in Figure 2, the SQL would look as follows (the bold represents the values of #Category# and #QueryText#, respectively):

```
SELECT * FROM Customers
WHERE Country LIKE '%USA%'
ORDER BY CompanyName
```

Once <CFQUERY> queries the database, it's time to print the records back to the waiting browser. All we need to do is construct a TABLE and neatly add the record information in each row. The <CFOUTPUT> tag helps here. Check it out in Listing 4.

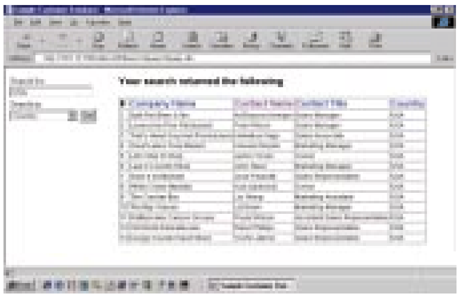


Figure 3: Sorting by contact title

The QUERY attribute identifies which record set to use for the output. There's only one in this script, SearchCustomers. All you have to do is create a table row with as many data cells as you need. In this case we use five: a record number (not in the database), the company name, the contact name, the contact title and the country. The Customers table in the database contains five fields: ID (not used in this application), CompanyName, ContactName, ContactTitle and Country.

#SearchCustomers# represents each matching record from the database table. Fields in each record are referenced by name, surrounded by pound signs (e.g., #CompanyName#). <CFOUTPUT> iterates through all the records in SearchCustomers and prints one row of HTML for each. #SearchCustomers.CurrentRow# is an index of each record in the query starting with 1 (then 2, 3, etc,). It is used here as an index for the results.

The Cool Gets Cooler

That lone <CFQUERY> tag enabled a basic free text search. That's great, but it would also be nice to custom-sort the search results. By default, the results are sorted by company name. What if you want to sort them by contact name, contact title or even country? The good news is that you can use the same <CFQUERY> tag to pull it off. Notice in Figure 2 that the results from Figure 1 have been sorted by contact name. That is, the results are in alphabetical order according to the contact name. Figure 3 shows the sort by contact title.

When I say *sort*, I mean arrange in ascending order – basically, alphabetically (A to Z, 0 to 9). The thing is, it doesn't apply as you might think for numbers. In other words, the number 1 comes before 2, 3, 4–9. However, so do 10, 100, 1000, because those numbers begin with a 1.

You can sort the results by any of the four fields by clicking on any of the table headings at the top. Think about how you create this type of functionality. You need three things to pass to the SQL statement in the <CFQUERY> tag:

1. The category you originally chose
2. The text you originally entered
3. The field by which you want to sort

If you could call search.cfm with these three things, you could query the database and custom-sort the results to your whim. When you call the file from the search form in select.cfm, you're already providing the in search.cfm with two of the three: the category and the text. The third is provided for you the first time around (#Sort# is set to CompanyName). All you need is that field you want to search. To accommodate, check out the remaining code in search.cfm located in Listing 5. The bolded code represents the field used for sorting.

After the search, ColdFusion begins the results display by returning a row of table headers containing a link in each one (except for the first). Each link call is a call to search.cfm with a slightly different query string. The name-value pairs of Category=#Category# and QueryText=#QueryText# provide the same data as the original call from the search for in select.cfm. The links in the table headers go one step further by adding the Sort=some_field name-value pair. One sets Sort equal to CompanyName, another to ContactName, another to ContactTitle, the last to Country. Now have another look at the <CFQUERY> tag:

```
<CFQUERY NAME="SearchCustomers" DATA-
SOURCE="CustomerDB">
 SELECT * FROM Customers
 WHERE #Category# LIKE '%#QueryText#%'
 ORDER BY #Sort#
</CFQUERY>
```

The records are arranged according to the value of #Sort#. Notice that #Sort# has a default value of "CompanyName," so calls to search.cfm without Sort in the query string (e.g., from select.cfm) will have the results sorted by CompanyName. The <CFOUTPUT> tags take care of printing the results. Not bad for a few lines of code.

Well, that does it for this article. Make sure you download all the code. You'll get the three .cfm files and the .mdb Access file with the data. By the way, make sure you let me know if you like what you read. If you want to see something here, drop me a line.

CODE LISTING

The complete code listing for this article can be located at **www.ColdFusionJournal.com**

About the Author

Jerry Bradenbaugh, a senior Web application developer in Los Angeles, is also the Webmaster of Hotsyte – The JavaScript Resource (www.serve.com/hotsyte). He can be reached at hotsyte@mail.serve.com.

hotsyte@mail.serve.com

Connecting ColdFusion to LDAP

The nitty-gritty of a CF tag

by **David Anderson**

When I started using ColdFusion, one of the tags that piqued my curiosity most was <CFLDAP>. At the time, I had only a vague idea what LDAP was and when I started to write applications that extended beyond single pages, I realized I needed some kind of authentication. I tried the <CF_LOGIN> tag but wasn't thrilled with the idea of storing usernames and passwords in a table.

At the same time, I was setting up Netscape's Messaging Server. While it can use a local file for user information, it prefers an LDAP directory. I immediately saw the advantage of using the LDAP server for both Messaging Server and my ColdFusion applications. So I set about putting up Netscape's Directory Server.

I quickly learned that LDAP directories are significantly different from SQL databases. An LDAP directory is a specialized database for storing information about people and places. As such, the field names, called attributes, are predefined. Although the actual database structure is fairly flat, the directory itself is treelike. The tree metaphor is used frequently in the LDAP world: branches are where the tree splits, leaves are the terminal entries off a branch.

Populating my LDAP directory proved to be a trial by fire as I learned how to use the <CFLDAP> tag. I used ColdFusion to populate the directory and to maintain synchronization between it and the Oracle database it's based on.

LDAP Basics

LDAP queries differ from SQL queries. For example, search information is passed as parameters. While the format for the query is different, the data usually comes back from the server formatted like an SQL query. I say "usually" because it's possible for a given attribute to have multiple values. These are returned as comma-delimited lists.

LDAP directories use a fixed list of column names called attributes, which are defined in the LDAP specification. In addition to the attributes listed in the *Advanced ColdFusion Development* book, you can find a more complete list at www3.innosoft.com/ldapworld/rfc1617.txt.

Like <CFQUERY>, <CFLDAP> handles all interactions with the LDAP directory. You'll use it for querying, inserting, updating and deleting information from the directory. ColdFusion does some error-checking, but more advanced functions are simply passed directly to the directory server. You'll have to rely on the directory server logs for detailed information about bad LDAP operations.

Custom CF LDAP Tags

A check of the Developer's Exchange turns up two tags for working with LDAP: LDAP_Group and LDIFToQuery. Of the two, the LDAP_Group has proven the most useful. It allows you to maintain directory groups via ColdFusion, rather than the UNIX commands or the server console.

LDIFToQuery will convert LDAP Directory Interchange Format files into queries. LDIF is generally used for bulk loading and unloading a directory server.

Querying an LDAP Directory

Building a basic query is fairly simple. You'll need to provide the name of your directory server, attributes you want, start and query names. The example below will get and print all the distinguished names (dn) at the United States branch in the University of Michigan's LDAP server:

```
<cfldap name="orglist"
 server="ldap.itd.umich.edu"
 attributes="dn"
 start="c=US">
<cfoutput query="orglist">
#dn#<br>
</cfoutput>
```

The equivalent search in SQL would be:

```
select dn
from ldap
where c = 'US'
```

Filters, the rough equivalent of SQL's WHERE, can be used to narrow the query. The syntax for filters is a bit tortured. To select, for instance, just the State University of New York entries in the example above, you'd add this:

```
 filter="(o=State University of New
York*)"
```

Implementing more complex filters is an interesting challenge. For instance, if I want all the people whose last names fall between A and D, I'd use a filter like:

```
 filter="(|(sn=A*)(sn=B*)(sn=C*)(sn=D*))"
```

In this example, the | ORs the four subfilters, so it'll return everything from A to D. You can combine filters from different attributes and nest them as much as you'd like. The more complex your filter is, the longer it'll take the LDAP server to process. Filter performance is affected by attribute indexing. If you plan to search frequently on a particular attribute, make sure the directory server indexes it.

SCOPEing the Query

The examples above look for entries at the level you specify in the START option. In this case we've been looking at entries that fall immediately below the c=US portion of the directory. If there are entries under lower branches, we won't see them. To get to those entries, we need to use SCOPE.

SCOPE can have one of three values: onelevel, base and subtree. The default, onelevel, will go down the directory exactly one branch. That's why we get the organization entries. Base will return entries only at the base level, which is the same as the start.

Subtree, the most productive of all three, will traverse down all branches under the START option, looking for entries that match your filter. If you're not sure where an entry might be, subtree is the easiest way to find it. Since subtree searches the entire LDAP directory, it can take a little longer.

Sorting Queries

The ability to sort your queries is a bit limited. You can only sort on one field. For instance, to sort by last name add SORT= "sn ASC" to your query. The LDAP server will happily return the list to you, sorted by last name. Because names are stored in the commonname (cn) attribute in a human-readable manner, SORT="cn ASC" will return a list sorted by first name.

Able Solutions

Enter the realm of browsable store building and administration – from your browser. Build "your_site.com" with secure Merchant Credit Card Processing. Maintain inventory, add discounts and specials to keep your customers coming back. Increase sales with cross selling and membership pricing.

11700 NE 95th Street, Suite 100, Vancouver, WA
www.ablecommerce.com • 360 253-4142

Backsoft Corp.

BackTalk for SAP Web application server is the fastest way to build and deliver scalable applications that integrate browser, server and SAP database technologies. BackTalk for SAP is the first web-enabled solution for the SAP R/3 system which uses Allaire's ColdFusion Technology. It leverages the scalability of Allaire's ColdFusion Architecture and builds an intuitive development tool to Web-enable SAP.

5971 Cattleridge Blvd, Suite 101, Sarasota, FL 34232
www.backsoft.com• 941 378-2325

BiznizWEB inc

DYNAMIC WEB PUBLISHING

Internet Publishing

Big Bucks? Heavy Web Expertise?
Operations Nightmare?
It doesn't have to be that way!

www.webpublishingtools.com

ad server poll

recruitment directories

local portals city guides calendars classifieds

Catouzer, Inc.

With the Synergy 1.0 Web application framework, creating custom intranet applications is a breeze. The Synergy Application Development Kit (ADK) gives you the tools to rapidly develop your custom applications, which can be fully integrated and managed under the Application Services Layer (ASL).

1228 Hamilton Street, Suite 501, Vancouver, B.C. V6B 2S8, Canada
www.catouzer.com • 604 662-7551

Data Return Corporation

Data Return offers extensive support for customers utilizing ColdFusion from Allaire. With customers delivering over 50,000 user sessions per day, we know how to provide high availability solutions for this advanced application server. Our technical support staff has extensive experience in coding custom ColdFusion Tags as well as managing Microsoft SQL server. We also support CyberCash for customers interested in real-time credit card processing. Our combination of support and experience offer an ideal environment for deploying your applications developed for ColdFusion.

801 Stadium Dr., Ste 117, Arlington, TX 76011
www.datareturn.com • 800 767-1514

Eprise Corporation

If your customers are looking for a content management solution, Eprise Participant Server FastStart Kit for Allaire ColdFusion Developers can save you time and resources. Participant server is a flexible content management framework that enhances high-value business relationships through the delivery of timely, targeted, Web-based communications.

1671 Worcester Road, Framingham, MA 01701
www.eprise.com • 800 274-2814

Fusion FX, Inc.

Fusion FX, Inc., is a full-service Web site design and hosting company. Our strength in ColdFusion Development, as well as with custom AbleCommerce Store building, and the fact that we have a Team Allaire member on staff will give you peace of mind because you know your ColdFusion development is in good hands.

819 Peacock Plaza, Suite 535, Key West, FL 33040
www.fusionfx.com • 800 780-2422

The Igneous Group, Inc.

The Igneous Group provides custom solutions for e-commerce business information systems and dynamic content publishing. We build systems to your specifications and integrate them with your existing backoffice data and business infrastructure. We can help you get your site online, or identify how to make it function more efficiently.

541 Seabright Avenue, Santa Cruz, CA 95062
www.igneous.com • 877 469-ROCK

Highlight your website with

ib infoboard

NT and UNIX
Cold Fusion Hosting

Development Consulting

Oracle, Informix, MS-SQL, E-Commerce Plug-ins

<allaire> Alliance Partner

1-800-514-2297
sales@infoboard.com
www.infoboard.com

Interland Inc.

Interland Inc. is a worldwide leader in Web hosting and Web site design. Over 19,000 customers around the world have chosen Interland to host their Web sites, and over 1,700 new customers join every month. Our mission is to provide you with great features, reliable technology, fast and knowledgeable support, low prices, and marketing advantages to make your Internet Web site a success.

One Park Tower, 34 Peachtree Street, 14th Floor, Atlanta, GA 30303
www.interland.net • 800 323-4532

Intermedia, Inc.

Our advanced virtual hosting packages (powered by Microsoft Windows NT and Internet Information Server 4.0) offer an environment supporting everything today's advanced Web developer or sophisticated client could ask for. Complete ODBC support is available on plans B and C. We support Microsoft Index Server on all hosting plans.

953 Industrial Avenue, Suite 121, Palo Alto, CA 94303
www.intermedia.net • 650 424-9935

Live Software

The power of Java and the simplicity of ColdFusion, <CF_Anywhere> give ColdFusion programmers the ability to leverage all the power and flexibility of Java using the familiar ColdFusion Markup Language (CFML). <CF_Anywhere> was perfected for CF developers, CF administrators, ISPs and everyone.

5703 Oberlin Drive, Suite 208, San Diego, CA 92121
www.cfanywhere.com • 408 996-0300

Macromedia Corp.

Macromedia is the leading developer of cross-platform software tools for digital media creation and publishing. Macromedia's products are used by organizations to create and deliver interactive applications that use a full range of media, from text and graphics, to animation. The product line includes Director, Authorware, ShockWave, FreeHand, and Dreamweaver.

600 Townsend Street, San Francisco, CA 94103
www.macromedia.com • 415 252-2000

RSW Software

RSW Software is a wholly owned subsidiary of Teradyne, Inc., and specializes in Web application testing software. Established with the goal of providing best-in-class testing products, RSW offers a suite of products called the e-TEST Suite, which automates the process of testing business-critical Internet and intranet applications.

44 Spring Street, Second Floor, Watertown, MA 02172
www.rswsoftware.com • 508 435-8000

Sitehosting.NET

Successful electronic commerce starts at SiteHosting.net; a division of Dynatek Infoworld, Inc., which provides total Web development services. We offer personal and efficient customer service with reliability at value prices. All our plans include access to SSL (Secure Socket Layer). We support ColdFusion, Active Server Pages, Real Audio/Video, Netshow Server, and more. Our hosting price starts at $14.95/month.

13200 Crossroads Parkway North, Suite 360, City of Industry, CA 91746
www.sitehosting.net • 877 684-6784

Virtualscape

Why host with Virtualscape? Nobody else on the Internet understands what it takes to host ColdFusion like we do. Virtualscape is the leader in advanced Web site hosting. From Fortune 500 extranets to e-commerce sites and more, developers recognize our speed, stability, reliability and technical support.

215 Park Avenue South, Suite 1905, New York, NY 10003
www.virtualscape.com • 212 460-8406

Yesler Software

You don't have to be an HTML programmer to create media-rich e-shows for your Web site. Use Yesler Alive's templates and prompts to quickly create your own slides, add audio, video and graphics, then synchronize it all as easily as clicking your mouse button. Need to put it on the intranet? Or the Web site? Yesler Alive could well be the most time and cost effective communication tool you'll ever own.

83 S. King St., Ste 414, Seattle, WA 98104
www.yesler.com • 206 674-7700

Other LDAP Peculiarities

Postal addresses are stored in the postalAddress attribute in an LDAP directory. The individual lines of a postal address are separated by a $. My work address, for instance, would be Plattsburgh SUNY$Computing Support$Feinberg Library$Plattsburgh, NY 12901. You could use ListToArray or a <CFLOOP> to properly output the list. The example below will print an address properly:

```
<cfif len(postaladdress) gt 0>
   <cfloop index="addr" list="#postalAd-
dress#" delimiters="$">
    <cfoutput>#addr#</cfoutput><br>
   </cfloop>
</cfif>
```

The standard for postal addresses specifies a maximum of 30 characters in six rows.

Other problems to watch out for are timeouts, maximum number of records and access controls lists (ACLs). The default timeout for queries through ColdFusion is 60 seconds. You can specify a longer timeout; however, you may run into the timeout of the server. There's no maximum number of records ColdFusion will ask for. You can specify a MAXROWS amount if you think your query may return an inordinately large result set. Most servers have an internal limit on the maximum number of entries they'll return. You may want to check the server settings.

Access control lists are a way of limiting who can access what information. For instance, the default ACLs in Netscape's Directory Server prevent a user from changing someone else's password and anonymous users from making any changes. You can implement ACLs that prevent certain fields from being returned or return only those fields to specific users. If you know the field exists and the data's there, chances are an ACL is preventing you from seeing it.

ColdFusion queries your LDAP server as an anonymous user. You can, and probably should, set up a user account in the LDAP directory that your CF server can use. You can give that account permission beyond that of an anonymous user but less than the directory administrator. You'll need to include the USERNAME and PASSWORD options in your <CFLDAP> queries.

Going Beyond Simple Queries

Okay. Now that you know how to query an LDAP directory, let's add a record. This is where things start to get really hairy. You have to think carefully about what information you want to store in the directory before you do it. Most LDAP server manuals have detailed guides for implementing a directory structure.

Actually, adding the record is relatively straightforward. You'll need to come up with a distinguished name that matches the one used in your directory server. Next, build the list of required object classes and the attributes list. To actually add the entry, you'll need a username that has sufficient permissions.

In Listing 1, I'm adding Rick Deckard to the Los Angeles Police Department directory. There's no real minimum to the information you must add other than the dn. Your directory server may do syntax checking to make sure you don't add attributes without the appropriate object classes, so it's always a good idea to make sure everything's okay before adding an entry.

In composing the attribute list to add an entry, the object classes come first, followed by a semicolon, then the attributes and their values. Each attribute is separated by a semicolon as well. I build the object class list and the attribute list separately because I sometimes need to add object classes based on the type of user. For instance, if one of my users has an e-mail account, I can easily add the appropriate object classes and attributes.

Modifying Existing Records

Modifying a directory entry is similar to an add. You'll need to construct a list of attributes, all separated by semicolons. If you're adding any object classes, you'll need to construct a new object class list. While it makes sense to us to just pass the additional object classes, most LDAP servers require the entire object class list. The program in Listing 2 will update Deckard's entry in the LAPD directory server.

Rather than trying to figure out what the dn of an entry should be, I do a query for the record I want to modify. I'll use the dn later in the modify query. The additional hit against the directory server doesn't slow down the process significantly.

In the program I use to keep my directory server in sync with our Oracle database, I'll query the Oracle database and do a lookup and modify for each entry. Even with an update of over 6,000 records, my program will usually run in just a few minutes.

Changing the DN

You may find at some time that you need to change the dn for an entry. In setting up my directory server, I initially used a number for the uid portion of the dn. After populating the database, I realized the folly of my ways and wrote a program to change the uid to something more reasonable: the individual's username on our mainframe. Listing 3 changes Deckard's uid from his last name and first initial to his BRU number.

When changing the dn, you can generally change only the leftmost portion. For instance, the dn for me is:

```
uid=andersdl, ou=people,
o=plattsburgh.edu
```

Using the modifydn query, only the uid=andersdl portion can be changed. It's not possible to move entries around in the tree structure using modifydn. Instead, you'll have to delete the entry and re-create it under the new branch.

Also, you can't use modifydn to change the name of a branch. If you want to change the ou=people branch to ou=employees, you have to remove all the entries under people, then rename it.

Deleting Attributes from an Entry

While you can delete portions of an LDAP entry at the UNIX command line, it's not really possible with ColdFusion. Instead, I do an update and set the values of the fields to null. Listing 4 removes Deckard's manager.

If you've heavily modified an LDAP entry and are concerned that it may contain many unused attributes, simply delete the entire record and re-add it.

Deleting an Entry

If you know the dn, removing an entry requires only one query. To delete the entry, set the query type to "delete" and provide the dn. As with any other query that modifies the directory, you'll need the proper user permissions to delete the entry. Listing 5 deletes Deckard's entry.

The delete operation removes only one entry at a time and will not remove leaf entries. If you need to remove a range of entries, you'll have to loop over the list and do a delete query for each. To delete a branch that has leaf entries, remove all its leaf entries first.

Tying It Together

Your directory server information can be used for the same purposes you'd use an SQL database. Because directory servers aren't SQL servers, there are some caveats.

As I mentioned above, directory queries can be sorted on only one field. If you need to sort on last and first name, you'll need to write some code to postprocess the query.

In some of my applications I use groups on the server to grant access to applications. In certain applications I'll generate a select list from the group. Since group queries come back from the server as comma-delimited lists, they require some conversion before they can be used for further queries. In my application I'll get the list of unique members, then build a filter that'll return the users I want. The code below will take a list of uids and convert it to a filter:

```
 <cfset uidq = "(|">
 <cfloop index="value"
list="#userlist.uniquemember#" delim-
iters=",">
 <cfif left(trim(value),"3") eq "uid">
   <cfset uidq = uidq & "(" &
trim(value) & ")">
 </cfif>
 </cfloop>
 <cfset uidq = uidq & ")">
```

One minor frustration is the inability to select attributes using AS, such as you can do in SQL. If you need the equivalent, you can certainly use the query functions (QueryNew, QueryAddRow, QuerySetCell) to reformat the directory. I've found it convenient when working with data from the directory server that's similar to a query from my Oracle server.

Using LDAP for Authentication

For me, the primary purpose of putting up a directory server was authentication. We're trying to reduce the number of passwords our users have to remember. At the same time, we're trying to expand the number of services we offer. Netscape's Directory Server, with its ability to synchronize Windows NT accounts, has proved to be a good fit for us.

I've configured my ColdFusion server to use the directory server for authentication. If you've skimmed the manual, you've probably noticed it can get complicated. Essentially, you can use your directory server as the source for user authentication. I won't go into how to set up security. Ben Forta's book, *Advanced ColdFusion Application Development*, covers setting up security rather well.

The only difficulty I've run into has been in adding and removing users and groups via the user administration screens in the CF Administrator. This is a known bug in the current version of ColdFusion Server. I have to type in the dn for users and groups that I want added. Unfortunately, removing users means I have to re-create my user lists.

Final Thoughts

Just as with any datasource, it pays to know as much about your directory schema as you can. Knowing what fields are used in your directory and the overall structure of the tree can help immensely in formulating queries. Knowing how LDAP servers work can also be informative. You may want to look at some of the LDAP Web pages listed below.

If you're just starting out with directory servers, try to set up a small test machine. A couple of free LDAP servers run on Linux, so if you have an extra machine, you might want to set it up as your testbed. (By the way, you may have noticed Barbara [Babs] Jansen used in many LDAP examples. If you don't know who she is, search the Internet Movie Database [www.imdb.com] for Martha Smith.) CFDJ

References

LDAP FAQ:
www3.innosoft.com/ldapworld/ldapfaq.html
LDAP Roadmap & FAQ:
www.kingsmountain com
ldapRoadmap.shtml
Lightweight Directory Access Protocol:
www.umich.edu/~dirsvcs/ldap/index.html
Netscape Directory Service Docs:
http://home.netscape.com/eng/server/
directory/4.0/

CODE LISTING

The complete code listing for this article can be located at
www.ColdFusionJournal.com

About the Author

David Anderson is the Web site administrator at Plattsburgh State University, New York. He's been playing with ColdFusion for about a year and refuses to use any other tool for Web pages. You can reach him at david.anderson@plattsburgh.edu.

david.anderson@plattsburgh.edu

Advertiser Index

Advertiser			Page
Able Solutions	www.ablecommerce.com	360-253-4142	2
Allaire Corporation	www.allaire.com	888-939-2545	21, 31, 43
BackSoft	www.backsoft.com	888-222-6047	13
BizNiz Web	www.webpublishingtools.com	281-367-4016	47
Catouzer Inc.	www.catouzer.com	604-662-7551	51
Data Return	www.datareturn.com	800-767-1514	15
Eprise Corporation	www.eprise.com	800-274-2814	33
FusionFX	www.fusionfx.com	800-780-2422	23
The Igneous Group	www.igneous.com	877-469-ROCK	49
Infoboard	www.infoboard.com	800-514-2297	47
Interland	www.interland.net	800-323-4532	4
Intermedia.net	www.intermedia.net	650-424-9935	52
Live Software	www.cfanywhere.com	408-996-0300	19
Macromedia	www.macromedia.com	800-457-1774	3, 26-27
RSW Software	www.rswsoftware.com	508-435-8000	39
Sitehosting.net	www.sitehosting.net	888-463-6168	37
Virtualscape	www.virtualscape.com	212-460-8406	41
Yesler Software	www.yesler.com	206-674-7700	9

Igneous

www.igneous.com

Building Enterprise Portals

Finally, a way to think about how complex technology and business tie together cohesively

by Jeremy Allaire

Every year I find myself contemplating the dramatic changes in the Internet industry over the previous year. And every year the changes seem more dramatic, more exciting – and, most important, clearer. Everyone involved in the Internet industry does the same thing, I'm sure, and as part of this ongoing reflection we try and find meaning in a few major concepts to help us grapple with all the change and opportunity. These concepts typically end up in buzzwords that we internalize and then attempt to indoctrinate our peers (and customers) with this new understanding.

For example, in 1993, "the Internet" meant "Online Services," which meant AOL, CompuServe and Prodigy. This was about as complex an idea as people could come up with, and given how vague and overwhelming it all seemed, that was probably okay.

So it was for the next six years. Each year we in the industry tried to figure out what it was all about, building products and services to support it, and, most important, giving it a name, or a concept. We need (and needed) these keywords and concepts, which we used to map complex technology to business or consumer reality.

In 1994 and 1995 it was the "Internet," which meant "wild-wild-west," "experimentation" and "new opportunities," but that's about it. After people (including many of you) began to really work with the technology, they figured out there were opportunities beyond building brochure-ware Web sites. They saw there were real opportunities in automating a business and leveraging the incredible cost economies of the Web computing model, and in reaching new customers. Thus they were dubbed "e-commerce" and "intranets." People finally understood this computing model wasn't about brochures, but about changing how information and products were used, created and sold. In fact, for quite some time, most companies focused exclusively on intranets and the unbounded opportunities that lay therein.

Sometime in late 1996, and into 1997, the concept of the "extranet" emerged. This was a site that was NOT an Internet site, but also NOT an intranet site; still deployed on the Internet, but secured like an intranet. Whoa! But it did have meaning, and that meaning was that you could use the Internet to tie together companies and organizations in a way that wasn't possible before.

Since 1997 and through 1998, we've taken these concepts and run with them. We've built products and services that enable these terms and concepts and we use them fluidly in selling what we do to our customers.

During this time some pretty significant things have been happening in our industry, all of which bode well for our future. First, corporations are finally "figuring out" the value of the Web. After years of experimentation, working with early adopter technology and through the inevitable pressures of customer feedback and competitive threats, many corporations are now at a point to fully embrace the Web business and computing platform throughout their business, even reinventing their business around this new economy. Second, we've seen the spectacular success of the most visible Internet companies, the e-commerce portals and "dot com" players. We're seeing this, aspiring and learning, and realizing that the earth is moving beneath us. And finally, all of this hard work and learning is establishing best-practices in both the technology infrastructure and the business and organizational models. Indeed, these three trends, happening over the past year or so, indicate that we're on the verge of a massive mainstream explosion in adoption of the Internet.

Out of all this, of course, must come a new organizing concept, one that ties together everything we've learned and becomes the basis for how the mainstream thinks about and creates the next wave of Internet business. I believe the organizing concept we'll come to know, speak and understand is the "Portal." Everywhere I go I hear customers talk about building a "Family Portal" or a "Kids Portal," and, most recently, and perhaps most significantly, an "Enterprise Portal." Increasingly, it seems, "Portal" has come to mean "successful Internet site" or "successful Internet business," and no longer carries its original meaning, "search engine." I strongly believe that the Enterprise Portal is the right organizing concept for furthering our work.

Understanding Enterprise Portals

An Enterprise Portal is the combination of software and technology infrastructure, new business models and new organizational structures that combine to create an Internet-centric business. In short, an Enterprise Portal is what companies need to build in order to become Internet-centric companies. In my estimation, Enterprise Portals reflect three distinct observations about the Internet business landscape.

The first is the realization that truly Internet-centric companies don't view their Internet, intranet and extranet systems as separate. Instead, they see the pervasiveness of the Web simply as a part of their business. An Enterprise Portal represents this idea. Your Web systems become a portal to your entire business -- internally, for your employees managing their work, then extending out through private and secure interfaces to customers, suppliers and partners. The Enterprise Portal allows us, for the first time, to think of the Web as the fabric of our business, and recognizes the reality that Web systems aren't about isolated corporate applications, but about an overall approach to doing business in the Internet economy. Indeed, with Enterprise Portals the Web becomes your business, and your business becomes the Web.

A second observation is that Enterprise Portals, by being modeled on the best-practices of "dot com" companies, pave the way to understanding the four broad solution components in running an Internet-centric business: rich content, e-commerce, customer interaction management and collaboration. All successful portals center around a rich-content application infrastructure, including models for dynamic publishing, workflow, roles-based security models and content asset management. Likewise, they tie in commerce systems, including Web transaction management infrastructure, as well as common systems for merchandise management and order processing. Finally, they bridge these components with applications that enhance end-user and customer experience, such as personalization, user forums and collaboration tools for document management, threaded discussions and managing projects over the Web.

The third key observation is that Enterprise Portals represent the technology and business best-practices in the Web environment. They demonstrate that there are well-observed user interface models, known systems and development architectures, and – most important – Web-native business and organization models. This shift toward best-practices–based approaches is critical as mainstream companies look to scale their Web efforts to enterprise-wide levels.

Conclusion

Enterprise Portals give us a model for our businesses. At Allaire it's driving us to build a comprehensive platform that spans visual tools, application servers and packaged systems for building and managing an Enterprise Portal. For corporate customers it provides a model and call to arms to think about their business and technology strategy in the Internet age. For solution companies it should drive new solutions based on best-practices in Enterprise Portals.

In any case, we have a new mantra, a new buzzword and, finally, a way to think about how these complex technology and business issues tie together in a cohesive manner, driving forward the next generation of the Internet economy. CFDJ

About the Author

Jeremy Allaire is a cofounder and vice president of technology strategy at Allaire. He helps determine the company's future product direction and is responsible for establishing key strategic partnerships within the Internet industry. Jeremy has been a regular author and analyst on Internet technologies for the past seven years, and he holds degrees in both political science and philosophy from Macalester College.

jeremy@allaire.com

Catouzer

www.catouzer.com

Intermedia

www.intermedia.net